Engineering Fracture Mechanics 247 (2021) 107638

Contents lists available at ScienceDirect

Engineering Fracture Mechanics

journal homepage: www.elsevier.com/locate/engfracmech

A complex-variable cohesive finite element subroutine to enable efficient determination of interfacial cohesive material parameters

Daniel Ramirez-Tamayo [a,b], Ayoub Soulami [b], Varun Gupta [d,1], David Restrepo [a], Arturo Montoya [a,c], Harry Millwater [a,*]

[a] Department of Mechanical Engineering, The University of Texas at San Antonio, San Antonio, TX, USA
[b] Pacific Northwest National Labs, Richland, WA, USA
[c] Department of Civil and Environmental Engineering, The University of Texas at San Antonio, San Antonio, TX, USA
[d] ExxonMobil Upstream Research Company, Spring, TX, USA

ARTICLE INFO

Keywords:
Complex-variable finite element method
Complex Taylor series expansion
Inverse determination of material parameters
UEL
Automatic differentiation

ABSTRACT

A new complex-variable version of a cohesive element is presented that provides highly accurate first order derivatives of the nodal displacements with respect to the cohesive fracture parameters. These sensitivities are provided as a byproduct of the analysis using the complex Taylor series expansion method. This information is useful for inversely determining the cohesive fracture parameters from experimental or synthetic data using a finite element-based approach. In particular, the PPR cohesive element (Park et al., 2009), was extended using complex variables as a user element for the well-known commercial finite element program, Abaqus. The source code for the element is provided as an educational resource. The advantage of having accurate first order derivatives on both accuracy and efficiency is demonstrated through numerical examples.

1. Introduction

The structural performance prediction of a mechanical joint requires characterization of the interface resulting from the joining process. Cohesive zone modeling (CZM) is a popular approach to investigate fracture, seams, and joints [1]. The CZM requires a cohesive constitutive law which, in the context of a joint, relates the tractions at the interface to the separation displacement of the two surfaces. Some techniques to obtain the parameters for a cohesive law place restrictions on the test geometries and require the existence of analytical solutions [2,3]. In addition, these techniques use a global response (load–displacement curve) to describe a local material property, resulting in uncertainties in the adopted model [4]. That is, different combinations of the cohesive parameters could recreate the global behavior of the joint while the local behavior (near crack fields) is inaccurate. Hence, the uniqueness of the CZM is not guaranteed.

Other techniques such as hybrid inverse techniques [4] use full-field kinematic measurements from a suitable test geometry obtained using the Digital Image Correlation (DIC) procedure and an inverse finite element analysis to identify the cohesive parameters by solving an optimization problem. Valoroso et al. [5] used an experimental data set consisting of the measured load–deflection curve and crack length to inversely determine the mode-I cohesive parameters of a bonded interface using a double cantilever beam test. In their optimization algorithm, derivatives of the nodal displacements and reaction forces with respect to

* Corresponding author.
E-mail addresses: daniel.ramirez@my.utsa.edu (D. Ramirez-Tamayo), ayoub.soulami@pnnl.gov (A. Soulami), varun.gupta@exxonmobil.com (V. Gupta), david.restrepo@utsa.edu (D. Restrepo), arturo.montoya@utsa.edu (A. Montoya), harry.millwater@utsa.edu (H. Millwater).
[1] Work was performed while still at PNNL.

https://doi.org/10.1016/j.engfracmech.2021.107638
Received 1 October 2020; Received in revised form 6 February 2021; Accepted 23 February 2021
Available online 1 March 2021

Nomenclature

a_0	Beam initial crack length
B	Beam width
E	Elastic modulus
f	Function
$\mathbf{f}_e$	Elemental load vector
$\mathbf{f}$	Global load vector
h	Perturbation step size along the imaginary axis
H	Beam arm height
i	Imaginary direction
$\mathbf{K}_e$	Elemental stiffness matrix
$\mathbf{K}$	Global stiffness matrix
L	Beam length
L_P	Beam distance to applied load
m, n	Non-dimensional exponents
$\boldsymbol{P}$	Reaction force
$\boldsymbol{R}_P$	Residual vector of the reaction force
T_n, T_t	Normal and tangential cohesive tractions
$\mathbf{u}$	Global solution vector
$\mathbf{u}^*$	Complex-valued gobal solution vector
w	Residual function to be minimized
α, β	Shape parameters of the PPR model
Γ	Energy constant
δ	Final crack opening width
δ_n, δ_t	Final crack opening width in the normal and tangential directions
δ_{nc}, δ_{tc}	Normal and tangential critical opening displacements
$\boldsymbol{\Delta}$	Separation along the fracture surface
Δ_n, Δ_t	Normal and tangential separations
$\boldsymbol{\epsilon}$	Finite element strain vector
$\boldsymbol{\theta}$	Vector consisting of the material parameters to be optimized
θ	Material parameter of interest
λ_n, λ_t	Initial slope indicators in the PPR model
ν	Poisson's ratio
$\boldsymbol{\sigma}$	Finite element stress vector
σ_{max}, τ_{max}	Normal and tangential cohesive strengths
ϕ_n, ϕ_t	Fracture energy components in the normal and tangential directions
Ψ	Potential function for cohesive fracture
Ω	Finite element domain
Subscript c	Cohesive finite element domain
Subscript e	Elastic finite element domain
Subscript Im	Imaginary component of a complex variable
Subscript n	Normal direction
Subscript Re	Real component of a complex variable
Subscript t	Tangential direction
Superscript COMP	Denotes computational results
Superscript EXP	Denotes experimental results
Superscript *	Denotes a complex variable
CR	Cauchy–Riemann
CTSE	Complex Taylor series expansion
CZM	Cohesive zone modeling

the maximum traction and fracture energy were computed using a custom version of the finite element code FEAP [6] with direct differentiation capabilities. Direct differentiation requires extensive chain rule formulations and source code modifications and is usually limited to first order derivatives [7–13]. This method is efficient but time consuming to implement, and not practical for

DIC	Digital image correlation
DOF	Degrees of freedom
FD	Finite difference
PPR	Park–Paulino–Roesler (Names of the authors that proposed the PPR cohesive law)
RF	Reaction force
RHS	An array containing the contributions of this element to the right-hand-side vectors of the overall system of equations
SLSQP	Sequential least squares programming
SVARS	Solution-dependent state variable
UEL	User-defined element
zCoords	Complex version of the coordinates
ZFEM	Hypercomplex finite element method
ZPPR	Complex-variable version of the PPR cohesive element
zU	Complex version of the nodal displacements
$\langle\cdot\rangle$	Macaulay bracket
$\mathrm{Re}[\cdot]$	Real component of a complex variable
$\mathrm{Im}[\cdot]$	Imaginary component of a complex variable

large-scale finite element programs that contain a large variety of element formulations. Shen and Paulino [4] used experimental DIC data from an edge-notched specimen and retrieved the bulk material properties and the mode-I parameters that govern the traction-separation law. These authors used a Nelder–Mead [14] gradient-free optimization algorithm which is known to have a slower convergence rate than gradient-based algorithms.

A key ingredient in all these problems is the existence of a finite element code containing a cohesive element formulation. Several numerical implementations of the cohesive element have been proposed [15–25]. Park and Paulino [26] presented a computational implementation of the potential-based cohesive zone model (PPR) into a commercial finite element software, in particular using a user-defined element (UEL) subroutine within the Abaqus software [27]. They provided the source code to the UEL to facilitate understanding and implementation of the element formulation and its capabilities. This source code provides the basis for our effort to develop a complex-variable UEL in order to compute highly accurate derivatives of the displacements with respect to the cohesive material properties. In this paper, derivatives with respect to the CZM parameters will be computed using the complex Taylor series expansion method. These gradients can then be used in gradient-based optimization algorithms for the determination of the specific numerical values of variables that govern the traction-separation law for any particular application.

The hypercomplex finite element method, ZFEM, incorporates the complex Taylor series expansion method (CTSE) [28] to compute highly accurate estimates of arbitrary shape, material property or loading sensitivities without the step size issues associated with finite differencing — complex variables provide first order derivatives, hypercomplex variables provide mixed and higher order derivatives. The use of this method allows existing codes to compute derivatives in a way that is conceptually simple and straightforward to implement, and once the code is "complexified", the derivative to compute is determined from the input file. By augmenting the finite element variables to complex type and perturbing the variable of interest along the imaginary axis, the user gains the capability to compute highly accurate derivatives. This method has been demonstrated for linear elastic fracture mechanics [29,30], elastic plastic fracture mechanics [31], thermoelastic fracture [32], mixed-mode loading and interface cracks [33]. All of these applications have been implemented through the use of a user element subroutine in Abaqus [27]. This paper explains and demonstrates how to implement CTSE into an existing finite element code. In particular, the application of this approach has been demonstrated by implementing the complex version of the PPR cohesive element in Abaqus as a user element subroutine [26]. The resulting complex element is hereafter referred to as the "ZPPR" element.

The paper is organized as follows. First, the methodologies for the cohesive zone element (CZM) and the CTSE method are briefly outlined. Section 2.3 discusses the specifics of how to implement ZFEM into a finite element code, in particular, the commercial software Abaqus through the use of a user element subroutine (UEL). Modifications required to the UEL and input file are shown for a particular example. Numerical examples are provided and the results are compared against other numerical solutions. Then, concluding remarks are provided. The UEL source code and input file are provided in the Appendix for the numerical example that will be discussed throughout the paper.

2. Background and methodology

2.1. Cohesive zone modeling

The cohesive zone model has been used to approximate the non-linear fracture process for a wide variety of engineering problems. It was first presented by Barenblatt [7] and Dugdale [34], and has since been used to investigate fracture of quasi-brittle materials [35], concrete [36], delamination of adhesive joints [20] and debonding [37], among others. Several numerical

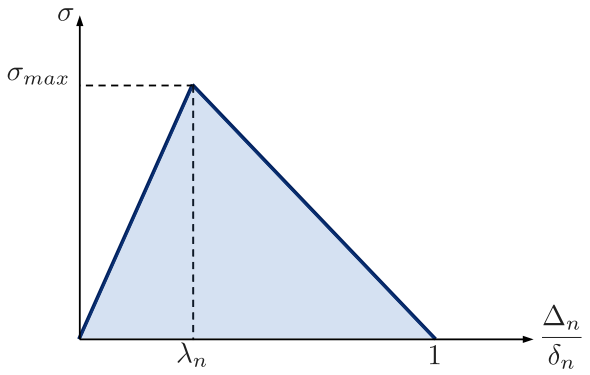

Fig. 1. Linear softening traction-separation law.

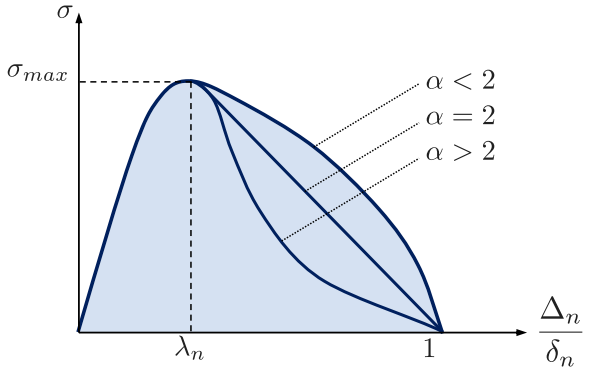

Fig. 2. Potential-based traction separation law.
Source: Adapted from [26].

implementations of the cohesive model have been presented [21,25,26,38]. A critical aspect in cohesive zone modeling is the selection of the traction-separation relation, which can lead to a variety of structural response and failure behaviors. Traction separation laws can be broadly categorized into potential-based and nonpotential-based responses.

For nonpotential-based models, several shapes of the traction-separation law have been proposed such as linear softening [39], trapezoidal [18], bilinear softening [17], trilinear softening [22] and exponential [24]. The simplest traction-separation response is the linear softening model where the behavior is defined by three parameters: the cohesive strength (σ_{max}), the critical opening displacement (λ_n), and the fracture energy (ϕ_n), which is the area under the $\sigma - \Delta$ curve, see Fig. 1, where Δ_n is the normal separation which is normalized by the final crack opening width in the normal direction, δ_n. This model has been implemented into the commercial finite element software Abaqus [19,27]. However, as shown by Park et al. [40], nonpotential-based models can provide non-physical behavior for certain separation paths because the model does not always provide a negative tangent stiffness within the softening region.

To address this issue, potential-based models, initially proposed by Xu and Needleman [37], use a cohesive energy potential in which the derivative of this potential describes the traction-separation behavior for the softening region. Several potential-based models have been proposed [15,41–44]. Amongst these models, the PPR model [15] (Park–Paulino–Roesler) has several advantages over other models as was shown in [45]. In this reference, the authors performed an extensive review of traction-separation relationships for cohesive models (potential- and nonpotential-based) and the limitations of other models are listed and compared against the PPR model. The PPR model contains shape parameters (α and β) that allow the user to select different behaviors within the softening region which can be used to characterize different material responses such as brittle and quasi-brittle. Fig. 2 shows a schematic of the mode-I traction-separation law of the PPR model for different values of α. When $\alpha = 2$, the PPR model exhibits an almost linear softening behavior.

The cohesive traction-separation relationship in the PPR model is obtained from a fracture potential given by

$$\Psi(\Delta_n, \Delta_t) = \min(\phi_n, \phi_t) + \left[\Gamma_n \left(1 - \frac{\Delta_n}{\delta_n}\right)^{\alpha} \left(\frac{m}{\alpha} + \frac{\Delta_n}{\delta_n}\right)^{m} + \langle \phi_n - \phi_t \rangle\right] \times \left[\Gamma_t \left(1 - \frac{|\Delta_t|}{\delta_t}\right)^{\beta} \left(\frac{n}{\beta} + |\Delta_t|\,\delta_t\right)^{n} + \langle \phi_t - \phi_n \rangle\right] \tag{1}$$

Table 1
PPR parameters.

Condition	Description
Complete normal failure	$T_n(\delta_n,\ \Delta_t) = 0$
Complete tangential failure	$T_t(\Delta_n, \delta_t) = 0$
mode-I fracture energy	$\int_0^{\delta_n} T_n(\Delta_n, 0)\, d\Delta_n = \phi_n$
mode-II failure energy	$\int_0^{\delta_t} T_t(0, \Delta_t)\, d\Delta_t = \phi_t$
Normal cohesive strength	$T_n(\delta_{nc}, 0) = \sigma_{max}$
Tangential cohesive strength	$T_t(0, \delta_{tc}) = \tau_{max}$

where Δ is the separation along the fracture surface, Γ is the energy constant, δ is the final crack opening width, α and β are the shape parameters, m and n are non-dimensional exponents, and ϕ is the fracture energy. The subscripts "n" and "t" denote normal and tangential directions, respectively.$\langle . \rangle$ is the Macaulay bracket, i.e.,

$$\langle x \rangle = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{2}$$

In this cohesive model, four fracture parameters categories are employed for each fracture mode: fracture energies (ϕ_n, ϕ_t), cohesive strengths (T_n, T_t), shape parameters (α, β), and initial slope indicators (λ_n, λ_t). The PPR model satisfies the conditions stated in Table 1.

T_n and T_t are the normal and tangential cohesive tractions, and δ_{nc} and δ_{tc} are the normal and tangential critical opening displacements which denote the start of the softening region. The derivatives of the PPR potential model (Eq. (1)) with respect to the normal and tangential separations, Δ_n and Δ_t, respectively, lead to the normal and tangential cohesive tractions, $T_n(\Delta_n, \Delta_t)$ and $T_t(\Delta_n, \Delta_t)$. Once the final crack opening width is reached, either in the normal (δ_n) or tangential (δ_t) direction, complete separation occurs. The area under the traction separation curve is known as the fracture energy, with its components in the normal and tangential directions, ϕ_n and ϕ_t, respectively. With these values and the slope indicators (m, n, α and β), it is possible to recreate the nonlinear fracture process using a cohesive model. For further references, please refer to [26,46].

2.2. Complex-variable finite element method, ZFEM

ZFEM calculates sensitivities with respect to variables of interest based on a Complex Taylor Series Expansion (CTSE) method [28]. In a ZFEM analysis, the perturbation is applied to the variable of interest, $X = x_0$, along the imaginary axis, to become, $X = x_0 + ih$, where x_0 is the original real part, i denotes the imaginary direction, and h is the step size along the imaginary axis. Using CTSE, a function f can be expanded as

$$f(x_0 + ih) = f(x_0) + f'(x_0)\frac{ih}{1!} + f''(x_0)\frac{(ih)^2}{2!} + f'''(x_0)\frac{(ih)^3}{3!} + \text{H.O.T} \tag{3}$$

where "H.O.T" stands for higher order terms. Taking the imaginary part of both sides, and dividing by h yields

$$\frac{\text{Im}[f(x_0 + ih)]}{h} = f'(x_0) + \mathcal{O}(h^2) \tag{4}$$

The error induced by the higher order terms can be reduced to below machine precision with the use of a very small h, e.g. 10^{-10} times the variable of interest. As a result, highly accurate first order derivatives of f with respect to the variable of interest can be computed as

$$\left.\frac{\partial f}{\partial x}\right|_{x=x_0} \approx \frac{\text{Im}[f(x_0 + ih)]}{h} \tag{5}$$

CTSE is equivalent in concept to the finite differencing approach except that the perturbation is applied along the imaginary axis. In CTSE, the derivative information can be obtained in a single run and the step size issues are circumvented if a sufficiently small h is used. This is in contrast to the finite difference method which requires at least two analyses and the determination of the appropriate step size is usually problematic. Obtaining derivatives in a finite element analysis requires a complex-variable finite element method that allows perturbations along the imaginary axis. This will be discussed in the following section. CTSE has been applied in a number of engineering fields such as shape sensitivity analysis [47], linear, non-linear and mixed mode fracture mechanics [29,31,33], aerodynamics [48], computational fluid dynamics [49], creep [50], structural dynamics [51,52], among others.

2.3. Complex-variable finite element implementation

In this section, the required changes to modify the PPR element to a complex-variable form, the ZPPR element, are described. In particular, the specifics of implementing the ZPPR element into Abaqus [27] are presented such as: modifications to the real-valued user element subroutine (UEL), and needed changes to the input file for complex nodes. While changes specific to the PPR element are described, similar changes can be applied to convert any element to complex-variable form for implementation within Abaqus as a UEL. If the finite element software lacks a complex-variable solver, a Cauchy–Riemann (CR) representation of a complex-variable can be used to address this issue as will be demonstrated below. In addition, higher order derivatives can be obtained using hypercomplex algebra [52–54]. The term "ZFEM" is used to describe a complex- or hypercomplex-variable finite element method.

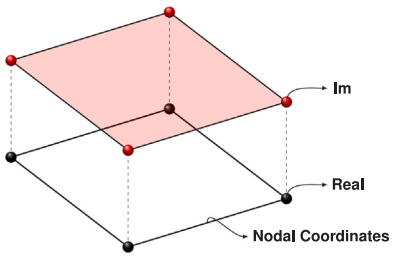

Fig. 3. 8-noded (4 real and 4 imaginary) element. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2.3.1. ZPPR element Abaqus implementation

The ZPPR element has been implemented in Abaqus through the complexification of a real-valued user element subroutine (UEL) [29–33,50,55]. In a real-valued UEL, the elemental stiffness matrix, $\mathbf{K}_e$, and the load vector, $\mathbf{f}_e$, are computed and returned to Abaqus for assembly and solution of the global system of equations as

$$\mathbf{Ku} = \mathbf{f} \tag{6}$$

where $\mathbf{K}$, $\mathbf{u}$ and $\mathbf{f}$ are the global stiffness matrix, displacement vector, and force vector, respectively.

In ZFEM, the finite element displacement degrees of freedom are now complex-valued where the real part is the conventional finite element variable and the imaginary part contains the derivative information with respect to the variable of interest. Therefore, a duplicate set of nodes is used to store the derivative information at each nodal point and, in the case of a shape perturbation analysis, define a perturbation of the real spatial coordinates. Fig. 3 shows the nodes from a conventional 4-noded element (black) augmented by 4 imaginary nodes (red). The result is a 8-noded (4 real and 4 imaginary) complex element with a total of 16 degrees of freedom for which the solution of a complex-variable system of equations is required.

Given the fact that Abaqus does not have a built-in complex solver, a Cauchy–Riemann (CR) representation of a complex-variable is employed to return a real-valued stiffness matrix to Abaqus. In summary, using the CR representation, a complex number $a^* = a+bi$ can be represented as a matrix of all real numbers as

$$a^* = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \tag{7}$$

where * denotes a complex variable. This approach can be applied at the element stiffness level before returning to Abaqus for assembly. The result is a $2N \times 2N$ system of equations, where N is the number of real degrees of freedom. The additional runtime using ZFEM depends upon the analysis type. For an analysis using the ZPPR element with a symmetric solver, the additional computational time is approximately 50% of a real analysis. Runtimes for other analysis types can be found in [30–32]. Eqs. (8) and (9) show the complex system of equations and its CR representation used in a ZFEM analysis

$$\mathbf{K}^*\mathbf{u}^* = \mathbf{f}^* \tag{8}$$

$$\begin{bmatrix} \mathbf{K}_{\mathrm{Re}} & -\mathbf{K}_{\mathrm{Im}} \\ \mathbf{K}_{\mathrm{Im}} & \mathbf{K}_{\mathrm{Re}} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\mathrm{Re}} \\ \mathbf{u}_{\mathrm{Im}} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_{\mathrm{Re}} \\ \mathbf{f}_{\mathrm{Im}} \end{bmatrix} \tag{9}$$

where the subscripts "Re" and "Im" denote the real and imaginary components of a complex variable, respectively. Numerically, the imaginary component, $\mathbf{K}_{\mathrm{Im}}$, is the derivative of $\mathbf{K}_{\mathrm{Re}}$ with respect to the parameter of interest, θ, times the step size, i.e., $\mathbf{K}_{\mathrm{Im}} = (\partial \mathbf{K}_{\mathrm{Re}}/\partial\theta) \times h$. Similarly, $\mathbf{f}_{\mathrm{Im}} = (\partial \mathbf{f}_{\mathrm{Re}}/\partial\theta) \times h$, and $\mathbf{u}_{\mathrm{Im}} = (\partial \mathbf{u}_{\mathrm{Re}}/\partial\theta) \times h$. After the CR system of equations is solved, the real part of the displacement vector, $\mathbf{u}_{\mathbf{Re}}$, contains the nodal displacements as in a real-variable analysis, and $\mathbf{u}_{\mathbf{Im}}$ contains the derivatives of the displacement with respect to the perturbed variable times the step size. Other post-processing quantities and their derivatives such as strains, stresses and potential energy can be computed using traditional equations but using the complex variable nodal displacement vector.

2.3.2. Formulation of the User Element Subroutine (UEL)

In this section, the formation of the CR form of the complex-variable stiffness matrix, $\mathbf{K}^*$, the solution vector, $\mathbf{u}^*$, and right hand side vector, $\mathbf{f}^*$, for the ZPPR element are defined. The changes required to complexify the real-variable PPR element (and, in general, any real-variable UEL) are: (i) redefine specific UEL variables as complex type, (ii) perturb the variable of interest through the input file, and (iii) convert the complex variables to CR format before exiting the UEL for assembly and solution of the system of equations. All of these changes will be explained in detail in this section. In this paper, only derivatives with respect to the cohesive parameters will be computed. Hence, for the bulk (4-noded linear elastic) elements, a traditional real-valued UEL will be used. The only modification required for these elements is that before exiting the UEL, the real-valued stiffness matrix has to be expanded to CR format with zero contributions to the imaginary parts.

Listing 1: UEL variable definitions

```
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
! Select double precision Real
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!

INTEGER, PARAMETER :: DP = SELECTED_REAL_KIND(15, 307)

!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! Parameters Useful for Calculations

PARAMETER  ZERO=0.0_DP, ONE=1.0_DP, NONE=-1.0_DP, HALF=0.5_DP, TWO=2.0_DP ! Zero, one, negative one,
    half, two
PARAMETER  gaussCoord=SQRT(3.0_DP)/3.0_DP, gaussWeight=1.0_DP, ninpt=2    ! For Gaussian integration

!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Variables used within the cohesive UEL

! Complex Variables

COMPLEX(DP), DIMENSION(mcrd*nnode/2,mcrd*nnode/2)  :: Sc       ! Element stiffness matrix of a cohesive
     element
COMPLEX(DP), DIMENSION(mcrd*nnode/2,nrhs)          :: Fc       ! Cohesive internal force vector
COMPLEX(DP), DIMENSION(mcrd,nrhs)                  :: T        ! Cohesive traction vector
COMPLEX(DP), DIMENSION(mcrd,mcrd)                  :: T_d      ! Derivative of the cohesive traction (
    Tangent matrix)
COMPLEX(DP), DIMENSION(mcrd, mcrd)                 :: Arot     ! Coordinate transformation matrix
COMPLEX(DP), DIMENSION(mcrd*nnode/2, mcrd*nnode/2) :: R        ! Rotation Matrix
COMPLEX(DP), DIMENSION(mcrd*nnode/2)               :: U_l      ! Nodal displacement in the local
    coordinate system
COMPLEX(DP), DIMENSION(mcrd,mcrd*nnode/2)          :: Bc       ! Global displacement-separation
    relation matrix
COMPLEX(DP), DIMENSION(mcrd)                       :: del      ! Normal and tangential separations
COMPLEX(DP), DIMENSION(nnode/2)                    :: del_l    ! local nodal displacement jumps
COMPLEX(DP), DIMENSION(mcrd*mcrd, mcrd*nnode/2)    :: L        ! local displacement-separation relation
     matrix
COMPLEX(DP), DIMENSION(mcrd*nnode/2, 1)            :: Fcoh     ! Cohesive Traction
COMPLEX(DP), DIMENSION(mcrd*nnode/2, mcrd*nnode/2) :: Kcoh     ! Stiffness Matrix
COMPLEX(DP), DIMENSION(mcrd, nnode/2)              :: zCoords  ! Complex Coordinates
COMPLEX(DP), DIMENSION(nnode/2*ndofel/2)           :: zU       ! Complex Displacement

COMPLEX(DP) :: G_n,   G_t          ! Normal and Tangential Fracture Energies
COMPLEX(DP) :: Tn_m, Tt_m          ! Normal and Tangential Cohesive Strength
COMPLEX(DP) :: alph, beta          ! Shape Parameters
COMPLEX(DP) :: l_n,   l_t          ! Slope Indicators
COMPLEX(DP) :: dn, dt              ! Final crack opening widths
COMPLEX(DP) :: m, n                ! Exponents to compute Potential Function
COMPLEX(DP) :: Gam_n, Gam_t        ! Energy constants, related to the fracture energies
COMPLEX(DP) :: dGnt, dGtn          ! Macauley bracket <G_n-G_t> or <G_t-G_n>, respectively
COMPLEX(DP) :: del1, del3          ! Nodal separations in x (Local Coords)
COMPLEX(DP) :: del2, del4          ! Nodal separations in y (Local Coords)
COMPLEX(DP) :: deln_max, delt_max ! Maximum displacements jumps during load history
COMPLEX(DP) :: el_length           ! Length of the cohesive element
COMPLEX(DP) :: dvol                ! Volume Differential

! Real and complex parts of the cohesive stiffness matrix
REAL(DP), DIMENSION(mcrd*nnode/2, mcrd*nnode/2) :: Kreal, Kimag
! Real and complex parts of the RHS vector
REAL(DP), DIMENSION(mcrd*nnode/2,1) :: Freal, Fimag

REAL(DP), DIMENSION(2,nnode/2) :: shapeN   ! Shape function matrix
REAL(DP), DIMENSION(2)         :: GP       ! Gauss points
REAL(DP), DIMENSION(2)         :: GP_w     ! Weight at the Gauss points
REAL(DP)                       :: N1, N2   ! Shape Functions

REAL(DP) :: h                 ! Perturbation step size
REAL(DP) :: th                ! Thickness
INTEGER  :: perturbation_flag ! Perturbation flag
INTEGER  :: I                 ! Counter
INTEGER  :: nn_real           ! Number of real nodes
INTEGER  :: ndof_real         ! Number of real DOF
```

Table 2
ZPPR element properties passed to the Abaqus UEL.

PPR properties	
UEL property ID	Property
1	Element type
2	Normal fracture energy, ϕ_n
3	Tangential fracture energy, ϕ_t
4	Normal cohesive strength, σ_{max}
5	Tangential cohesive strength, τ_{max}
6	Normal shape parameter, α
7	Tangential shape parameter, β
8	Normal slope indicator, λ_n
9	Tangential slope indicator, λ_t
10	Thickness of the element, t
11	Perturbation step size, h
12	Perturbation flag

Table 3
Linear elastic element properties passed to the Abaqus UEL.

Linear elastic element properties	
UEL property ID	Property
1	Element type
2	Elastic modulus, E
3	Poisson's ratio, ν
4	Thickness of the element, t

The first modification performed to the UEL is on the variable definition section. The following lines of Fortran code (Listing 1) show the variable definitions of the complex variable ZPPR UEL. Note that not all the variables need to be complexified, some variables such as the shape functions and the Gaussian integration quadrature remain the same as in the real-valued UEL. For the matrix variables, the only required modification is on the variable type, that is, change REAL to COMPLEX. The dimensions of the complex-valued cohesive stiffness matrix (Kcoh) are given by the variables "nnode/2=4" and "mcrd=2". "nnode" and "mcrd" are Abaqus UEL variables that are defined through the input file as will be shown in Section 2.3.3. "nnode" is the number of nodes per element and "mcrd" is the number of dimensions or degrees of freedom per node. For the real-valued 2D PPR element nnode=4 and mcrd=2; for the ZPPR element nnode=8 (4 real and 4 imaginary nodes) and mcrd remains unchanged. For the scalar variables such as the volume differential, dvol, and the fracture energies, Gn and Gt, no modifications other than changing the variable type to complex are needed. When a Fortran code is provided, a comment or an index from an array is represented with black color, magenta denotes a variable and blue means a Fortran keyword.

Next, through the input file (as will be shown in the following section) the user defines the material parameter θ that will be perturbed along the imaginary axis, yielding derivatives of the nodal displacements with respect to the parameter of interest, $d\mathbf{u}_{\mathbf{Re}}/d\theta$. In this paper, only derivatives with respect to the material properties of the PPR element will be discussed. However, for the ZFEM methodology in general, shape derivatives can be also obtained by perturbing the coordinates of the imaginary nodes as shown in [29–33]. Shape derivatives can be used for shape optimization and fracture mechanics analyses where the energy release rate is computed as the derivative of the potential energy with respect to the crack length.

Listing 2 shows a Fortran case statement where the perturbation is being applied along the imaginary axis of the cohesive material parameter of interest where the "perturbation_flag" variable is passed from the input file. The real variable material values of the PPR elements are passed from the input file in PROPS(1–10) (Table 2). PROPS(11–12) are the perturbation step size and perturbation flag respectively. These properties are only passed for the cohesive elements. For the linear elastic elements, the properties passed to the UEL are shown in Table 3. More details regarding the input file are explained in the following Section 2.3.3. The perturbation step size, h, is set as 1×10^{-10} times the variable of interest.

Two additional changes are needed in the UEL, one in the beginning of the UEL to convert the nodes and displacements from CR to complex form, and one before exiting the UEL to convert the stiffness matrix and load vector from complex to CR form. By default, Abaqus calls the UEL at least twice per increment before obtaining the solution vector for the current increment. The first time is when the elemental stiffness matrix and right hand side vector are computed for each element and then returned for assembly. Once the global stiffness matrix is assembled, Abaqus solves the system of equations and calls the UEL once more to compute the force residual for each element. Post-processing quantities such as strains, stresses and energies are computed in this second pass through the UEL. If a tolerance is satisfied, the same process is repeated for the next increment. Otherwise, the loading increment is reduced until convergence is achieved. As it was explained in Section 2.3.1, the solution of the CR system of equations

returns a vector of the form

$$\mathbf{u}^* = \begin{bmatrix} \mathbf{u}_{\text{Re}} \\ \mathbf{u}_{\text{Im}} \end{bmatrix} = \begin{bmatrix} \mathbf{u}^1_{\text{Re}} \\ \mathbf{v}^1_{\text{Re}} \\ \mathbf{u}^2_{\text{Re}} \\ \vdots \\ \mathbf{v}^4_{\text{Re}} \\ \mathbf{u}^1_{\text{Im}} \\ \mathbf{v}^1_{\text{Im}} \\ \mathbf{u}^2_{\text{Im}} \\ \vdots \\ \mathbf{v}^4_{\text{Im}} \end{bmatrix} \tag{10}$$

where there are a total of 16 degrees of freedom (DOF) per element, 8 real and 8 imaginary. $\mathbf{u}^1_{\text{Re}}$ and $\mathbf{u}^1_{\text{Im}}$ denote the real and complex x-displacements of node number 1. $\mathbf{v}^1_{\text{Re}}$ and $\mathbf{v}^1_{\text{Im}}$ denote the real and complex y-displacements of node number 1, etc. The derivative with respect to the perturbed parameter is obtained as

$$\frac{\partial \mathbf{u}}{\partial \theta} \approx \frac{\text{Im}[\mathbf{u}_{\text{Im}}]}{h}$$

where θ is the perturbed variable and h is the perturbation step size. In order to compute post-processing quantities (strains, stresses and energies) and the force residual in the second pass through the UEL using the intrinsic Fortran complex type, the CR form of the solution vector, $\mathbf{u}^*$, is converted to complex format as

$$\mathbf{u}^* = \text{complex}(\mathbf{u}_{\text{Re}}, \mathbf{u}_{\text{Im}}) \tag{11}$$

Listing 2: Perturbation definition

```
!~~~~~~~~~~~~~~~~~~~~
! Read UEL Properties

perturbation_flag = PROPS(12) ! Defines variable to be perturbed
h                 = PROPS(11) ! ZFEM's perturbation step size

! Define fracture parameters
G_n  = PROPS(2)
G_t  = PROPS(3)
Tn_m = PROPS(4)
Tt_m = PROPS(5)
alph = PROPS(6)
beta = PROPS(7)
l_n  = PROPS(8)
l_t  = PROPS(9)
th   = PROPS(10)

! Apply perturbation to the parameter of interest
SELECT CASE (perturbation_flag)
    CASE (1)
        G_n  = CMPLX(PROPS(2), h*PROPS(2))
    CASE (2)
        G_t  = CMPLX(PROPS(3), h*PROPS(3))
    CASE (3)
        Tn_m = CMPLX(PROPS(4), h*PROPS(4))
    CASE (4)
        Tt_m = CMPLX(PROPS(5), h*PROPS(5))
    CASE (5)
        alph = CMPLX(PROPS(6), h*PROPS(6))
    CASE (6)
        beta = CMPLX(PROPS(7), h*PROPS(7))
    CASE (7)
        l_n  = CMPLX(PROPS(8), h*PROPS(8))
    CASE (8)
        l_t  = CMPLX(PROPS(9), h*PROPS(9))
END SELECT
```

In addition to the solution vector, $\mathbf{u}^*$, the nodal coordinates are converted into a complex-variable format to link the real and complex degrees of freedom as shown in Fig. 4. The offset value, shown in Fig. 4 is a constant value that is greater than the total number of real nodes. The offset is a convenience for mesh generation and results extraction. Any value can be used that is greater

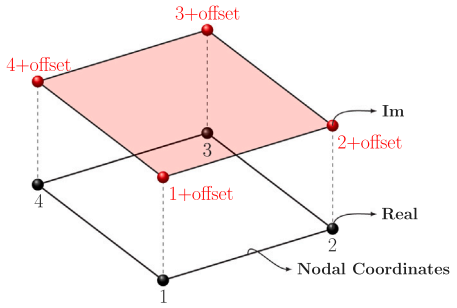

Fig. 4. 8-noded complex element (4 real and 4 imaginary).

than the number of real nodes. For ease of understanding, a particular example will be shown in the following section where the creation of the input file will be discussed.

The implementation of this change into the UEL can be observed in Listing 3 where "zCoords" and "zU" denote the complex version of the coordinates and nodal displacements, respectively.

Listing 3: Conversion of the nodes and displacements from CR format to complex variable type

```
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Complex Version of the Coordinates and displacements

zCoords = CMPLX(coords(1:2,1:nn_real), coords(1:2,(nn_real+1):(2*nn_real)))
zU = CMPLX(u(1:ndof_real*nn_real), u(ndof_real*nn_real+1:2*ndof_real*nn_real))
```

After this change, all mathematical operations within the UEL are carried without further modifications from the standard PPR element but now using the complex-variable intrinsic type. If a comparison operator such as greater than (>) or less than or equal (≥) is used, the real part of the variable has to be used as in the real-valued UEL. Listing 4 shows an example where the Macauley bracket of the energy constants is computed using comparison operators.

Listing 4: Comparison operator example

```
! Macauley bracket of the Energy constants
IF (REAL(G_n) > REAL(G_t)) THEN
    dGnt = G_n - G_t
    dGtn = ZERO
ELSEIF (REAL(G_n) < REAL(G_t)) THEN
    dGnt = ZERO
    dGtn = G_t - G_n
ELSE
    dGnt = ZERO
    dGtn = ZERO
END IF
```

As a final step, the complex-valued stiffness matrix and force vector have to be converted back into CR format and assigned to the Abaqus intrinsic variables (amatrx and RHS), respectively, before exiting the UEL for assembly and solution as aforementioned. The assignments of the CR form of the cohesive stiffness matrix and force vector to the Abaqus intrinsic variables within the PPR UEL are shown in Listing 5. The same process is done within the linear elastic UEL as can be seen in the source code listing Appendix section.

The Cauchy–Riemann form of the stiffness matrix is non-symmetric due to the negative sign in the upper right quadrant of the stiffness matrix as shown in Eq. (9). However, although not discussed here, it can be shown that the magnitude of this term is not significant in the solution of the system of equations. As a result, one can use an unsymmetric solver keeping the negative sign in place, or make that term positive and use a symmetric solver. Empirical evidence shows that the symmetric solver is approximately 13% faster. Listing 5 uses a symmetric version of the CR stiffness matrix.

Listing 5: Conversion from complex type to CR format for the stiffness matrix and force vector

```
! Volume Differential
dvol = HALF * el_length * GP_w(i) * th

! Cohesive Traction
Fcoh = MATMUL(TRANSPOSE(Bc), T)*dvol

! Cohesive Stiffness Mat
Kcoh = MATMUL(TRANSPOSE(Bc),MATMUL(T_d,Bc))*dvol

! Real and complex parts of the rhs and stiffness matrix

Freal(:,1) = REAL(Fcoh(:,1))
Fimag(:,1) = AIMAG(Fcoh(:,1))


Kreal = REAL(Kcoh)
Kimag = AIMAG(Kcoh)

! Add contributions for this integration point

!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
!  --- Convert Fcoh and Kcoh to Cauchy-Riemann form ---   !
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
! NOTE: CR form is unsymmetric but upper right term is not
! significant. Therefore it is set as +Kim to have a
! symmetric matrix.

! K = |Kre    +Kim|
!     |Kim     Kre|

! CR form of right hand side vector (Abaqus UEL intrinsic variable)

rhs(1:mcrd*nn_real,1)                 = rhs(1:mcrd*nn_real,1)                 - Freal(:,1)
rhs(mcrd*nn_real+1:2*mcrd*nn_real,1) = rhs(mcrd*nn_real+1:2*mcrd*nn_real,1) - Fimag(:,1)

! CR form of stiffness matrix (Abaqus UEL intrinsic variable)

! Upper left - REAL part of Kcoh
amatrx(1:mcrd*nn_real,1:mcrd*nn_real) = amatrx(1:mcrd*nn_real, 1:mcrd*nn_real) + Kreal

! Lower right - REAL part of Kcoh(same as upper left)
amatrx(mcrd*nn_real+1:2*mcrd*nn_real,mcrd*nn_real+1:2*mcrd*nn_real) = amatrx(mcrd*nn_real+1:2*mcrd*nn_real, mcrd*nn_real
     +1:2*mcrd*nn_real) + Kreal

! Lower left - Imaginary part of Kcoh
amatrx(mcrd*nn_real+1:2*mcrd*nn_real, 1:mcrd*nn_real) = amatrx(mcrd*nn_real+1:2*mcrd*nn_real, 1:mcrd*nn_real) + Kimag

! Upper right - Imaginary part of Kcoh (same magnitude as lower left but sign difference. To use a symmetric solver use the
     same sign)
amatrx(1:mcrd*nn_real,mcrd*nn_real+1:2*mcrd*nn_real) = amatrx(1:mcrd*nn_real, mcrd*nn_real+1:2*mcrd*nn_real) + Kimag
```

Algorithm 1 summarizes how the complex-valued Abaqus UEL works in order to compute the CR form of the stiffness matrix and right hand side vector for both, cohesive and linear elastic elements. If the element is cohesive, then the cohesive material variable of interest is perturbed along the imaginary axis and the complex-valued elemental stiffness matrix and RHS vector are computed using Fortran intrinsic complex type. As Abaqus does not have a complex-valued solver, a CR representation is used to return a real-valued matrix and vector for subsequent assembly and solution of the system of equations. If the element is linear elastic, only the real-valued stiffness matrix and RHS vector are required as the variable of interest belongs to the ZPPR subroutine. A CR form of the linear elastic elemental stiffness matrix and RHS vector with zero imaginary components are returned to Abaqus for assembly and solution. Finally, during the second pass through the UEL where the residual is checked, post-processing quantities such as strain and stresses, and their derivatives with respect to the perturbed variable, are computed.

2.3.3. Input file generation

In this section, the generation of the input file for a particular example will be shown. In Section 3, a plate under mode-I traction will be used as a verification example, see Fig. 5. This example has been used by other authors to verify their method [26]. The plate contains 2 elements, a linear elastic element (shaded in gray) and cohesive element (outlined in red). Fig. 5 shows the finite element mesh for a regular (real-valued) finite element analysis. In order to run a ZFEM analysis, the degrees of freedom are augmented, resulting in a mesh with both, real and imaginary nodes. Fig. 6 shows the complex mesh with the real nodes (black) and imaginary

// **ZFEM UEL Procedure**
Data: Nodes, Elements, Properties, Loads, Boundary Conditions (BCs), Perturbation Information.
Result: Displacement vector $\mathbf{u}$, Derivatives of displacement vector with respect to the variable of interest $\partial\mathbf{u}/\partial\theta$
for $e \in \Omega$ **do** // For all the elements in the mesh
  if $e \in \Omega_c$ **then** // If the element is cohesive
    $\theta^* = \theta + i\,h$; // Apply perturbation to the parameter of interest (defined in the input file)
    $\mathbf{u}^* = cmx(\mathbf{u}_{\text{Re}}, \mathbf{u}_{\text{Im}})$; // Create complex form of displacement vector
    $\mathbf{xy}^* = cmx(\mathbf{xy}_{\text{Re}}, \mathbf{xy}_{\text{Im}})$; // Create complex form of nodal coordinates
    $\mathbf{K}^*_{e_c}, \mathbf{f}^*_{e_c} \leftarrow$ ZPPR_UEL(e_c); // Compute complex-valued elemental stiffness matrix and RHS vector
    $\mathbf{K} = \begin{bmatrix} \mathbf{K}_{\text{Re}} & -\mathbf{K}_{\text{Im}} \\ \mathbf{K}_{\text{Im}} & \mathbf{K}_{\text{Re}} \end{bmatrix}, \mathbf{f} = \begin{bmatrix} \mathbf{f}_{\text{Re}} \\ \mathbf{f}_{\text{Im}} \end{bmatrix}$; // Convert $\mathbf{K}^*_{e_c}$ and $\mathbf{f}^*_{e_c}$ to CR form
    Return to Abaqus for assembly and solution;
  else if $e \in \Omega_e$ **then** // If the element is linear elastic
    $\mathbf{K}_{e_e}, \mathbf{f}_{e_e} \leftarrow$ ELASTIC_UEL(e_c); // Compute real-valued elemental stiffness matrix and RHS vector
    $\mathbf{K} = \begin{bmatrix} \mathbf{K}_{\text{Re}} & 0 \\ 0 & \mathbf{K}_{\text{Re}} \end{bmatrix}, \mathbf{f} = \begin{bmatrix} \mathbf{f}_{\text{Re}} \\ 0 \end{bmatrix}$; // Convert CR form with zero imaginary components
    Return to Abaqus for assembly and solution;
  end
  // Compute complex-valued post-processing quantities such as strains and stresses and its derivatives with respect to the perturbed variable
  $\epsilon^*(\mathbf{u}^*) = \epsilon + (\partial\epsilon/\partial\theta) \times h$; // Strains and derivatives with respect to θ
  $\sigma^*(\mathbf{u}^*) = \sigma + (\partial\sigma/\partial\theta) \times h$; // Stresses and derivatives with respect to θ
end

Algorithm 1: Pseudocode for ZFEM UEL

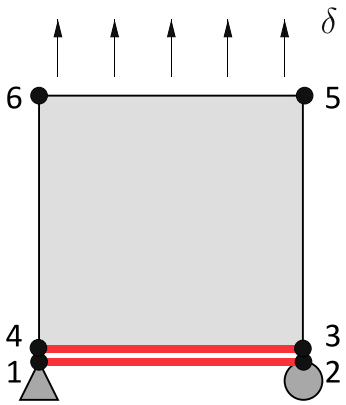

Fig. 5. Real-valued finite element mesh corresponding to the mode-I verification example. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
Source: Adapted from [26].

nodes (red). Now, instead of having 4 nodes per element, both elements (linear elastic and cohesive) consist of 8 nodes (4 real and 4 imaginary). Since Abaqus has no knowledge of imaginary coordinates, the input file contains real coordinates that are then "stitched" together as a complex-variable node within the UEL. As such, a pairing of element numbers is needed to indicate which real and imaginary numbers comprise a complex node. In this example the nodal values are "offset" by an arbitrary large integer number, 1000, e.g., the linear elastic element is formed by the nodes (4, 3, 5, 6, 1004, 1003, 1005, 1006). As aforementioned in Fig. 4, the offset value used to define the imaginary node is a number greater than the number of real nodes, - for this particular case 1000 was chosen as the offset value for ease of understanding. For instance, node 4 (real) and node 1004 (imaginary) are stitched together as a complex node.

Listing 6 shows the section of the Abaqus input file for the mode-I example where the nodes and nodal sets are defined. Nodes 1–6 define the real-variable nodes and nodes 1001–1006 define the imaginary nodes. The imaginary nodal coordinate values define the "perturbation" of the geometry, e.g., a shape sensitivity. Since only material property values are being perturbed here, the

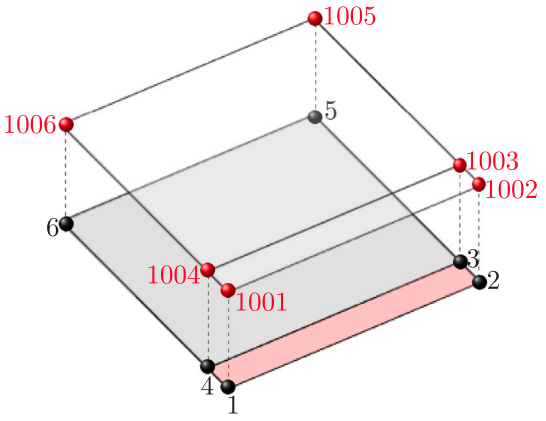

Fig. 6. Complex-valued finite element mesh corresponding to the mode-I verification example. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

imaginary $x-$ and $y-$coordinates are zero. The entire input file is defined by combining Listings 6–8. A double star "**" means a comment in the input file.

Listing 6: Nodes and node sets definition for ZFEM's input file

```
*HEADING
Patch test, Mode I
*NODE
** Real nodes
1, 0.0, -0.1
2, 100, -0.1
3, 100, 0.0
4, 0.0, 0.0
5, 100, 100
6, 0.0, 100
** Imaginary nodes (offset=1000)
1001, 0.0, 0.0
1002, 0.0, 0.0
1003, 0.0, 0.0
1004, 0.0, 0.0
1005, 0.0, 0.0
1006, 0.0, 0.0
** Nodal sets
*NSET, NSET=NODES_RE, generate
1, 6
*NSET, NSET=NODES_IM, generate
1001, 1006
```

Subsequently, the element type and the elements and element sets are defined as shown in Listing 7. The first line defines the element type, the number of nodes per element, the number of active coordinates, number of real property values associated with the element, and the number of solution-dependent state variables associated with the element using the keywords: TYPE, NODE, COORDINATES, PROPERTIES, and VARIABLES, respectively. TYPE=U1 means a user defined element. For the complex element NODE=8, and this variable is accessed within the UEL through the variable "nnode" as was shown in Section 2.3.2. Similarly, the number of active degrees of freedom per node is given by COORDINATES=2 and is accessed within the UEL through the variable "mcrd". PROPERTIES=12 is defined by the maximum number of properties passed to an element set. In particular, 12 properties will be passed to the cohesive elements (ELSET=COHESIVE_ELEMS) and 4 to the linear elastic elements (ELSET=ELASTIC_ELEMS).

Table 4
Solution-dependent state variables, SVARS ("*" denotes a complex-variable).

SVARS	Description
1	Real component of the normal displacement, Re $[\Delta_n^*]$, from integration point 1
2	Real component of the tangential displacement, Re $[\Delta_t^*]$, from integration point 1
3	Real component of the normal displacement, Re $[\Delta_n^*]$, from integration point 2
4	Real component of the tangential displacement, Re $[\Delta_t^*]$, from integration point 2
5	Imaginary component of the normal displacement, Im $[\Delta_n^*]$, from integration point 1
6	Imaginary component of the tangential displacement, Im $[\Delta_t^*]$, from integration point 1
7	Imaginary component of the normal displacement, Im $[\Delta_n^*]$, from integration point 2
8	Imaginary component of the tangential displacement, Im $[\Delta_t^*]$, from integration point 2

Last, there is a total of 8 solution-dependent state variables which will be updated for every cohesive element at the end of every increment. These variables, accessed within the UEL through the variable SVARS, are shown in Table 4.

Listing 7: Elements definition for ZFEM's input file

```
**
** User Defined Elements
**
*USER ELEMENT, TYPE=U1, NODE=8, COORDINATES=2, PROPERTIES=12, VARIABLES=8
1, 2
*ELEMENT, TYPE=U1, ELSET=ALL_ELEMS
1, 4, 3, 5, 6, 1004, 1003, 1005, 1006
2, 1, 2, 3, 4, 1001, 1002, 1003, 1004
** Element sets
*ELSET, ELSET=ELASTIC_ELEMS
1
*ELSET, ELSET=COHESIVE_ELEMS
2
```

The line following "*USER ELEMENT" in Listing 7, shows the active degrees of freedom in the analysis. "1" corresponds to displacements in the x-direction and "2" to displacements in the y-direction. The keyword "*ELEMENT" is used to define the elements the two complex-elements (one linear elastic and one cohesive), as shown in Fig. 6. The first column is the element ID and the following columns define the nodes that form the element, 4 real and 4 imaginary. Then, linear elastic and cohesive element sets are defined. Element 1 is linear elastic and 2 is cohesive.

Lastly, the loading amplitude, additional nodal sets to define boundary conditions or output requests, and the loading step are defined (see Listing 8). Then, the UEL properties for each element set are defined starting with the "*UEL PROPERTY" keyword. A total of 4 properties are passed to the linear elastic elements which belong to the element set "ELASTIC_ELEMS", see Table 3. For the cohesive elements, corresponding to the element set "COHESIVE_ELEMS", there are a total of 12 properties, see Table 2. The first property denotes the element type, where 1 means linear elastic and 2 cohesive. Then, properties 2–10 correspond to the normal and tangential fracture energies (G_n, G_t), normal and tangential cohesive strength (T_n, T_t), shape parameters (α, β), slope indicators (λ_n, λ_t) and the thickness of the element (t). These properties define the mechanical response of a cohesive element using the PPR model. The additional variables define ZFEM's perturbation step size, h, and an indicator that defines the variable to be perturbed. In Abaqus, a total of 8 properties have to be defined per line. If there are less than 8 properties, Abaqus will default the missing properties to zero. For example, properties 5–8 for the linear elastic elements in the example input file have a value of zero.

The analysis type and the boundary conditions are subsequently defined. A non-linear static analysis is defined with an initial step size of 0.05, total time of 3.0, minimum step size of 3.0×10^{-5}, and a maximum step size of 0.05. Then the boundary conditions are defined. Note that if a boundary condition is defined on the real nodes, the displacement along the imaginary direction is set to zero, that is, if the displacement is enforced to be a constant, the derivative of that displacement with respect to any parameter must be enforced to be zero. Finally, output to the ".dat" file is requested. In this particular example, the nodal displacements and reaction forces denoted with U and RF, respectively in Abaqus were requested for the real (NODES_RE) and imaginary (NODES_IM) nodal sets.

Listing 8: Loading amplitude, nodal sets, element properties and step definitions

```
** ~~~~~~~~~~~~~~~~~~~~
** Loading Amplitude
** ~~~~~~~~~~~~~~~~~~~~
*Amplitude, name=Amp-1
0., 0., 1., 0.03, 2., -0.01, 3., 0.1
** ~~~~~~~~~~~~~~~~~~~~
** Other nodal sets
** fof BC definition
** ~~~~~~~~~~~~~~~~~~~~
*NSET, NSET=UP_RE
5, 6
*NSET, NSET=UP_IM
1005, 1006
*NSET, NSET=ROLLER_RE
2
*NSET, NSET=ROLLER_IM
1002
*NSET, NSET=PIN_RE
1
*NSET, NSET=PIN_IM
1001
** Everything should be in [mm]
** Elastic element properties
*UEL PROPERTY, ELSET=ELASTIC_ELEMS
** elem_type, Emod    nu    thick
     1,       32.0e3, 0.20,  10
** Cohesive element properties
*UEL PROPERTY, ELSET=COHESIVE_ELEMS
** elem_type  Gn     Gt     T_n  T_t  alph  beta   ln
     2,      0.100, 0.200,  4,   3,    5,  1.6, 0.005
**  lt    thick   h     pert_flag
   0.005, 10,    1e-10,    1
******************************************************
** ~~~~~~~~~~~~~~~~~~~~
** Step definition
** ~~~~~~~~~~~~~~~~~~~~
*STEP, NLGEOM, INC=4000, UNSYMM=NO
*STATIC
0.05, 3.0, 3e-05, 0.05
** ~~~~~~~~~~~~~~~~~~~~
** Boundary Conditions
** ~~~~~~~~~~~~~~~~~~~~
** Fixed sets
*BOUNDARY
ROLLER_RE, 2, 2
ROLLER_IM, 2, 2
PIN_RE,    1, 2
PIN_IM,    1, 2
** Moving sets
*Boundary, amplitude=Amp-1
UP_RE, 2, 2, 1.0
UP_IM, 2, 2, 0.0
** ~~~~~~~~~~~~~~~~~~~~
** Prints to dat file
** ~~~~~~~~~~~~~~~~~~~~
** Real nodes
*NODE PRINT, NSET=NODES_RE
U, RF
** Imaginary nodes
*NODE PRINT, NSET=NODES_IM
U, RF
*END STEP
```

3. Numerical examples

In this section, ZFEM's capabilities of computing accurate sensitivities of displacements with respect to the parameters of the PPR cohesive model are demonstrated. Once the derivatives are verified, its use to inversely determine the cohesive material parameters is shown for an adhesively bonded double cantilever beam.

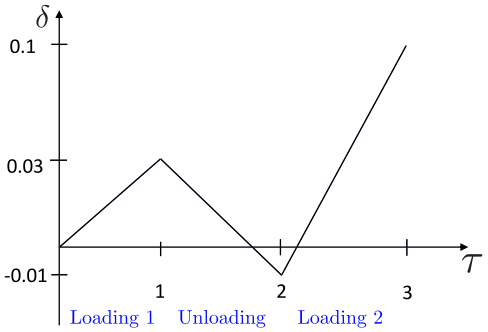

Fig. 7. Loading history.

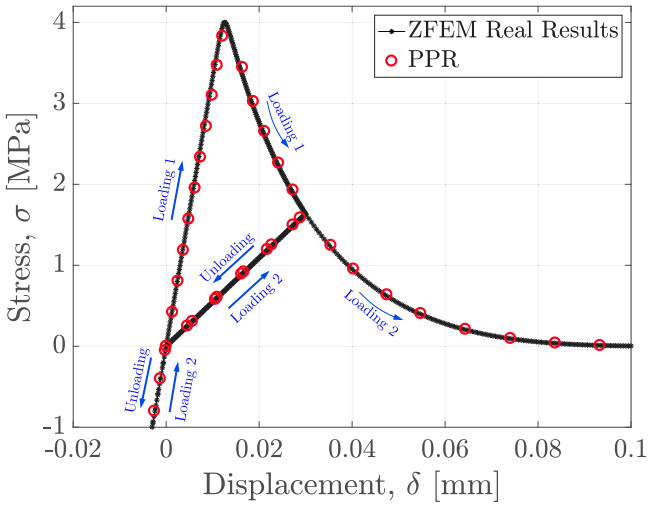

Fig. 8. Comparison of Park and Paulino [26] and ZFEM's real-valued results.

3.1. Mode-I patch test

A plate subjected to mode-I traction was used as a verification example for both real displacements and their derivatives with respect to the chosen traction-separation variable. The geometry consisted of a square plate (100 mm × 100 mm) with a cohesive element at the bottom with an aspect ratio of 1000:1, see Fig. 5. The geometry was first elongated at the top up to 0.003 mm, then compressed to 0.001 mm, and finally elongated again until failure as shown in Fig. 7. Linear elastic material behavior was considered for the plate with the following parameters: elastic modulus $E = 32\,\text{GPa}$, Poisson's ratio $\nu = 0.2$ and the PPR parameters were $\phi_n = 0.1\,\text{N/mm}$, $\phi_t = 0.2\,\text{N/mm}$, $\sigma_{max} = 4\,\text{MPa}$, $\tau_{max} = 3\,\text{MPa}$, $\alpha = 5$, $\beta = 1.6$, $\lambda_n = 0.005$ and $\lambda_t = 0.005$. These are the same numerical values used in [26] in their verification example.

The plate was modeled using two 8-noded (4 real and 4 imaginary) quadrilateral plane strain elements. One element followed a traditional linear-elastic formulation (gray element in Fig. 6) and the other followed the PPR cohesive formulation (red element in Fig. 5). In order to compute the derivatives, the perturbation step size, h, was set as 10^{-10} times the value of the parameter being perturbed. A total of 4 nonlinear ZFEM analyses were needed to compute 4 derivatives, one for each mode-I cohesive property: normal fracture energy (ϕ_n), maximum normal traction (σ_{max}), normal shape parameter (α) and the normal slope indicator (λ_n).

Fig. 8 shows a comparison of the y-displacement vs. stress of the upper edge (either node 5 or 6 in Fig. 5) obtained from [26] and the real part of ZFEM's results. As expected, the results obtained from the real valued analysis and the real part of ZFEM's results are overlapping.

The derivatives of the y-displacement of node 3 ($\mathbf{v}_3$) with respect to the mode-I cohesive parameters (normal fracture energy, normal cohesive strength, normal shape parameter and normal slope indicator) were computed, $\partial\mathbf{v}_3/\partial\phi_n$, $\partial\mathbf{v}_3/\partial\sigma_{max}$, $\partial\mathbf{v}_3/\partial\alpha$, and $\partial\mathbf{v}_3/\partial\lambda_n$, respectively. The derivatives were verified against those obtained with the finite difference (FD) method with step sizes varying from 0.001 and 0.05 times the perturbed variable. Refer to Fig. 7 for the different loading/unloading zones and Table 5 for the step size used in finite differences for each variable. In order to find the optimal step size for the FD scheme, a convergence analysis was carried out where several step sizes were tested until convergence was achieved. In FD, the step size is problem dependent and the convergence analysis must be performed for each derivative obtained through FD. ZFEM is independent of

Table 5
Step sizes used with finite differences.

Variable	Finite difference step size
ϕ_n	$0.01 \times \phi_n$
σ_{max}	$0.001 \times \sigma_{max}$
α	$0.05 \times \alpha$
λ_n	$0.01 \times \lambda_n$

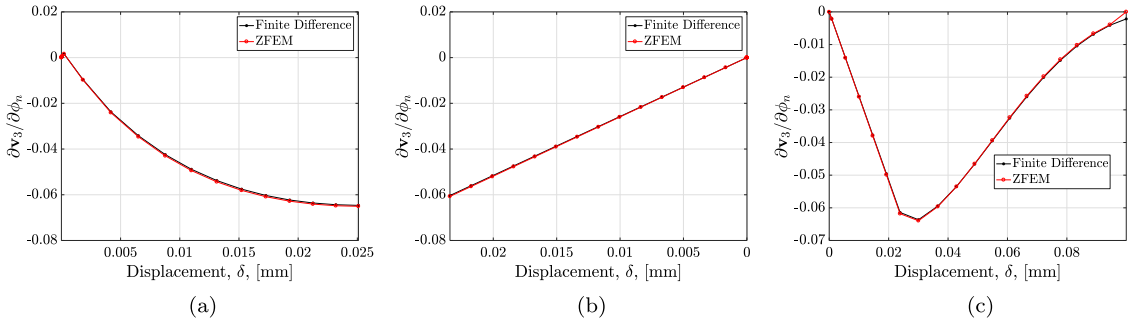

Fig. 9. Derivatives with respect to ϕ_n for (a) loading zone 1, (b) unloading zone, and (c) loading zone 2.

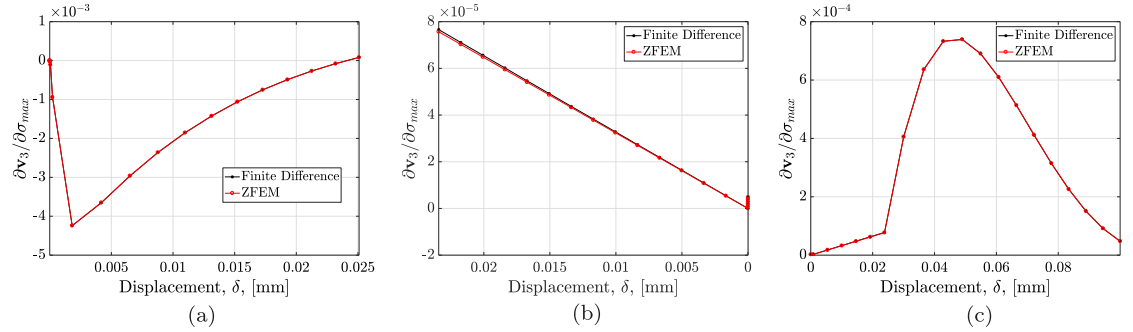

Fig. 10. Derivatives with respect to σ_{max} for (a) loading zone 1, (b) unloading zone, and (c) loading zone 2.

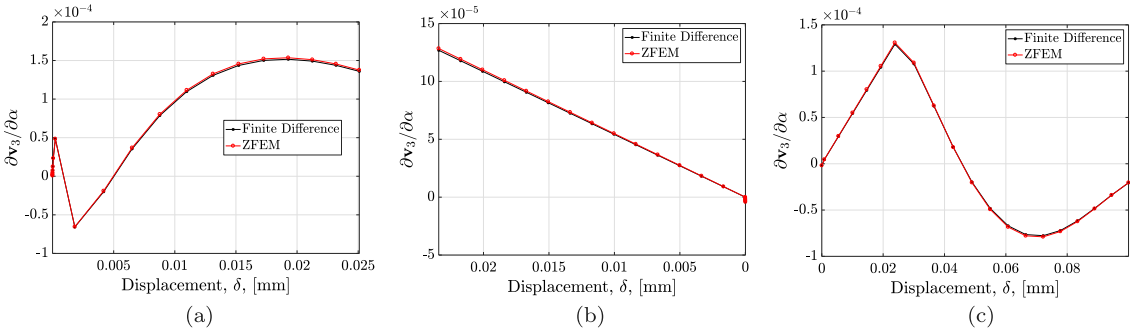

Fig. 11. Derivatives with respect to α for (a) loading zone 1, (b) unloading zone, and (c) loading zone 2.

this issue and does not require a convergence analysis - a single analysis provides accurate derivative results. It can be observed that the results presented in Figs. 9–12 are in excellent agreement.

For the derivatives with respect to the normal fracture energy (Fig. 9), it can be observed that during loading zone 1, an increase of ϕ_n results in lower y-displacement of node 3 (derivative with negative sign), which is caused by greater stiffness of the cohesive element, see Fig. 9a. During unloading, Fig. 9b, an increase of ϕ_n decreases $\mathbf{v}_3$ with lower magnitude as the displacement decreases. Finally, during loading zone 2 (Fig. 9c), an increase of ϕ_n will result in a lower value for $\mathbf{v}_3$ and the effect becomes greater until the last point of loading zone 1 is reached again. Then, the effect is still negative but it diminishes its magnitude as loading zone 2 continues.

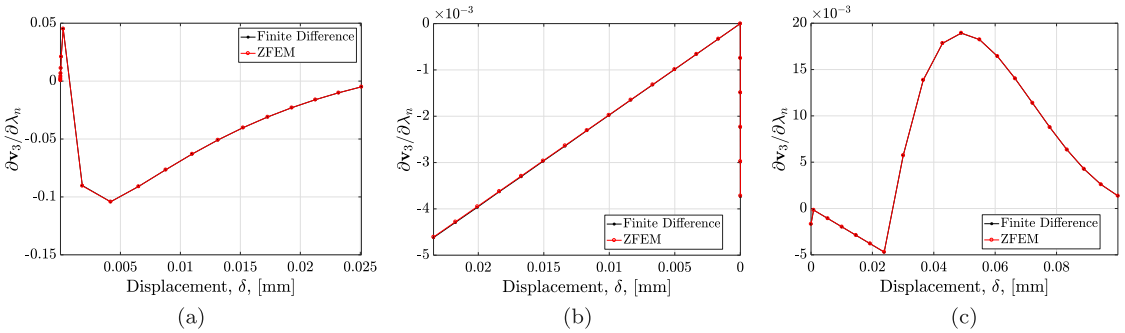

Fig. 12. Derivatives with respect to λ_n for (a) loading zone 1, (b) unloading zone, and (c) loading zone 2.

Fig. 10 shows the derivatives of $\mathbf{v}_3$ with respect to σ_{max}. During loading zone 1, an increase of σ_{max} will result in lower y-displacement of node 3. The effect reduces after the peak load is reached and softening starts. Then, during unloading, an increase of σ_{max} increases the magnitude of $\mathbf{v}_3$ and its effect decreases as the last point of the unloading zone is reached. During loading zone 2 the derivative is positive and its magnitude decreases towards the end.

The derivatives of $\mathbf{v}_3$ with respect to the normal shape parameter, α, are shown in Fig. 11. During loading zone 1, an increment of α results in a larger y-displacement of node 3 with increasing magnitude towards the end of this loading zone. For the unloading zone, see Fig. 11b, the effect is still positive but its magnitude decreases as the end of the unloading zone is reached. During the last loading zone (Fig. 11c), initially, an increase of α results in a greater $\mathbf{v}_3$. Then, the curve shifts from positive to negative magnitude and from concave to convex where an increase of α will now reduce the magnitude of $\mathbf{v}_3$. The change from concave to convex curve can be also observed in Fig. 2 where the dependency of the traction-separation curve with respect to different values of α is shown.

Finally, the derivatives of $\mathbf{v}_3$ with respect to λ_n are shown in Fig. 12. During loading 1, see Fig. 12a, an increase of λ_n will decrease the magnitude of $\mathbf{v}_3$. Then, during unloading (Fig. 12b), the effect is still negative with a decreasing magnitude towards the end of this loading zone. For the last loading zone, see Fig. 12c, the effect is now positive and the magnitude starts decreasing towards the end of loading zone 2.

3.2. Inverse determination of cohesive material properties for an adhesively bonded double cantilever beam

As shown in Section 2.1, the mode-I traction-separation behavior of the PPR constitutive model is defined by 4 parameters: normal fracture energy (ϕ_n), maximum normal traction (σ_{max}), normal shape parameter (α) and the normal slope indicator (λ_n). In order to inversely determine the cohesive material parameters a residual function, w, which compares the experimental and computational load–displacement curve, is defined as

$$w(\boldsymbol{\theta}) = \left\| \boldsymbol{R}_P \right\|^2 \tag{12}$$

where $\boldsymbol{\theta}$ is a vector consisting on the material parameters to be optimized as

$$\boldsymbol{\theta} = \begin{bmatrix} \phi_n \\ \sigma_{max} \\ \alpha \\ \lambda_n \end{bmatrix} \tag{13}$$

and $\boldsymbol{R}_P$ is the residual vector of the reaction force

$$\boldsymbol{R}_P = \sum_{\mathbf{t}} \frac{\mathbf{P}^{\mathrm{EXP}} - \mathbf{P}^{\mathrm{COMP}}(\boldsymbol{\theta})}{\|\mathbf{P}^{\mathrm{EXP}}\|}$$

where the superscripts "EXP" and "COMP" denote the reaction force obtained experimentally and computationally, respectively. The load residual, $\boldsymbol{R}_P$, is computed for every loading increment of the finite element simulation. Note, however, that this approach can also be used to determine elastic material properties such as E and ν. These properties can be added to θ.

Then, the optimization procedure is defined as

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} w(\boldsymbol{\theta}) \tag{14}$$

where a gradient-based optimization algorithm will be used for its solution. For a gradient based optimization algorithm, it is necessary to compute the derivatives of the finite element output variables with respect to the unknown parameters. In this paper, ZFEM, will be used to compute the derivatives of the finite element solution vector with respect to the cohesive fracture parameters that govern the interfacial behavior of an adhesively bonded joint. This allows one to compute the derivative of the residual function with respect to the cohesive material parameters as

$$\frac{\partial w(\boldsymbol{\theta})}{\partial \theta_i} = 2\left(\frac{\boldsymbol{P}^{\mathrm{EXP}} - \boldsymbol{P}^{\mathrm{COMP}}(\boldsymbol{\theta})}{\|\boldsymbol{P}^{\mathrm{EXP}}\|} \right) \frac{\partial \boldsymbol{R}_P}{\partial \theta_i} \tag{15}$$

Table 6
Dimensions for the DCB test specimen.

Dimension	Symbol	Value [mm]
Length	L	101.6
Initial crack size	a_0	38.57
Distance to applied load	L_P	12.96
Arm height	H	1.016
Specimen width	B	25.4

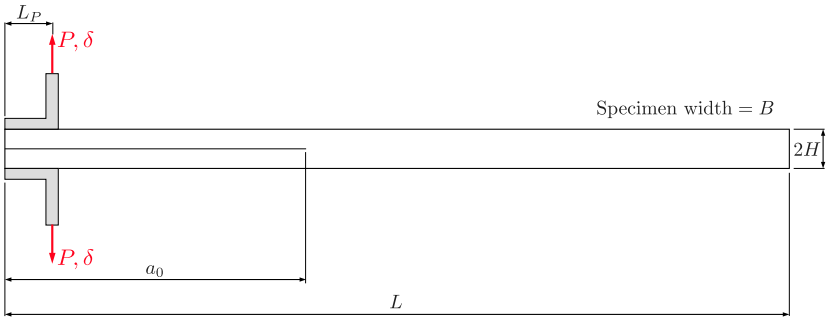

Fig. 13. Schematic of the DCB sample.

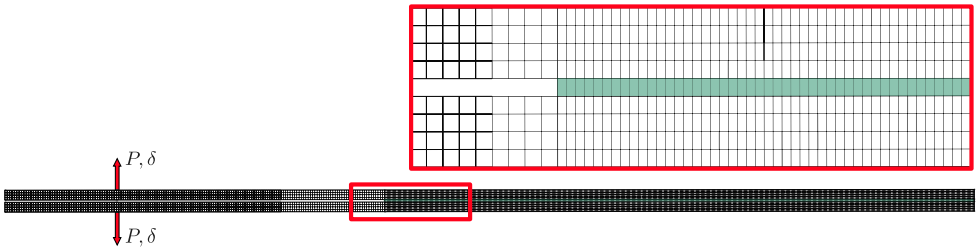

Fig. 14. Finite element mesh for the DCB specimen. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

where $\theta_i (i = 1, \ldots, N_\theta)$ is the ith element from the solution vector $\boldsymbol{\theta}$ and N_θ is the number of material properties to be optimized. The derivative of the load residual, $\boldsymbol{R}_P$, is given by

$$\frac{\partial \boldsymbol{R}_P}{\partial \theta_i} = \sum_t \frac{\frac{-\partial P^{\text{COMP}}}{\partial \theta_i}}{\|\mathbf{P}^{\text{EXP}}\|} \tag{16}$$

For each material property in $\boldsymbol{\theta}$, a ZFEM analysis is required to compute the derivative of the residual function (Eq. (15)). Hence, a total of N_θ ZFEM analyses are required to provide the gradient of the residual function to the optimizer.

A double cantilever beam (DCB) test was used to demonstrate the use of ZFEM's gradients towards the inverse determination of the interfacial properties. A "synthetic" load–displacement data set from a finite element simulation was used as the reference curve. The mode-I cohesive parameters optimized were the normal fracture energy (ϕ_n), maximum normal traction (σ_{max}). The geometry of the beam is shown in Fig. 13 with the dimensions summarized in Table 6.

The DCB was modeled using a total of 5639 8-noded (4 real and 4 imaginary) ZFEM elements. For the steel adherends (plates), a total of 5159 8-noded (4 real and 4 imaginary) quadrilateral plane stress linear elastic elements were used. A total of 480 8-noded (4 real and 4 imaginary) ZPPR cohesive elements (green elements in Fig. 14) were used to recreate the bond at the interface. Linear elastic material behavior was considered for the steel with elastic modulus $E = 200\,\text{GPa}$ and Poisson's ratio $\nu = 0.3$. The finite element mesh can be observed in Fig. 14. The PPR parameters used to generate the "synthetic" load–displacement curve were $\phi_n = 0.835\,\text{N/mm}$, $\phi_t = 0.835\,\text{N/mm}$, $\sigma_{max} = 13\,\text{MPa}$, $\tau_{max} = 13\,\text{MPa}$, $\alpha = 2$, $\beta = 2$, $\lambda_n = 0.005$ and $\lambda_t = 0.005$.

In order to minimize the objective function and find the optimized parameters, the scipy.optimize.minimize [56] library was used; in particular a conjugate gradient method by Polak and Ribiere, which is a variant of the Fletcher-Reeves method described in [57] and the sequential least squares programming (SLSQP), a non-gradient based method, were employed to minimize the residual. For the finite difference-based optimization, the gradient was approximated using 2-point finite differencing. Hence, two additional function evaluations (per variable being optimized) were needed in order to compute the gradient of the residual function using FD. With ZFEM, no additional function calls are needed in order to compute the gradient as the derivatives are obtained as a byproduct

Table 7
Optimized values comparison.

Method	ϕ_n [N/mm]	ϕ_n relative error [%]	σ_{max} [MPa]	σ_{max} relative error [%]
ZFEM	0.83063	0.527	13.435	3.3070
FD	0.83009	0.588	10.001	23.076
SLSQP	0.83153	0.415	10.087	22.406

Table 8
Optimization times comparison.

Method	Function calls	Iterations	Total time [s]
ZFEM	38	4	2361
FD	72	1	3794
SLSQP	48	7	2529

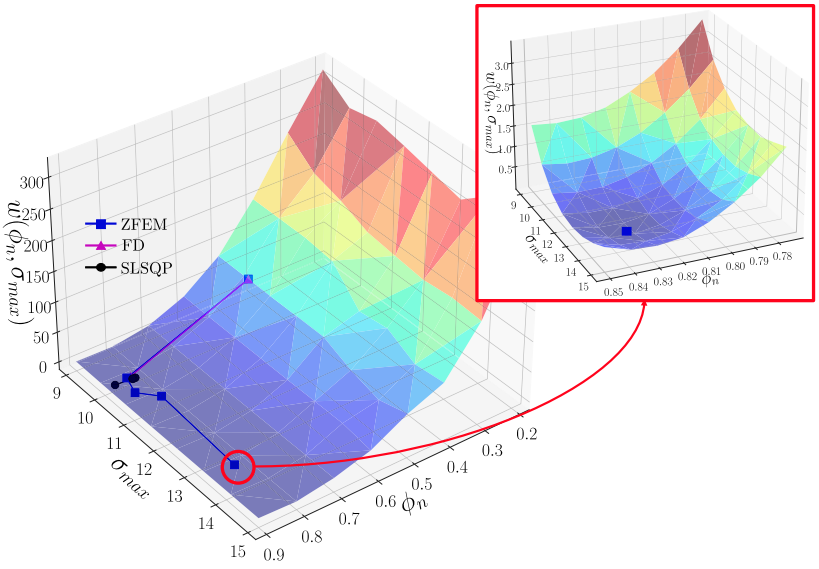

Fig. 15. Optimization results for the DCB.

of the complex-valued analysis. For this particular example involving two parameters, a total of 2 ZFEM analyses are needed in order to compute the residual function (Eq. (12)) and its derivative with respect to the variables being optimized (Eq. (15)). An initial educated guess of the parameters to be optimized was provided to the three methods (ZFEM, FD and SLSQP)

$$\boldsymbol{\theta}_0 = \begin{bmatrix} \phi_{n_0} \\ \sigma_{max_0} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 10.0 \end{bmatrix} \tag{17}$$

Table 7 shows a comparison of the optimized cohesive material parameters obtained using ZFEM, FD and SLSQP. The convergence criterion for the gradient-based optimization was that the norm of the gradient vector must be less than 1.0. As it can be observed, ZFEM's results are in excellent agreement with those used to generate the synthetic data set ($\phi_n = 0.835$ N/mm and $\sigma_{max} = 13$ MPa). However, the FD result for σ_{max} has a large relative error due to truncation errors (a common issue associated with finite differencing). Similarly, for the SLSQP method, there is good agreement for ϕ_n but σ_{max} has a large error. In addition, as shown in Table 8, ZFEM requires a total of 38 complex-valued analyses for a total of 4 iterations of the conjugate gradient method. In contrast, FD required 72 real-valued analyses for only one iteration of the optimizer. When using SLSQP, 48 real-valued analyses were required for a total of 7 iterations. Hence, when highly accurate and truncation error free derivatives are provided to the optimization subroutine, the optimization algorithm converges faster and to a more accurate answer. In summary, using ZFEM with accurate derivatives resulted in a 40% reduction in computational time compared to finite differencing, and 7% reduction in time compared to SLSQP, see Table 8, yet ZFEM obtained significantly superior numerical results.

Fig. 15 shows a three-dimensional representation of the residual function with 4 iterations for ZFEM, 1 for FD and the last three iterations for SLSQP (for illustrative purposes). As it can be observed, both FD and SLSQP converge in the same region. A close-up of the region where ZFEM converges is also shown.

Fig. 16 compares the load–displacement curves for the converged values of ZFEM, FD and SLSQP against the "synthetic" data curve. The curves corresponding to the initial guess of cohesive parameters and the analytical solution obtained using beam theory

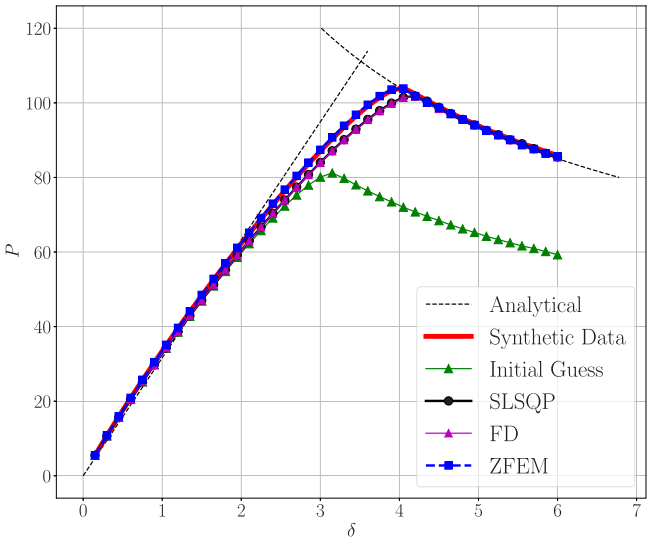

Fig. 16. Load–displacement curves comparison.

and linear elastic fracture mechanics [1] are also plotted for reference. As it can be observed, ZFEM and the "synthetic" curves are in excellent agreement while the FD and SLSQP curves have a discrepancy caused by the error in the converged value for σ_{max}.

4. Discussion

The process described here to complexify the PPR element is a generic method that can be used to compute numerically exact derivatives from any finite element implementation. In summary, the process is: (a) duplicate the nodes in order to represent imaginary nodes required by the complex element, (b) change the variable definition from real to complex for those variables that are affected by the perturbed variable, (c) modify the input file with information of which variable is going to be perturbed and request output for the imaginary displacements and (d) if the software that is being used lacks a complex-valued solver as is the Abaqus case, expand both the stiffness matrix and right hand side vector to a Cauchy–Riemann form.

The use of complex-variables provides the capability to compute first order derivatives of displacements and post-processing quantities, e.g., strains, stresses, strain energy, with respect to any parameter of the cohesive element. This approach can be extended to higher order and mixed derivatives through the use of hypercomplex algebra [53,58]. However, the use of hypercomplex algebra requires the implementation of operator overloading and hypercomplex algebra support [30], and the computational runtimes are greatly increased as a result.

In this paper only derivatives with respect to the cohesive material parameters were computed. However, the methodology is general and derivatives with respect to any shape e.g., thickness and interface length, material property or loads are possible.

During the optimization procedure, it was shown that FD and SLSQP converge to erroneous σ_{max} values. When a tighter tolerance was defined, ZFEM converged to the same values obtained with a larger tolerance. However, the FD and SLSQP methods did not show improvements to the optimized values despite almost tripling the number of function calls.

5. Conclusions

The complex Taylor series expansion approach embodied within a finite element formulation, ZFEM, was shown to be an effective tool to compute highly accurate derivatives of the nodal displacements with respect to the PPR cohesive material parameters. The results were compared against those obtained using the finite difference with excellent agreement.

The additional computational cost for an analysis with the ZPPR element is approximately 50% over that of a real-variable PPR element. The results are step size independent and highly accurate with the use of a small step size h, of 10^{-10} times the parameter of interest. Contrary to finite differences, ZFEM does not require a search for an accurate step size and a single analysis provides accurate derivative results.

ZFEM's derivatives can be incorporated within an optimization scheme to inversely determine material parameters as demonstrated in this work. The accuracy of ZFEM derivatives was shown to expedite the convergence rate of a gradient-based optimization procedure and minimize the error. The performance of the gradient-based optimization scheme fed with ZFEM derivatives was shown to be superior than a gradient-free method, both in terms of runtime and relative error. Thus, ZFEM derivatives can be used to improve the efficiency of gradient-based algorithms used for optimizing the design of mechanical systems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was funded by Pacific Northwest National Laboratories, USA under Contract No. 484440

Appendix

A complete working UEL for the complex-variable PPR element and the input file corresponding to the numerical example presented in Section 3.1 are presented below.

Listing 9: Complex-valued UEL containing the ZPPR and the linear elastic element

```
! @mainpage Complex-valued UEL Documentation
!!
!! \b Subroutines:
!!
!! \b UEL: Master Subroutine
!!
!! \b coh_UEL: 8 noded (4 real, 4 imag) PPR Cohesive Element
!!
!! \b elastic_UEL \b: 8 noded (4 real, 4 imag) Linear Elastic Element
!!
!! ------------------------------------------------------------------------
!!
!! @page  tutorials   How to run the ZUEL
!!
!! @section   Linux    Using the UEL in Linux
!!
!! 1. Load Abaqus and the Intel FORTRAN Compiler
!!
!!        $ module load abaqus/6.12 intel/13/64bit
!!
!! 2. Change directory to the run directory (where you want output files).
!!
!!        $ cd rundir/
!!
!! 3. Run Abaqus
!!
!!        $ abaqus job=job_name inp=input_file_name.inp user=UEL_name.f
!!
!! The .log and .sta files will let you know if the job has been completed.
!! .dat file has all the output requested through the input file.
!! Compilation errors appear in the log file.
!!
!**************************************************************************************************
! SUBROUTINE UEL
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!> @author Daniel Ramirez-Tamayo, Arturo Montoya, Harry Millwater, The University of
!! Texas at San Antonio
!! based on the paper "Computational implementation of the PPR potential-based cohesive model in ABAQUS:
!! Educational perspective" by K. Park, G.H. Paulino, and J.R. Roesler.
!!
!> @brief This is the master subroutine that contains the cohesive and the linear elastic
!! element subroutines. Here it is decided which subroutine is accessed.
!!
!> @details At the end of the subroutine you need to provide the elemental stiffness
!! matrix (amatrx) and the right hand side vector (RHS). For the cohesive elements
!! the "coh_UEL" is used. For the rest, a 8 noded (4 real and 4 imag) linear
!! elastic element is used.
!!
!! @param[in,out] rhs(mlvarx,1)           Contributions from the element to the right-hand-side vectors
!! @param[in,out] amatrx(ndofel, ndofel)  Stiffness matrix of the element
!! @param[in,out] svars(*)                solution-dependent state variables
!! @param[in,out] energy(*)               Energy quantities associated with the element
!! @param[in]     ndofel                  Number of degrees of freedom in the element
!! @param[in]     nrhs                    Number of load vectors (1)
!! @param[in]     nsvars                  User-defined number of solution-dependent state variables
!! @param[in]     props(*)                Properties assigned to the UEL
!! @param[in]     nprops                  User-defined number of real property values
!! @param[in]     coords(mcrd, nnode)     Spacial coordinates of the element
!! @param[in]     mcrd                    Number of coordinates per node (2)
!! @param[in]     nnode                   User-defined number of nodes on the element
```

```
!! @param[in]     u(ndofel)                Solution vector
!! @param[in]     du(mlvarx,*)             Incremental values of the solution vector
!! @param[in]     v(ndofel)                Time rate of change of the variables
!! @param[in]     a(ndofel)                Acceleration of the variables
!! @param[in]     jtype                    Integer defining the element type
!! @param[in]     time(2)                  Time step and total time
!! @param[in]     dtime                    Time increment
!! @param[in]     kstep                    Current step number
!! @param[in]     kinc                     Current increment number
!! @param[in]     jelem                    User-assigned element number
!! @param[in]     params(*)                Parameters associated with the solution procedure
!! @param[in]     ndload                   Identification number of the distributed load or flux
!! @param[in]     jdltyp(mdload,*)         To define distributed loads
!! @param[in]     adlmag(mdload,*)         Load magnitude of the K1th distributed load
!! @param[in]     predef(2, npredf, nnode) Values of the predefined field variables
!! @param[in]     npredf                   Number of predefined field variables,
!! @param[in]     lflags(*)                To define the current solution procedure
!! @param[in]     mlvarx                   Dimensioning parameter
!! @param[in]     ddlmag(mdload,*)         Increments in the magnitudes of the distributed loads
!! @param[in]     mdload                   Total number of distributed loads and/or fluxes
!! @param[in]     pnewdt                   Ratio of suggested new time increment to the current time
    increment
!! @param[in]     jprops(*)                Integer property values assigned to the UEL
!! @param[in]     njprop                   User-defined number of integer property values
!! @param[in]     period                   Time period of the current step
!**********************************************************************************************
SUBROUTINE UEL(rhs, amatrx, svars, energy, ndofel, nrhs, nsvars, props, &
nprops, coords, mcrd, nnode, u, du, v, a, jtype, time, dtime, kstep,    &
kinc, jelem, params, ndload, jdltyp, adlmag, predef, npredf, lflags,    &
mlvarx, ddlmag, mdload, pnewdt, jprops, njprop, period)

    ! Include Abaqus Parameters as inputs
    INCLUDE 'ABA_PARAM.INC'

    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
    ! Select double precision Real
    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!

    ! Select precision that provides 15 digits for a real data type and an exponent range of 10^+-307"
    INTEGER, PARAMETER :: DP = SELECTED_REAL_KIND(15, 307)

    ! Variables used for counters
    INTEGER :: i                          ! Counter for the current cohesive element
    INTEGER :: elem_type                  ! Flag to determine if the element is cohesive (=1) or linear
        elastic (=2)

    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ! Abaqus Variables

    ! Scalar parameters

    REAL(DP) :: dtime                     ! Time increment
    REAL(DP) :: pnewdt                    ! Ratio of suggested new time increment to the current time
        increment
    REAL(DP) :: period                    ! Time period of the current step
    INTEGER  :: mlvarx                    ! Dimensioning parameter
    INTEGER  :: ndofel                    ! Number of degrees of freedom in the element
    INTEGER  :: nrhs                      ! Number of load vectors (1)
    INTEGER  :: nsvars                    ! User-defined number of solution-dependent state variables
    INTEGER  :: nprops                    ! User-defined number of real property values
    INTEGER  :: njprop                    ! User-defined number of integer property values
    INTEGER  :: mcrd                      ! Number of coordinates per node (2)
    INTEGER  :: nnode                     ! User-defined number of nodes on the element
    INTEGER  :: jtype                     ! Integer defining the element type
    INTEGER  :: kstep                     ! Current step number
    INTEGER  :: kinc                      ! Current increment number
    INTEGER  :: jelem                     ! User-assigned element number
    INTEGER  :: ndload                    ! Identification number of the distributed load or flux
    INTEGER  :: mdload                    ! Total number of distributed loads and/or fluxes
    INTEGER  :: npredf                    ! Number of predefined field variables,

    ! Arrays

    REAL(DP) :: props(*)                  ! Properties assigned to the UEL
    INTEGER  :: jprops(*)                 ! Integer property values assigned to the UEL
    REAL(DP) :: coords(mcrd, nnode)       ! Original coordinates of the element
    REAL(DP) :: u(ndofel)                 ! Solution vector
    REAL(DP) :: du(mlvarx,*)              ! Incremental values of the solution vector
    REAL(DP) :: v(ndofel)                 ! Time rate of change of the variables
```

```
    REAL(DP) :: a(ndofel)                ! Acceleration of the variables
    INTEGER  :: jdltyp(mdload,*)         ! To define distributed loads
    REAL(DP) :: adlmag(mdload,*)         ! Load magnitude of the K1th distributed load
    REAL(DP) :: ddlmag(mdload,*)         ! Increments in the magnitudes of the distributed loads
    REAL(DP) :: predef(2, npredf, nnode) ! Values of the predefined field variables
    REAL(DP) :: params(*)                ! Parameters associated with the solution procedure
    INTEGER  :: lflags(*)                ! To define the current solution procedure
    REAL(DP) :: time(2)                  ! Time step and total time

    ! Variables to be defined

    REAL(DP) :: rhs(mlvarx,1)            ! Contributions from the element to the right-hand-side vectors
    REAL(DP) :: amatrx(ndofel, ndofel)   ! Stiffness matrix of the element
    REAL(DP) :: svars(*)                 ! solution-dependent state variables
    REAL(DP) :: energy(*)                ! Energy quantities associated with the element
    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    ! Flag that determines if the element is cohesive or linear elastic
    elem_type = PROPS(1)

    SELECT CASE (elem_type)
        CASE (1) ! If the element is cohesive
            CALL elastic_uel(rhs, amatrx, svars, energy, ndofel, nrhs, nsvars, props,          &
                         nprops, coords, mcrd, nnode, u, du, v, a, jtype, time, dtime, kstep, &
                         kinc, jelem, params, ndload, jdltyp, adlmag, predef, npredf, lflags, &
                         mlvarx, ddlmag, mdload, pnewdt, jprops, njprop, period)
        CASE (2) ! If the element is linear elastic
            CALL coh_UEL(rhs, amatrx, svars, energy, ndofel, nrhs, nsvars, props,              &
                         nprops, coords, mcrd, nnode, u, du, v, a, jtype, time, dtime, kstep, &
                         kinc, jelem, params, ndload, jdltyp, adlmag, predef, npredf, lflags, &
                         mlvarx, ddlmag, mdload, pnewdt, jprops, njprop, period)
    END SELECT

END SUBROUTINE uel
!****************************************************************************************************
! SUBROUTINE coh_UEL
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!> @author Daniel Ramirez-Tamayo, Arturo Montoya, Harry Millwater, The University of
!! Texas at San Antonio (Complexification of the PPR UEL)
!! based on the paper "Computational implementation of the PPR potential-based cohesive model in ABAQUS:
!! Educational perspective" by K. Park, G.H. Paulino, and J.R. Roesler.
!!
!> @brief Complex version of the PPR subroutine proposed by Park et al.
!!
!> @details This subroutine returns the Cauchy-Riemann version of both, stiffness matrix and RHS vector.
!! The variable to be perturbed is defined through the input file.
!!
!! @param[in,out] rhs(mlvarx,1)          Contributions from the element to the right-hand-side vectors
!! @param[in,out] amatrx(ndofel, ndofel)  Stiffness matrix of the element
!! @param[in,out] svars(*)               solution-dependent state variables
!! @param[in,out] energy(*)              Energy quantities associated with the element
!! @param[in]     ndofel                 Number of degrees of freedom in the element
!! @param[in]     nrhs                   Number of load vectors (1)
!! @param[in]     nsvars                 User-defined number of solution-dependent state variables
!! @param[in]     props(*)               Properties assigned to the UEL
!! @param[in]     nprops                 User-defined number of real property values
!! @param[in]     coords(mcrd, nnode)    Original coordinates of the element
!! @param[in]     mcrd                   Number of coordinates per node (2)
!! @param[in]     nnode                  User-defined number of nodes on the element
!! @param[in]     u(ndofel)              Solution vector
!! @param[in]     du(mlvarx,*)           Incremental values of the solution vector
!! @param[in]     v(ndofel)              Time rate of change of the variables
!! @param[in]     a(ndofel)              Acceleration of the variables
!! @param[in]     jtype                  Integer defining the element type
!! @param[in]     time(2)                Time step and total time
!! @param[in]     dtime                  Time increment
!! @param[in]     kstep                  Current step number
!! @param[in]     kinc                   Current increment number
!! @param[in]     jelem                  User-assigned element number
!! @param[in]     params(*)              Parameters associated with the solution procedure
!! @param[in]     ndload                 Identification number of the distributed load or flux
!! @param[in]     jdltyp(mdload,*)       To define distributed loads
!! @param[in]     adlmag(mdload,*)       Load magnitude of the K1th distributed load
!! @param[in]     predef(2, npredf, nnode) Values of the predefined field variables
!! @param[in]     npredf                 Number of predefined field variables,
!! @param[in]     lflags(*)              To define the current solution procedure
!! @param[in]     mlvarx                 Dimensioning parameter
!! @param[in]     ddlmag(mdload,*)       Increments in the magnitudes of the distributed loads
!! @param[in]     mdload                 Total number of distributed loads and/or fluxes
```

```
!! @param[in]     pnewdt                   Ratio of suggested new time increment to the current time
     increment
!! @param[in]     jprops(*)                Integer property values assigned to the UEL
!! @param[in]     njprop                   User-defined number of integer property values
!! @param[in]     period                   Time period of the current step
!*******************************************************************************************************
SUBROUTINE coh_uel(rhs, amatrx, svars, energy, ndofel, nrhs, nsvars, props, &
    nprops, coords, mcrd, nnode, u, du, v, a, jtype, time, dtime, kstep,    &
    kinc, jelem, params, ndload, jdltyp, adlmag, predef, npredf, lflags,    &
    mlvarx, ddlmag, mdload, pnewdt, jprops, njprop, period)


    IMPLICIT NONE

    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
    ! Properties passed to the UEL from input file
    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!

    ! PROPS(1):  Element type. Cohesive (=1), linear elastic (=2)
    ! PROPS(2):  Normal fracture energy (G_n)
    ! PROPS(3):  Tangential fracture energy (G_t)
    ! PROPS(4):  Normal cohesive strength (Tn_m)
    ! PROPS(5):  Tangential cohesive strength (Tt_m)
    ! PROPS(6):  Normal shape parameter (alph)
    ! PROPS(7):  Tangential shape parameter (beta)
    ! PROPS(8):  Normal initial slope indicator (l_n)
    ! PROPS(9):  Tangential initial slope indicator (l_t)
    ! PROPS(10): Thickness of the cohesive element
    ! PROPS(11): Perturbation step size (usually 1e-10, then it is multiplied by the variable of interest)
    ! PROPS(12): Perturbation flag - indicates which parameter to perturb

    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
    ! Solution-dependent state variables (SVARS)
    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!

    ! SVARS(1): Real component of the normal displacement from integration point # 1
    ! SVARS(2): Real component of the tangential displacement from integration point # 1
    ! SVARS(3): Real component of the normal displacement from integration point # 2
    ! SVARS(4): Real component of the tangential displacement from integration point # 2
    ! SVARS(5): Imaginary component of the normal displacement from integration point # 1
    ! SVARS(6): Imaginary component of the tangential displacement from integration point # 1
    ! SVARS(7): Imaginary component of the normal displacement from integration point # 2
    ! SVARS(8): Imaginary component of the tangential displacement from integration point # 2

    !~~~~~~~~~~~~~~~~~~~~~~~~~~~!
    ! Select double precision Real
    !~~~~~~~~~~~~~~~~~~~~~~~~~~~!

    INTEGER, PARAMETER :: DP = SELECTED_REAL_KIND(15, 307)

    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ! Abaqus Variables

    ! Scalar parameters

    REAL(DP) :: dtime                      ! Time increment
    REAL(DP) :: pnewdt                     ! Ratio of suggested new time increment to the current time
        increment
    REAL(DP) :: period                     ! Time period of the current step
    INTEGER  :: mlvarx                     ! Dimensioning parameter
    INTEGER  :: ndofel                     ! Number of degrees of freedom in the element
    INTEGER  :: nrhs                       ! Number of load vectors (1)
    INTEGER  :: nsvars                     ! User-defined number of solution-dependent state variables
    INTEGER  :: nprops                     ! User-defined number of real property values
    INTEGER  :: njprop                     ! User-defined number of integer property values
    INTEGER  :: mcrd                       ! Number of coordinates per node (2)
    INTEGER  :: nnode                      ! User-defined number of nodes on the element
    INTEGER  :: jtype                      ! Integer defining the element type
    INTEGER  :: kstep                      ! Current step number
    INTEGER  :: kinc                       ! Current increment number
    INTEGER  :: jelem                      ! User-assigned element number
    INTEGER  :: ndload                     ! Identification number of the distributed load or flux
    INTEGER  :: mdload                     ! Total number of distributed loads and/or fluxes
    INTEGER  :: npredf                     ! Number of predefined field variables,

    ! Arrays

    REAL(DP) :: props(*)                   ! Properties assigned to the UEL
    INTEGER  :: jprops(*)                  ! Integer property values assigned to the UEL
```

```
REAL(DP) :: coords(mcrd, nnode)      ! Original coordinates of the element
REAL(DP) :: u(ndofel)                ! Solution vector
REAL(DP) :: du(mlvarx,*)             ! Incremental values of the solution vector
REAL(DP) :: v(ndofel)                ! Time rate of change of the variables
REAL(DP) :: a(ndofel)                ! Acceleration of the variables
INTEGER  :: jdltyp(mdload,*)         ! To define distributed loads
REAL(DP) :: adlmag(mdload,*)         ! Load magnitude of the K1th distributed load
REAL(DP) :: ddlmag(mdload,*)         ! Increments in the magnitudes of the distributed loads
REAL(DP) :: predef(2, npredf, nnode) ! Values of the predefined field variables
REAL(DP) :: params(*)                ! Parameters associated with the solution procedure
INTEGER  :: lflags(*)                ! To define the current solution procedure
REAL(DP) :: time(2)                  ! Time step and total time

! Variables to be defined

REAL(DP) :: rhs(mlvarx,1)            ! Contributions from the element to the right-hand-side vectors
REAL(DP) :: amatrx(ndofel, ndofel)   ! Stiffness matrix of the element
REAL(DP) :: svars(*)                 ! solution-dependent state variables
REAL(DP) :: energy(*)                ! Energy quantities associated with the element
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! Parameters Useful for Calculations

PARAMETER  ZERO=0.0_DP, ONE=1.0_DP, NONE=-1.0_DP, HALF=0.5_DP, TWO=2.0_DP ! Zero, one, negative one,
     half, two
PARAMETER  gaussCoord=SQRT(3.0_DP)/3.0_DP, gaussWeight=1.0_DP, ninpt=2    ! For Gaussian integration

!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Variables used within the cohesive UEL

! Complex Variables

COMPLEX(DP), DIMENSION(mcrd*nnode/2,mcrd*nnode/2)  :: Sc       ! Element stiffness matrix of a cohesive
      element
COMPLEX(DP), DIMENSION(mcrd*nnode/2,nrhs)          :: Fc       ! Cohesive internal force vector
COMPLEX(DP), DIMENSION(mcrd,nrhs)                  :: T        ! Cohesive traction vector
COMPLEX(DP), DIMENSION(mcrd,mcrd)                  :: T_d      ! Derivative of the cohesive traction (
     Tangent matrix)
COMPLEX(DP), DIMENSION(mcrd, mcrd)                 :: Arot     ! Coordinate transformation matrix
COMPLEX(DP), DIMENSION(mcrd*nnode/2, mcrd*nnode/2) :: R        ! Rotation Matrix
COMPLEX(DP), DIMENSION(mcrd*nnode/2)               :: U_l      ! Nodal displacement in the local
     coordinate system
COMPLEX(DP), DIMENSION(mcrd,mcrd*nnode/2)          :: Bc       ! Global displacement-separation
     relation matrix
COMPLEX(DP), DIMENSION(mcrd)                       :: del      ! Normal and tangential separations
COMPLEX(DP), DIMENSION(nnode/2)                    :: del_l    ! local nodal displacement jumps
COMPLEX(DP), DIMENSION(mcrd*mcrd, mcrd*nnode/2)    :: L        ! local displacement separation relation
      matrix
COMPLEX(DP), DIMENSION(mcrd*nnode/2, 1)            :: Fcoh     ! Cohesive Traction
COMPLEX(DP), DIMENSION(mcrd*nnode/2, mcrd*nnode/2) :: Kcoh     ! Stiffness Matrix
COMPLEX(DP), DIMENSION(mcrd, nnode/2)              :: zCoords  ! Complex Coordinates
COMPLEX(DP), DIMENSION(nnode/2*ndofel/2)           :: zU       ! Complex Displacement

COMPLEX(DP) :: G_n,   G_t          ! Normal and Tangential Fracture Energies
COMPLEX(DP) :: Tn_m, Tt_m          ! Normal and Tangential Cohesive Strength
COMPLEX(DP) :: alph, beta          ! Shape Parameters
COMPLEX(DP) :: l_n,   l_t          ! Slope Indicators
COMPLEX(DP) :: dn, dt              ! Final crack opening widths
COMPLEX(DP) :: m, n                ! Exponents to compute Potential Function
COMPLEX(DP) :: Gam_n, Gam_t        ! Energy constants, related to the fracture energies
COMPLEX(DP) :: dGnt, dGtn          ! Macauley bracket <G_n-G_t> or <G_t-G_n>, respectively
COMPLEX(DP) :: del1, del3          ! Nodal separations in x (Local Coords)
COMPLEX(DP) :: del2, del4          ! Nodal separations in y (Local Coords)
COMPLEX(DP) :: deln_max, delt_max  ! Maximum displacements jumps during load history
COMPLEX(DP) :: el_length           ! Length of the cohesive element
COMPLEX(DP) :: dvol                ! Volume Differential

! Real and complex parts of the cohesive stiffness matrix
REAL(DP), DIMENSION(mcrd*nnode/2, mcrd*nnode/2) :: Kreal, Kimag

! Real and complex parts of the RHS vector
REAL(DP), DIMENSION(mcrd*nnode/2,1) :: Freal, Fimag

REAL(DP), DIMENSION(2,nnode/2) :: shapeN   ! Shape function matrix
REAL(DP), DIMENSION(2)         :: GP       ! Gauss points
REAL(DP), DIMENSION(2)         :: GP_w     ! Weight at the Gauss points
REAL(DP)                       :: N1, N2   ! Shape Functions

REAL(DP) :: h                 ! Perturbation step size
```

```
REAL(DP) :: th                  ! Thickness
INTEGER  :: perturbation_flag ! Perturbation flag
INTEGER  :: I                   ! Counter
INTEGER  :: nn_real             ! Number of real nodes
INTEGER  :: ndof_real           ! Number of real DOF

!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! Rename Abaqus variables that define the number of real nodes and DOF
nn_real   = nnode/TWO
ndof_real = mcrd

!~~~~~~~~~~~~~~~~~~
! Read UEL Properties

perturbation_flag = PROPS(12) ! Defines variable to be perturbed
h                 = PROPS(11) ! ZFEM's perturbation step size

! Define fracture parameters
G_n  = PROPS(2)
G_t  = PROPS(3)
Tn_m = PROPS(4)
Tt_m = PROPS(5)
alph = PROPS(6)
beta = PROPS(7)
l_n  = PROPS(8)
l_t  = PROPS(9)
th   = PROPS(10)

! Apply perturbation to the parameter of interest
SELECT CASE (perturbation_flag)
    CASE (1)
        G_n  = CMPLX(PROPS(2), h*PROPS(2))
    CASE (2)
        G_t  = CMPLX(PROPS(3), h*PROPS(3))
    CASE (3)
        Tn_m = CMPLX(PROPS(4), h*PROPS(4))
    CASE (4)
        Tt_m = CMPLX(PROPS(5), h*PROPS(5))
    CASE (5)
        alph = CMPLX(PROPS(6), h*PROPS(6))
    CASE (6)
        beta = CMPLX(PROPS(7), h*PROPS(7))
    CASE (7)
        l_n  = CMPLX(PROPS(8), h*PROPS(8))
    CASE (8)
        l_t  = CMPLX(PROPS(9), h*PROPS(9))
END SELECT

! ~~~~~~~~~~~~~~~~~~
! Initialize Variables
rhs     = ZERO
amatrx  = ZERO

GP(1)   =  gaussCoord
GP(2)   = -gaussCoord

GP_w(1) = gaussWeight
GP_w(2) = gaussWeight

!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Complex Version of the Coordinates and displacements

zCoords = CMPLX(coords(1:2,1:nn_real), coords(1:2,(nn_real+1):(2*nn_real)))
zU = CMPLX(u(1:ndof_real*nn_real), u(ndof_real*nn_real+1:2*ndof_real*nn_real))

!~~~~~~~~~~~~~~~~~~~~~~~~~~
! Determine Cohesive Parameters

! Eqn 20 (from PPR Paper)
m = (alph-ONE)*alph*l_n**TWO/(ONE-alph*l_n**TWO)
n = (beta-ONE)*beta*l_t**TWO/(ONE-beta*l_t**TWO)

! Eqn 21
dn = alph*G_n/(m*Tn_m)*(ONE-l_n)**(alph-ONE) &
     * (alph/m*l_n+ONE)**(m-ONE)*(alph+m)*l_n

! Eqn 22
dt = beta*G_t/(n*Tt_m)*(ONE-l_t)**(beta-ONE) &
```

```
        * (beta/n*l_t+ONE)**(n-ONE)*(beta+n)*l_t

    ! Macauley bracket of the Energy constants
    IF (REAL(G_n) > REAL(G_t)) THEN
        dGnt = G_n - G_t
        dGtn = ZERO
    ELSEIF (REAL(G_n) < REAL(G_t)) THEN
        dGnt = ZERO
        dGtn = G_t - G_n
    ELSE
        dGnt = ZERO
        dGtn = ZERO
    END IF

    ! Determine Energy Constants (Eq 18 and 19)
    IF (REAL(G_n) == REAL(G_t)) THEN
        Gam_n = -G_n*(alph/m)**m
        Gam_t = (beta/n)**n
    ELSE
        Gam_n = (-G_n)**(dGnt/(G_n-G_t))*(alph/m)**m
        Gam_t = (-G_t)**(dGtn/(G_t-G_n))*(beta/n)**n
    ENDIF

    ! Compute transformation matrix (A, Eq 27)
    Arot      = ZERO
    el_length = ZERO
    CALL Coords_Transform(Arot, el_length, zCoords, zU, nn_real, mcrd)

    R = ZERO
    ! Assembly of Rotation Matrix (Eq 26)
    R(1:2, 1:2) = Arot
    R(3:4, 3:4) = Arot
    R(5:6, 5:6) = Arot
    R(7:8, 7:8) = Arot

    ! Find displacements in local coords
    U_l = MATMUL(R, zU)

    ! Local displacement separation Matrix (Eq 29)
    ! Set the diagonal to -1
    L = ZERO
    DO i = 1, nn_real
        L(i,i) = NONE
    END DO

    L(1,7) = ONE
    L(2,8) = ONE
    L(3,5) = ONE
    L(4,6) = ONE

    ! Find nodal displacement jumps (Eq 28)
    del_l = MATMUL(L, U_l)

    del1 = U_l(7) - U_l(1)
    del2 = U_l(8) - U_l(2)
    del3 = U_l(5) - U_l(3)
    del4 = U_l(6) - U_l(4)

    ! For all the Gaussian integration points
    ! integrate over the length of the cohesive element
    DO i = 1, ninpt

        ! Eq 31
        N1 = HALF*(ONE - GP(i))
        N2 = HALF*(ONE + GP(i))

        ! E1 30
        shapeN = ZERO
        shapeN(1,1) =  N1
        shapeN(2,2) =  N1

        shapeN(1,3) =  N2
        shapeN(2,4) =  N2

        ! Normal and Tangential separations
        del = MATMUL(shapeN, del_l)

        ! Update maximum separation from previous increment
        delt_max = CMPLX(SVARS(ninpt*(i-1)+1), SVARS(ninpt*(i-1)+1+4))
```

```
    deln_max = CMPLX(SVARS(ninpt*(i-1)+2), SVARS(ninpt*(i-1)+2+4))

    ! Cohesive traction-separation relation of the PPR model
    call Cohesive_PPR(T, T_d, Gam_n, Gam_t, alph,     beta, m, n,      &
                       dn,  dt,  dGtn,  dGnt,  del, deln_max, delt_max)

    ! Global displacement separation relation matrix (Eq 32)
    Bc = MATMUL(shapeN, MATMUL(L,R))

    ! Volume Differential
    dvol = HALF * el_length * GP_w(i) * th

    ! Cohesive Traction
    Fcoh = MATMUL(TRANSPOSE(Bc), T)*dvol


    ! Cohesive Stiffness Mat
    Kcoh = MATMUL(TRANSPOSE(Bc),MATMUL(T_d,Bc))*dvol

    ! Real and complex parts of the rhs and stiffness matrix

    Freal(:,1) = REAL(Fcoh(:,1))
    Fimag(:,1) = AIMAG(Fcoh(:,1))


    Kreal = REAL(Kcoh)
    Kimag = AIMAG(Kcoh)


    ! Add contributions for this integration point

    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
    !  --- Convert Fcoh and Kcoh to Cauchy-Riemann form ---  !
    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
    ! NOTE: CR form is unsymmetric but upper right term is not
    ! significant. Therefore it is set as +Kim to have a
    ! symmetric matrix.

    ! K = |Kre   +Kim|
    !     |Kim    Kre|

    ! CR form of right hand side vector (Abaqus UEL intrinsic variable)

    rhs(1:mcrd*nn_real,1)                 = rhs(1:mcrd*nn_real,1)                 - Freal(:,1)
    rhs(mcrd*nn_real+1:2*mcrd*nn_real,1) = rhs(mcrd*nn_real+1:2*mcrd*nn_real,1) - Fimag(:,1)

    ! CR form of stiffness matrix (Abaqus UEL intrinsic variable)

    ! Upper left - REAL part of Kcoh
    amatrx(1:mcrd*nn_real,1:mcrd*nn_real) = amatrx(1:mcrd*nn_real, 1:mcrd*nn_real) + Kreal

    ! Lower right - REAL part of Kcoh(same as upper left)
    amatrx(mcrd*nn_real+1:2*mcrd*nn_real,mcrd*nn_real+1:2*mcrd*nn_real) = amatrx(mcrd*nn_real+1:2*mcrd*
         nn_real, mcrd*nn_real+1:2*mcrd*nn_real) + Kreal

    ! Lower left - Imaginary
    amatrx(mcrd*nn_real+1:2*mcrd*nn_real, 1:mcrd*nn_real) = amatrx(mcrd*nn_real+1:2*mcrd*nn_real, 1:
         mcrd*nn_real) + Kimag

    ! Upper right - Imaginary (same magnitude as lower left but sign difference. To use a symmetric
         solver use the same sign)
    amatrx(1:mcrd*nn_real,mcrd*nn_real+1:2*mcrd*nn_real) = amatrx(1:mcrd*nn_real, mcrd*nn_real+1:2*mcrd
         *nn_real) + Kimag

    ! Update State Vars
    IF((REAL(delt_max) < ABS(REAL(del(1)))).AND.(ABS(REAL(del(1))) > REAL(l_t*dt))) THEN
        SVARS(ninpt*(i-1)+1)    = abs( REAL(del(1)))
        SVARS(ninpt*(i-1)+1+4) = (AIMAG(del(1)))
    END IF
    IF ((REAL(deln_max) < REAL(del(2))) .AND. (REAL(del(2)) > REAL(l_n*dn))) THEN
        SVARS(ninpt*(i-1)+2)    =  REAL(del(2))
        SVARS(ninpt*(i-1)+2+4) = AIMAG(del(2))
    END IF

END DO ! Integration points

!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Aditional Subroutines for the Cohesive Element
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
CONTAINS

!##########################################################################################################
! SUBROUTINE Cohesive_PPR
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!> @brief Computation of the traction vector and the tangent matrix
!!
!!
!! @param[in]          Gam_n     Normal Fracture Energy
!! @param[in]          Gam_t     Tangential Fracture Energy
!! @param[in]          alph      Mode  I shape parameter
!! @param[in]          beta      Mode II shape parameter
!! @param[in]          m         Exponent to compute the potential function
!! @param[in]          n         Exponent to compute the potential function
!! @param[in]          dn        Final normal crack opening width
!! @param[in]          dt        Final tangential crack opening width
!! @param[in]          dGtn      Macauley bracket <G_n-G_t>
!! @param[in]          dGnt      Macauley bracket <G_t-G_n>
!! @param[in]          deln_max  Maximum normal displacement jump during load history
!! @param[in]          delt_max  Maximum tangential displacement jump during load history
!! @param[in]          del       Jumps at the gauss point
!! @param[in, out]     T         Traction vector
!! @param[in, out]     Td        Derivative of the coh. traction (tangent matrix)
!##########################################################################################################
SUBROUTINE Cohesive_PPR(T, T_d, Gam_n, Gam_t, alph, beta, m, n,        &
                        dn,  dt,  dGtn,  dGnt,  del, deln_max, delt_max)
    IMPLICIT NONE

    INTEGER, PARAMETER :: DP = SELECTED_REAL_KIND(15, 307)

    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ! INPUTS

    COMPLEX(DP), INTENT(IN) :: Gam_n, Gam_t, alph, beta, m, n, &
                               dn,  dt,  dGtn,  dGnt, deln_max, delt_max

    COMPLEX(DP), DIMENSION(2), INTENT(IN) :: del ! Jumps at the gauss point

    ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ! Outputs
    COMPLEX(DP), DIMENSION(2,1), INTENT(INOUT) :: T   ! Traction vector
    COMPLEX(DP), DIMENSION(2,2), INTENT(INOUT) :: T_d ! Derivative of the coh. traction (tangent matrix
        )
    ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ! Internal Variables
    COMPLEX(DP) :: Tn, Tt      ! Normal and Tangential tractions
    COMPLEX(DP) :: deln, delt  ! Normal and Tangential separations
    COMPLEX(DP) :: sign_dt     ! Sign of the incremental traction
    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    delt = abs(del(1))
    deln = del(2)

    ! Find the sign of the tangential displacement
    IF (REAL(del(1)) >= ZERO) THEN
        sign_dt = ONE
    ELSE
        sign_dt = NONE
    END IF
    Tn = ZERO

    ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ! Cohesive Normal Traction, Eq 12

    IF (REAL(deln) < ZERO) THEN
        deln = ZERO
    ELSEIF ((REAL(deln) >= REAL(dn)) .OR. (REAL(delt) >= REAL(dt)))  THEN
        Tn = ZERO
    ELSEIF (REAL(deln) >= REAL(deln_max)) THEN
        Tn = (Gam_t*(ONE-delt/dt)**beta*(delt/dt+n/beta)**n+dGtn) *     &
             Gam_n/dn*(m*(ONE-deln/dn)**alph*(m/alph+deln/dn)**(m-ONE) &
             -alph*(ONE-deln/dn)**(alph-ONE)*(m/alph+deln/dn)**m)
    ELSE
        Tn = (Gam_t*(ONE-delt/dt)**beta*(delt/dt+n/beta)**n+dGtn) *             &
             Gam_n/dn*(m*(ONE-deln_max/dn)**alph*(m/alph+deln_max/dn)**(m-ONE) &
               -alph*(ONE-deln_max/dn)**(alph-ONE)*(m/alph+deln_max/dn)**m) *  &
             deln/deln_max
    END IF
```

```
! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Cohesive Tangential Traction, Eq 12
IF ((REAL(deln) >= REAL(dn)) .OR. (REAL(delt) >= REAL(dt)))  THEN
    Tt = ZERO
ELSEIF (REAL(delt) >= REAL(delt_max)) THEN
    Tt = (Gam_n*(ONE-deln/dn)**alph*(deln/dn+m/alph)**m+dGnt) *     &
         Gam_t/dt*(n*(ONE-delt/dt)**beta*(delt/dt+n/beta)**(n-ONE) &
         -beta*(ONE-delt/dt)**(beta-ONE)*(delt/dt+n/beta)**n)
ELSE
    Tt = (Gam_n*(ONE-deln/dn)**alph*(deln/dn+m/alph)**m+dGnt) *             &
         Gam_t/dt*(n*(ONE-delt_max/dt)**beta*(delt_max/dt+n/beta)**(n-ONE) &
           -beta*(ONE-delt_max/dt)**(beta-ONE)*(delt_max/dt+n/beta)**n) *  &
         delt/delt_max
END IF

! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Algorithm 1 (from PPR paper)

! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Normal Cohesive Interaction
! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! ~~~~~~~~~~
! Contact
! ~~~~~~~~~~
IF (REAL(del(2)) < ZERO) THEN

    T_d(2,2) = -Gam_n/dn**TWO*(m/alph)**(m-ONE)*(alph+m)* &
               (Gam_t*(n/beta)**n + dGtn)
    T_d(2,1) = ZERO
    T(2,1)   = T_d(2,2)*del(2)

ELSE IF ((REAL(deln)<REAL(dn)).AND.(REAL(delt)<REAL(dt)).AND.(REAL(Tn)>=-1.0E-5)) THEN

    T(2,1) = Tn

    ! ~~~~~~~~~~~~~~~~~~~~
    ! Softening condition
    ! ~~~~~~~~~~~~~~~~~~~~
    IF (REAL(deln) >= REAL(deln_max)) THEN

        T_d(2,2) = (Gam_t*(ONE-delt/dt)**beta*(delt/dt+n/beta)**n+dGtn) *           &
                   Gam_n/dn**TWO *                                                 &
                   ((ONE-deln/dn)**(alph-TWO)*(alph-ONE)*alph*(deln/dn+m/alph)**m - &
                   TWO*(ONE-deln/dn)**(alph-ONE)*alph*(deln/dn+m/alph)**(m-ONE)*m + &
                   (ONE-deln/dn)**alph*(deln/dn+m/alph)**(m-TWO)*(m-ONE)*m)

         T_d(2,1) = Gam_t/dt*(-(ONE-delt/dt)**(beta-ONE)*beta*(delt/dt+n/beta)**n + &
                    (ONE-delt/dt)**beta*(delt/dt+n/beta)**(n-ONE)*n) * sign_dt *    &
                    Gam_n/dn*(-(ONE-deln/dn)**(alph-ONE)*alph*(deln/dn+m/alph)**m + &
                    (ONE-deln/dn)**alph*(deln/dn+m/alph)**(m-ONE)*m)

    ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ! Unloading/reloading condition
    ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ELSE

        T_d(2,2) = (Gam_t*(one-delt/dt)**beta*(delt/dt+n/beta)**n+dGtn) *             &
                   Gam_n/dn*((one-deln_max/dn)**alph*(deln_max/dn+m/alph)**(m-one)*m &
                   -(one-deln_max/dn)**(alph-one)*alph*(deln_max/dn+m/alph)**m)      &
                   / deln_max

        T_d(2,1) = Gam_t/dt*(-(one-delt/dt)**(beta-one)*beta*(delt/dt+n/beta)**n +   &
                   (one-delt/dt)**beta*(delt/dt+n/beta)**(n-one)*n) * sign_dt *      &
                   Gam_n/dn*(m*(one-deln_max/dn)**alph*(m/alph+deln_max/dn)**(m-one) &
                   -alph*(one-deln_max/dn)**(alph-one)*(m/alph+deln_max/dn)**m) *    &
                   deln/deln_max

    END IF

ELSE

    ! ~~~~~~~~~~~~~~~~~~
    ! Failure condition
    ! ~~~~~~~~~~~~~~~~~~

    T(2,1)   = ZERO
    T_d(2,2) = ZERO
```

```
            T_d(2,1) = ZERO

        END IF


        ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ! Tangential Cohesive Interaction
        ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        IF ((REAL(delt)<REAL(dt)) .AND. (REAL(deln)<REAL(dn)) .AND. (REAL(Tt)>=-1.0E-5)) THEN

            T(1,1) = Tt*sign_dt

            ! ~~~~~~~~~~~~~~~~~~~~
            ! Softening condition
            ! ~~~~~~~~~~~~~~~~~~~~
            IF (REAL(delt) >= REAL(delt_max)) THEN

                T_d(1,1) = (Gam_n*(ONE-deln/dn)**alph*(deln/dn+m/alph)**m+dGnt) *            &
                           Gam_t/dt**TWO *                                                   &
                           ((ONE-delt/dt)**(beta-TWO)*(beta-ONE)*beta*(delt/dt+n/beta)**n - &
                           TWO*(ONE-delt/dt)**(beta-ONE)*beta*(delt/dt+n/beta)**(n-ONE)*n + &
                           (ONE-delt/dt)**beta*(delt/dt+n/beta)**(n-TWO)*(n-ONE)*n)

                 T_d(1,2) = Gam_t/dt*(-(ONE-delt/dt)**(beta-ONE)*beta*(delt/dt+n/beta)**n + &
                           (ONE-delt/dt)**beta*(delt/dt+n/beta)**(n-ONE)*n) * sign_dt *     &
                           Gam_n/dn*(-(ONE-deln/dn)**(alph-ONE)*alph*(deln/dn+m/alph)**m + &
                           (ONE-deln/dn)**alph*(deln/dn+m/alph)**(m-ONE)*m)

            ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            ! Unloading/reloading condition
            ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            ELSE
                T_d(1,1) = (Gam_n*(ONE-deln/dn)**alph*(deln/dn+m/alph)**m+dGnt) *           &
                           Gam_t/dt*(n*(ONE-delt_max/dt)**beta*(delt_max/dt+n/beta)**(n-ONE) &
                                -beta*(ONE-delt_max/dt)**(beta-ONE)*(delt_max/dt+n/beta)**n) &
                           / delt_max

                 T_d(1,2) = Gam_n/dn*(-(ONE-deln/dn)**(alph-ONE)*alph*(deln/dn+m/alph)**m +   &
                           (ONE-deln/dn)**alph*(deln/dn+m/alph)**(m-ONE)*m) * sign_dt *       &
                           Gam_t/dt*(n*(ONE-delt_max/dt)**beta*(delt_max/dt+n/beta)**(n-ONE) &
                             -beta*(ONE-delt_max/dt)**(beta-ONE)*(delt_max/dt+n/beta)**n) *   &
                           delt/delt_max
            END IF

        ELSE

            ! ~~~~~~~~~~~~~~~~~~
            ! Failure condition
            ! ~~~~~~~~~~~~~~~~~~

            T(1,1)   = 0.0_DP
            T_d(1,1) = 0.0_DP
            T_d(1,2) = 0.0_DP

        ENDIF

    END SUBROUTINE Cohesive_PPR

    !****************************************************************************************************
    ! SUBROUTINE Coords_Transform
    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    !> @brief Find transformation matrix of the deformed shape
    !!
    !! @param[in]          nnode     Number of real nodes in the element (4)
    !! @param[in]          mcrd      Number of directions per node (x and y)
    !! @param[in]          COORD     Complex-valued nodal coordinates
    !! @param[in]          zU        Complex-valued displacement vector
    !! @param[in, out]     el_length Length of the cohesive element
    !! @param[in, out]     A         Transformation Matrix
    !****************************************************************************************************
    SUBROUTINE Coords_Transform(A, el_length, COORD, zU, nnode, mcrd)

        IMPLICIT NONE

        INTEGER, PARAMETER :: DP = SELECTED_REAL_KIND(15, 307)

        ! ~~~~~~~
        ! INPUTS
```

```
        INTEGER,     INTENT(IN)                          :: nnode, mcrd
        COMPLEX(DP), INTENT(IN), DIMENSION(mcrd,nnode)   :: COORD
        COMPLEX(DP), INTENT(IN), DIMENSION(nnode)        :: zU

        ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        ! OUTPUTS (to be computed within this subroutine)
        COMPLEX(DP), INTENT(INOUT)                         :: el_length
        COMPLEX(DP), INTENT(INOUT) , DIMENSION(mcrd,mcrd) :: A

        ! ~~~~~~~~~~~~~~~~~~
        ! Internal Variables
        COMPLEX(DP), DIMENSION(mcrd,nnode) :: Co_de   ! Coord. of a cohesive element in the deformed
            configuration
        COMPLEX(DP), DIMENSION(2,2)        :: Co_de_m ! Mid-points of a cohesive element to compute the
            orientation

        COMPLEX(DP) :: d_x, d_y, cos_a, sin_a
        INTEGER     :: i, j

        ! ~~~~~~~~~~~~~~~~~~
        ! Find Deformed Coords
        DO i = 1, mcrd
            DO j = 1, nnode
                Co_de(i,j) = COORD(i,j) + zU(2*(j-1)+i)
            END DO
        END DO

        ! Mid-points
        DO i = 1, 2
            Co_de_m(i,1) = (Co_de(i,1)+Co_de(i,4))*HALF
            Co_de_m(i,2) = (Co_de(i,2)+Co_de(i,3))*HALF
        END DO

        ! Direction cosines and A matrix
        d_x = Co_de_m(1,2) - Co_de_m(1,1)
        d_y = Co_de_m(2,2) - Co_de_m(2,1)

        el_length = (d_x**TWO + d_y**TWO)**HALF

        cos_a = d_x/el_length
        sin_a = d_y/el_length

        A(1,1) =  cos_a
        A(1,2) =  sin_a
        A(2,1) = -sin_a
        A(2,2) =  cos_a

    END SUBROUTINE Coords_Transform

END SUBROUTINE coh_UEL

!**********************************************************************************************************
! SUBROUTINE elastic_UEL
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
!> @author Daniel Ramirez-Tamayo, Arturo Montoya, Harry Millwater, The University of
!! Texas at San Antonio
!!
!> @brief 8 noded (4 real and 4 imag) linear elastic element with 4 integration points.
!!
!> @details This subroutine returns the Cauchy-Riemann version of both, stiffness matrix and RHS vector.
!! The variable to be perturbed is defined through the input file.
!!
!!
!! @param[in,out] rhs(mlvarx,1)            Contributions from the element to the right-hand-side vectors
!! @param[in,out] amatrx(ndofel, ndofel)   Stiffness matrix of the element
!! @param[in,out] svars(*)                 solution-dependent state variables
!! @param[in,out] energy(*)                Energy quantities associated with the element
!! @param[in]     ndofel                   Number of degrees of freedom in the element
!! @param[in]     nrhs                     Number of load vectors (1)
!! @param[in]     nsvars                   User-defined number of solution-dependent state variables
!! @param[in]     props(*)                 Properties assigned to the UEL
!! @param[in]     nprops                   User-defined number of real property values
!! @param[in]     coords(mcrd, nnode)      Original coordinates of the element
!! @param[in]     mcrd                     Number of coordinates per node (2)
!! @param[in]     nnode                    User-defined number of nodes on the element
!! @param[in]     u(ndofel)                Solution vector
!! @param[in]     du(mlvarx,*)             Incremental values of the solution vector
!! @param[in]     v(ndofel)                Time rate of change of the variables
!! @param[in]     a(ndofel)                Acceleration of the variables
```

```
!! @param[in]     jtype                    Integer defining the element type
!! @param[in]     time(2)                  Time step and total time
!! @param[in]     dtime                    Time increment
!! @param[in]     kstep                    Current step number
!! @param[in]     kinc                     Current increment number
!! @param[in]     jelem                    User-assigned element number
!! @param[in]     params(*)                Parameters associated with the solution procedure
!! @param[in]     ndload                   Identification number of the distributed load or flux
!! @param[in]     jdltyp(mdload,*)         To define distributed loads
!! @param[in]     adlmag(mdload,*)         Load magnitude of the K1th distributed load
!! @param[in]     predef(2, npredf, nnode) Values of the predefined field variables
!! @param[in]     npredf                   Number of predefined field variables,
!! @param[in]     lflags(*)                To define the current solution procedure
!! @param[in]     mlvarx                   Dimensioning parameter
!! @param[in]     ddlmag(mdload,*)         Increments in the magnitudes of the distributed loads
!! @param[in]     mdload                   Total number of distributed loads and/or fluxes
!! @param[in]     pnewdt                   Ratio of suggested new time increment to the current time
     increment
!! @param[in]     jprops(*)                Integer property values assigned to the UEL
!! @param[in]     njprop                   User-defined number of integer property values
!! @param[in]     period                   Time period of the current step
!**********************************************************************************************
SUBROUTINE elastic_UEL(rhs, amatrx, svars, energy, ndofel, nrhs, nsvars, props, &
    nprops, coords, mcrd, nnode, u, du, v, a, jtype, time, dtime, kstep,          &
    kinc, jelem, params, ndload, jdltyp, adlmag, predef, npredf, lflags,          &
    mlvarx, ddlmag, mdload, pnewdt, jprops, njprop, period)

    IMPLICIT NONE

    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
    ! Properties passed to the UEL
    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!

    ! PROPS(9):  Thickness of the element
    ! PROPS(10): Elastic Modulus
    ! PROPS(11): Poisson's Ratio

    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!
    ! Select double precision Real
    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!

    INTEGER, PARAMETER :: DP = SELECTED_REAL_KIND(15, 307)

    !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ! Abaqus Variables

    ! Scalar parameters

    REAL(DP) :: dtime                   ! Time increment
    REAL(DP) :: pnewdt                  ! Ratio of suggested new time increment to the current time
        increment
    REAL(DP) :: period                  ! Time period of the current step
    INTEGER  :: mlvarx                  ! Dimensioning parameter
    INTEGER  :: ndofel                  ! Number of degrees of freedom in the element
    INTEGER  :: nrhs                    ! Number of load vectors (1)
    INTEGER  :: nsvars                  ! User-defined number of solution-dependent state variables
    INTEGER  :: nprops                  ! User-defined number of real property values
    INTEGER  :: njprop                  ! User-defined number of integer property values
    INTEGER  :: mcrd                    ! Number of coordinates per node (2)
    INTEGER  :: nnode                   ! User-defined number of nodes on the element
    INTEGER  :: jtype                   ! Integer defining the element type
    INTEGER  :: kstep                   ! Current step number
    INTEGER  :: kinc                    ! Current increment number
    INTEGER  :: jelem                   ! User-assigned element number
    INTEGER  :: ndload                  ! Identification number of the distributed load or flux
    INTEGER  :: mdload                  ! Total number of distributed loads and/or fluxes
    INTEGER  :: npredf                  ! Number of predefined field variables,

    ! Arrays

    REAL(DP) :: props(*)                ! Properties assigned to the UEL
    INTEGER  :: jprops(*)               ! Integer property values assigned to the UEL
    REAL(DP) :: coords(mcrd, nnode)     ! Original coordinates of the element
    REAL(DP) :: u(ndofel)               ! Solution vector
    REAL(DP) :: du(mlvarx,*)            ! Incremental values of the solution vector
    REAL(DP) :: v(ndofel)               ! Time rate of change of the variables
    REAL(DP) :: a(ndofel)               ! Acceleration of the variables
    INTEGER  :: jdltyp(mdload,*)        ! To define distributed loads
    REAL(DP) :: adlmag(mdload,*)        ! Load magnitude of the K1th distributed load
```

```
REAL(DP) :: ddlmag(mdload,*)          ! Increments in the magnitudes of the distributed loads
REAL(DP) :: predef(2, npredf, nnode) ! Values of the predefined field variables
REAL(DP) :: params(*)                 ! Parameters associated with the solution procedure
INTEGER  :: lflags(*)                 ! To define the current solution procedure
REAL(DP) :: time(2)                   ! Time step and total time

! Variables to be defined

REAL(DP) :: rhs(mlvarx,1)             ! Contributions from the element to the right-hand-side vectors
REAL(DP) :: amatrx(ndofel, ndofel)    ! Stiffness matrix of the element
REAL(DP) :: svars(*)                  ! solution-dependent state variables
REAL(DP) :: energy(*)                 ! Energy quantities associated with the element
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! Parameters Useful for Calculations

PARAMETER  ZERO=0.0_DP,  ONE=1.0_DP,  NONE=-1.0_DP                        ! Zero, one, negative one
PARAMETER  HALF=0.5_DP,  TWO=2.0_DP, THREE= 3.0_DP, FOUR=4.0_DP          ! half, two, three, four
PARAMETER  gaussCoord=SQRT(3.0_DP)/3.0_DP, gaussWeight=1.0_DP, ninpt=4 ! For Gaussian integration
PARAMETER  nn_real    = 4                                                  ! Number of real nodes
PARAMETER  ndof_real = 2                                                   ! Number of real DOF


!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Variables used in the UEL

REAL(DP), DIMENSION(nn_real*mcrd, nn_real*mcrd) :: Kmat        ! Stiffness Matrix
REAL(DP), DIMENSION(nn_real*mcrd, nn_real*mcrd) :: Kmat_iter   ! Stiffness Matrix of the integration
     point
REAL(DP), DIMENSION(nn_real*mcrd, 1)            :: Fvec        ! Force Vector
REAL(DP), DIMENSION(3,3)                         :: Dmat        ! Constitutive Elastic Matrix
REAL(DP), DIMENSION(nn_real)                     :: shape_f     ! Shape Functions
REAL(DP), DIMENSION(mcrd,nn_real)                :: dShape      ! Derivatives of Shape Functions
REAL(DP), DIMENSION(3,nn_real)                   :: sd_mat      ! Strain-disp Matrix
REAL(DP), DIMENSION(4,nn_real*mcrd)              :: dN_mat      ! ! Expanded Shape Functions
REAL(DP), DIMENSION(3,nn_real*mcrd)              :: Bmat        ! Global displacement-strain relation
     matrix
REAL(DP), DIMENSION(mcrd,mcrd)                   :: xjac, xjaci ! Jacobian and inverse
REAL(DP), DIMENSION(4,4)                         :: iJac_mat
REAL(DP), DIMENSION(ninpt)                       :: weight      ! Gauss weight
REAL(DP), DIMENSION(ninpt, mcrd)                 :: coord       ! Gauss Coords
REAL(DP), DIMENSION(mcrd,nn_real)                :: coords_int  ! Internal Coords, Real-Only

! Scalars
REAL(DP) :: dvol                        ! Volume Differential
REAL(DP) :: Emod                        ! Elastic Modulus
REAL(DP) :: nu                          ! Poisson's Ratio
REAL(DP) :: th                          ! Thickness
REAL(DP) :: djac                        ! Determinant of the Jacobian
REAL(DP) :: g,h                         ! Coordinates of gauss int points
REAL(DP) :: CONS, CONS1, CONS2, CONS3 ! For Hooke's Law

! Integers
INTEGER :: PS_flag  ! Plane strain or plane stress
INTEGER :: kintk, I ! Counters
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Read Input Data
Emod = PROPS(2)
nu   = PROPS(3)
th   = PROPS(4)

! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Initialize Variables
RHS    = ZERO
AMATRX = ZERO
Kmat   = ZERO

! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Gaussian Points and Weights
coord(1,1) = -sqrt(THREE)/THREE
coord(2,1) = -sqrt(THREE)/THREE
coord(3,1) =  sqrt(THREE)/THREE
coord(4,1) =  sqrt(THREE)/THREE

coord(1,2) = -sqrt(THREE)/THREE
coord(2,2) =  sqrt(THREE)/THREE
coord(3,2) = -sqrt(THREE)/THREE
```

```
coord(4,2) =  sqrt(THREE)/THREE

weight(1) = ONE
weight(2) = ONE
weight(3) = ONE
weight(4) = ONE

! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
! Material Matrix

PS_flag = 1 ! 1->Plane Strain, 2-> Plane Stress

Dmat = ZERO

IF (PS_flag .eq. ONE) THEN
    ! Plane Strain Case
    CONS  = Emod/((ONE-NU*TWO)*(ONE+Nu))
    CONS1 = (ONE-NU)
    CONS2 = NU
    CONS3 = (ONE-TWO*NU)

    Dmat(1,1) = CONS*CONS1
    Dmat(1,2) = CONS*CONS2

    Dmat(2,1) = CONS*CONS2
    Dmat(2,2) = CONS*CONS1

    Dmat(3,3) = CONS*CONS3/TWO

ELSEIF (PS_flag .eq. TWO) THEN
    ! Plane Stress Case
    CONS  = Emod/(ONE-NU*NU)
    CONS1 = ONE
    CONS2 = NU
    CONS3 = (ONE-NU)

    Dmat(1,1) = CONS*CONS1
    Dmat(1,2) = CONS*CONS2

    Dmat(2,1) = CONS*CONS2
    Dmat(2,2) = CONS*CONS1

    Dmat(3,3) = CONS*CONS3/TWO

ENDIF

! For each integration point of the element:
DO kintk = 1,ninpt

    ! Variables that require an update in every loop through the integration points
    Kmat_iter = ZERO
    dShape    = ZERO
    shape_f   = ZERO
    xjac      = ZERO
    xjaci     = ZERO
    Bmat      = ZERO
    djac      = ZERO
    g         = ZERO
    h         = ZERO
    dvol      = ZERO
    sd_mat    = ZERO
    iJac_mat  = ZERO

    g = coord(kintk,1)! First natural coordinate
    h = coord(kintk,2)! Second natural coordinate

    ! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ! Shape Functions of the Quadrilateral Element
    shape_f(1) = (ONE-g)*(ONE-h)/FOUR
    shape_f(2) = (ONE+g)*(ONE-h)/FOUR
    shape_f(3) = (ONE+g)*(ONE+h)/FOUR
    shape_f(4) = (ONE-g)*(ONE-h)/FOUR

    ! Derivative d(Ni)/d(g) - ksi
    dShape(1,1) = -(ONE - h)/FOUR
    dShape(1,2) =  (ONE - h)/FOUR
    dShape(1,3) =  (ONE + h)/FOUR
    dShape(1,4) = -(ONE + h)/FOUR
```

```
        ! Derivative d(Ni)/d(h)  - eta
        dShape(2,1) = -(ONE - g)/FOUR
        dShape(2,2) = -(ONE + g)/FOUR
        dShape(2,3) =  (ONE + g)/FOUR
        dShape(2,4) =  (ONE - g)/FOUR


        ! ~~~~~~~~
        ! Jacobian
        coords_int = ZERO
        coords_int = coords(1:2,1:8)

        xjac = MATMUL(dShape, TRANSPOSE(coords_int))

        ! Jacobian determinant
        djac =    xjac(1,1)*xjac(2,2) - xjac(1,2)*xjac(2,1)

        !Check for Positive Determinant of the Jacobian and find the inverse
        IF (djac > ZERO) THEN
            ! Jacobian is positive - o.k.
            xjaci(1,1) =  xjac(2,2)/djac
            xjaci(2,2) =  xjac(1,1)/djac
            xjaci(1,2) = -xjac(1,2)/djac
            xjaci(2,1) = -xjac(2,1)/djac
        ELSE
            ! negative or zero Jacobian
            PRINT*, 'WARNING: element',jelem,'has neg. Jacobian'
        ENDIF

        dvol = weight(kintk)*djac*th

        ! Strain Displacement Relationship
        sd_mat = ZERO
        sd_mat(1,1) = ONE
        sd_mat(2,4) = ONE
        sd_mat(3,2) = ONE
        sd_mat(3,3) = ONE

        ! Expanded Inverse Jacobian Matrix
        iJac_mat(1:2,1:2) = xjaci(1:2,1:2)
        iJac_mat(3:4,3:4) = xjaci(1:2,1:2)

        ! EXPANDED SHAPE MATRIX
        dN_mat = ZERO
        dN_mat(1:2,1:7:2) = dShape
        dN_mat(3:4,2:8:2) = dShape

        ! B Matrix
        Bmat = MATMUL(sd_mat, MATMUL(iJac_mat, dN_mat))


        ! Stiffness Matrix
        Kmat_iter = MATMUL(TRANSPOSE(Bmat),MATMUL(Dmat,Bmat))*dvol

        Kmat = Kmat + Kmat_iter


    END DO ! Integration

    AMATRX(1:8,  1:8)   =  Kmat
    AMATRX(9:16, 9:16)  =  Kmat
    AMATRX(1:8,  9:16)  =  ZERO ! Symmetric Version of CR Matrix
    AMATRX(9:16, 1:8)   =  ZERO

    ! Right Hand Side Vector
    RHS(:,1) = -MATMUL(AMATRX, U)

END SUBROUTINE elastic_UEL
```

Listing 10: Input file for the verification example

```
*HEADING
Patch test, Mode I
*NODE
** Real nodes
1, 0.0, -0.1
2, 100, -0.1
3, 100, 0.0
4, 0.0, 0.0
```

```
5, 100, 100
6, 0.0, 100
** Imaginary nodes (offset=1000)
1001, 0.0, 0.0
1002, 0.0, 0.0
1003, 0.0, 0.0
1004, 0.0, 0.0
1005, 0.0, 0.0
1006, 0.0, 0.0
** Nodal sets
*NSET, NSET=NODES_RE, generate
1, 6
*NSET, NSET=NODES_IM, generate
1001, 1006
**
** User Defined Elements
**
*USER ELEMENT, TYPE=U1, NODE=8, COORDINATES=2, PROPERTIES=17, VARIABLES=8
1, 2
*ELEMENT, TYPE=U1, ELSET=ALL_ELEMS
1, 4, 3, 5, 6, 1004, 1003, 1005, 1006
2, 1, 2, 3, 4, 1001, 1002, 1003, 1004
** ~~~~~~~~~~~~~~~~~~~
** Loading Amplitude
** ~~~~~~~~~~~~~~~~~~~
*Amplitude, name=Amp-1
0., 0., 1., 0.03, 2., -0.01, 3.,  0.1
** ~~~~~~~~~~~~~~~~~~~
** Other nodal sets
** fof BC definition
** ~~~~~~~~~~~~~~~~~~~
*NSET, NSET=UP_RE
5, 6
*NSET, NSET=UP_IM
1005, 1006
*NSET, NSET=ROLLER_RE
2
*NSET, NSET=ROLLER_IM
1002
*NSET, NSET=PIN_RE
1
*NSET, NSET=PIN_IM
1001
** Everything should be in [mm]
*UEL PROPERTY, ELSET=ALL_ELEMS
**  Gn      Gt     T_n  T_t  alph  beta   ln      lt
   0.100, 0.200, 4,   3,    5,    1.6, 0.005, 0.005,
**  thick   Emod    nu    num_coh_elems,  h,    pert_flag
    10,    32.0e3, 0.20,     1,          1e-10,   1
** coh_elem_id
2
******************************************************
** ~~~~~~~~~~~~~~~~~~~
** Step definition
** ~~~~~~~~~~~~~~~~~~~
*STEP, NLGEOM, INC=4000, UNSYMM=YES
*STATIC
0.05, 3.0, 3e-05, 0.05
** ~~~~~~~~~~~~~~~~~~~
** Boundary Conditions
** ~~~~~~~~~~~~~~~~~~~
** Fixed sets
*BOUNDARY
ROLLER_RE, 2, 2
ROLLER_IM, 2, 2
PIN_RE,    1, 2
PIN_IM,    1, 2
** Moving sets
*Boundary, amplitude=Amp-1
UP_RE, 2, 2, 1.0
UP_IM, 2, 2, 0.0
** ~~~~~~~~~~~~~~~~~~~
** Prints to dat file
** ~~~~~~~~~~~~~~~~~~~
** Real nodes
*NODE PRINT, NSET=NODES_RE
U, RF
** Imaginary nodes
*NODE PRINT, NSET=NODES_IM
```

```
U, RF
*END STEP
```

References

[1] Mi Y, Crisfield M, Davies G, Hellweg H. Progressive delamination using interface elements. J Compos Mater 1998;32(14):1246–72.
[2] Shah SP, Swartz SE, Ouyang C. Fracture mechanics of concrete: applications of fracture mechanics to concrete, rock and other quasi-brittle materials. John Wiley & Sons; 1995.
[3] van Mier JGM, van Vliet MRA. Uniaxial tension test for the determination of fracture parameters of concrete: state of the art. Eng Fract Mech 2002;69(2):235–47. http://dx.doi.org/10.1016/S0013-7944(01)00087-X, URL http://www.sciencedirect.com/science/article/pii/S001379440100087X.
[4] Shen B, Paulino G. Direct extraction of cohesive fracture properties from digital image correlation: a hybrid inverse technique. Exp Mech 2011;51(2):143–63.
[5] Valoroso N, Sessa S, Lepore M, Cricrì G. Identification of mode-i cohesive parameters for bonded interfaces based on DCB test. Eng Fract Mech 2013;104:56–79. http://dx.doi.org/10.1016/j.engfracmech.2013.02.008, URL http://www.sciencedirect.com/science/article/pii/S0013794413000507.
[6] Taylor RL. FEAP-A finite element analysis program. 2014.
[7] Barenblatt GI. The formation of equilibrium cracks during brittle fracture. General ideas and hypotheses. Axially-symmetric cracks. J Appl Math Mech 1959;23(3):622–36.
[8] Bischof C, Khademi P, Mauer A, Carle A. Adifor 2.0: Automatic differentiation of fortran 77 programs. IEEE Comput Sci Eng 1996;3(3):18–32.
[9] Bischof CH, Roh L, Mauer-Oats AJ. Adic: an extensible automatic differentiation tool for ansi-c. Softw - Pract Exp 1997;27(12):1427–56.
[10] Phipps ET, Gay DM. Automatic differentation of C++ codes with sacado. Tech. rep. SAND2006-7054C, Albuquerque, NM (United States): Sandia National Laboratories (SNL-NM); 2006.
[11] Griewank A, Walther A. Evaluating derivatives: principles and techniques of algorithmic differentiation, Vol. 105. Siam; 2008.
[12] Hascoet L, Pascual V. The tapenade automatic differentiation tool: principles, model, and specification. ACM Trans Math Softw (TOMS) 2013;39(3):20.
[13] Pascual V, Hascoët L. Mixed-language automatic differentiation. Optim Methods Softw 2018;33(4–6):1192–206. http://dx.doi.org/10.1080/10556788.2018.1435650.
[14] Nelder JA, Mead R. A simplex method for function minimization. Comput J 1965;7(4):308–13.
[15] Park K, Paulino GH, Roesler JR. A unified potential-based cohesive model of mixed-mode fracture. J Mech Phys Solids 2009;57(6):891–908. http://dx.doi.org/10.1016/j.jmps.2008.10.003, URL http://www.sciencedirect.com/science/article/pii/S0022509608001713.
[16] Analysis of crack formation and crack growth in concrete by means of fracture mechanics and finite elements. Cem Concr Res 1976;6(6):773–81. http://dx.doi.org/10.1016/0008-8846(76)90007-7, URL http://www.sciencedirect.com/science/article/pii/0008884676900077.
[17] Wittmann F, Rokugo K, Brühwiler E, Mihashi H, Simonin P. Fracture energy and strain softening of concrete as determined by means of compact tension specimens. Mater Struct 1988;21(1):21–32.
[18] Yang Q, Thouless MD. Mixed-mode fracture analyses of plastically-deforming adhesive joints. Int J Fract 2001;110(2):175–87.
[19] Camanho PP, Davila CG, de Moura MF. Numerical simulation of mixed-mode progressive delamination in composite materials. J Compos Mater 2003;37(16):1415–38. http://dx.doi.org/10.1177/0021998303034505, arXiv:https://doi.org/10.1177/0021998303034505.
[20] Xu C, Siegmund T, Ramani K. Rate-dependent crack growth in adhesives: I. Modeling approach. Int J Adhes Adhes 2003;23(1):9–13.
[21] Xie D, Waas AM. Discrete cohesive zone model for mixed-mode fracture using finite element analysis. Eng Fract Mech 2006;73(13):1783–96.
[22] Park K, Paulino GH, Roesler J. Cohesive fracture model for functionally graded fiber reinforced concrete. Cem Concr Res 2010;40(6):956–65.
[23] Campilho R, Banea M, Neto J, da Silva L. Modelling adhesive joints with cohesive zone models: effect of the cohesive law shape of the adhesive layer. Int J Adhes Adhes 2013;44:48–56. http://dx.doi.org/10.1016/j.ijadhadh.2013.02.006, URL http://www.sciencedirect.com/science/article/pii/S0143749613000353.
[24] JMcGarry JP, Máirtín ÉÓ, Parry G, Beltz GE. Potential-based and non-potential-based cohesive zone formulations under mixed-mode separation and over-closure. Part I: Theoretical analysis. J Mech Phys Solids 2014;63:336–62. http://dx.doi.org/10.1016/j.jmps.2013.08.020, URL http://www.sciencedirect.com/science/article/pii/S0022509613001737.
[25] Spring DW, Paulino GH. A growing library of three-dimensional cohesive elements for use in ABAQUS. Eng Fract Mech 2014;126:190–216. http://dx.doi.org/10.1016/j.engfracmech.2014.04.004, URL http://www.sciencedirect.com/science/article/pii/S0013794414001003.
[26] Park K, Paulino GH. Computational implementation of the PPR potential-based cohesive model in ABAQUS: educational perspective. Eng Fract Mech 2012;93:239–62.
[27] ABAQUS. Abaqus finite element software. Providence, RI: Dassault Systèmes Simulia Corp.; 2015.
[28] Squire W, Trapp G. Using complex variables to estimate derivatives of real functions. SIAM Rev 1998;40(1):110–2.
[29] Millwater H, Wagner D, Baines A, Montoya A. A virtual crack extension method to compute energy release rates using a complex variable finite element method. Eng Fract Mech 2016;162:95–111. http://dx.doi.org/10.1016/j.engfracmech.2016.04.002, URL http://www.sciencedirect.com/science/article/pii/S0013794416301540.
[30] Aguirre-Mesa AM, Ramirez-Tamayo D, Garcia MJ, Montoya A, Millwater H. A stiffness derivative local hypercomplex-variable finite element method for computing the energy release rate. Eng Fract Mech 2019;218:106581. http://dx.doi.org/10.1016/j.engfracmech.2019.106581, URL http://www.sciencedirect.com/science/article/pii/S0013794419306666.
[31] Montoya A, Ramirez-Tamayo D, Millwater H, Kirby M. A complex-variable virtual crack extension finite element method for elastic-plastic fracture mechanics. Eng Fract Mech 2018;202:242–58. http://dx.doi.org/10.1016/j.engfracmech.2018.09.023, URL http://www.sciencedirect.com/science/article/pii/S0013794418306775.
[32] Tamayo DR, Montoya A, Millwater H. A virtual crack extension method for thermoelastic fracture using a complex-variable finite element method. Eng Fract Mech 2018;192:328–42. http://dx.doi.org/10.1016/j.engfracmech.2017.12.013, URL http://www.sciencedirect.com/science/article/pii/S0013794417309049.
[33] Ramirez Tamayo D, Montoya A, Millwater H. Complex-variable finite-element method for mixed mode fracture and interface cracks. AIAA J 2018;56(11):4632–7. http://dx.doi.org/10.2514/1.J057231, URL arXiv:https://doi.org/10.2514/1.J057231.
[34] Dugdale DS. Yielding of steel sheets containing slits. J Mech Phys Solids 1960;8(2):100–4.
[35] Hillerborg A, Modéer M, Petersson P-E. Analysis of crack formation and crack growth in concrete by means of fracture mechanics and finite elements. Cem Concr Res 1976;6(6):773–81.
[36] Ingraffea AR, Gerstk WH, Gergely P, Saouma V. Fracture mechanics of bond in reinforced concrete. J Struct Eng 1984;110(4):871–90.
[37] Xu X-P, Needleman A. Numerical simulations of fast crack growth in brittle solids. J Mech Phys Solids 1994;42(9):1397–434. http://dx.doi.org/10.1016/0022-5096(94)90003-5, URL http://www.sciencedirect.com/science/article/pii/0022509694900035.
[38] Mergheim J, Kuhl E, Steinmann P. A finite element method for the computational modelling of cohesive cracks. Internat J Numer Methods Engrg 2005;63(2):276–89.
[39] Camacho G, Ortiz M. Computational modelling of impact damage in brittle materials. Int J Solids Struct 1996;33(20):2899–938. http://dx.doi.org/10.1016/0020-7683(95)00255-3, URL http://www.sciencedirect.com/science/article/pii/0020768395002553.

[40] Park K, Choi H, Paulino GH. Assessment of cohesive traction-separation relationships in ABAQUS: A comparative study. Mech Res Commun 2016;78:71–8. http://dx.doi.org/10.1016/j.mechrescom.2016.09.004, URL http://www.sciencedirect.com/science/article/pii/S0093641316301628 Recent Advances in Multiscale, Multifunctional and Functionally Graded Materials.
[41] Needleman A. A continuum model for void nucleation by inclusion debonding. ASME J Appl Mech 1987;54(3):525–31.
[42] Needleman A. An analysis of tensile decohesion along an interface. J Mech Phys Solids 1990;38(3):289–324. http://dx.doi.org/10.1016/0022-5096(90)90001-K, URL http://www.sciencedirect.com/science/article/pii/002250969090001K.
[43] Beltz GE, Rice J. Dislocation nucleation versus cleavage decohesion at crack tips. Model Deformation Cryst Solids 1991;457–80.
[44] Freed Y, Banks-Sills L. A new cohesive zone model for mixed mode interface fracture in bimaterials. Eng Fract Mech 2008;75(15):4583–93. http://dx.doi.org/10.1016/j.engfracmech.2008.04.013, URL http://www.sciencedirect.com/science/article/pii/S0013794408001100.
[45] Park K, Paulino GH. Cohesive zone models: A critical review of traction-separation relationships across fracture surfaces. Appl Mech Rev 2013;64(6):060802. http://dx.doi.org/10.1115/1.4023110, URL arXiv:https://asmedigitalcollection.asme.org/appliedmechanicsreviews/article-pdf/64/6/060802/6073587/amr_64_6_060802.pdf.
[46] Park K. Potential-based fracture mechanics using cohesive zone and virtual internal bond modeling. University of Illinois at Urbana-Champaign; 2009.
[47] Voorhees A, Millwater H, Bagley R. Complex variable methods for shape sensitivity of finite element models. Finite Elem Anal Des 2011;47(10):1146–56. http://dx.doi.org/10.1016/j.finel.2011.05.003, URL http://www.sciencedirect.com/science/article/pii/S0168874X11000990.
[48] Anderson WK, Newman JC, Whitfield DL, Nielsen EJ. Sensitivity analysis for Navier-Stokes equations on unstructured meshes using complex variables. AIAA J 2001;39(1):56–63.
[49] Burg CE, Newman III JC. Computationally efficient, numerically exact design space derivatives via the complex taylor's series expansion method. Comput & Fluids 2003;32(3):373–83. http://dx.doi.org/10.1016/S0045-7930(01)00044-5, URL http://www.sciencedirect.com/science/article/pii/S0045793001000445.
[50] Montoya A, Fielder R, Gomez-Farias A, Millwater H. Finite-element sensitivity for plasticity using complex variable methods. J Eng Mech 2015;141(2):04014118. http://dx.doi.org/10.1061/(ASCE)EM.1943-7889.0000837.
[51] Wang BP, Apte AP. Complex variable method for eigensolution sensitivity analysis. AIAA J 2006;44(12):2958–61.
[52] Garza J, Millwater H. Multicomplex newmark-beta time integration method for sensitivity analysis in structural dynamics. AIAA J 2015;53(5):1188–98. http://dx.doi.org/10.2514/1.J053282, URL arXiv:https://doi.org/10.2514/1.J053282.
[53] Lantoine G, Russell RP, Dargent T. Using multicomplex variables for automatic computation of high-order derivatives. ACM Trans Math Software 2012;38(3):16:1–21. http://dx.doi.org/10.1145/2168773.2168774, URL http://doi.acm.org/10.1145/2168773.2168774.
[54] Millwater HR, Shirinkam S. Multicomplex taylor series expansion for computing high order derivatives. Int J Appl Math 2014;27(4):311–34.
[55] Gomez-Farias A, Montoya A, Millwater H. Complex finite element sensitivity method for creep analysis. Int J Press Vessel Pip 2015;132–133:27–42. http://dx.doi.org/10.1016/j.ijpvp.2015.05.006, URL http://www.sciencedirect.com/science/article/pii/S0308016115000587.
[56] Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. Nat Methods 2020;17(3):261–72.
[57] Nocedal J, Wright S. Numerical optimization. Springer Science & Business Media; 2006.
[58] Fike J, Alonso J. The development of hyper-dual numbers for exact second-derivative calculations. In: 49th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition. 2011, p. 886.