Implementation of phase-field model of ductile fracture

Release 1

Sai Viswanadha Sastry Upadhyayula

Oct 12, 2022

Contents:

1 Main file for staggered scheme

1.1 Implementation of phase-field model of ductile fracture

Phase-field model for elastic and elastic-plastic case is implemented using staggered scheme.

The whole program is divided into pre-processing, processing and post.processing stages.

1. **Pre-processing:** The pre-processing stage invloves the following steps,
   - Defining geometry, generating nodes and elements
   - Calling input data
   - Initialize all arrays
   - Generating boundary points
2. **Processing:** The processing stage involves the following steps,
   - Calculating Gauss quadrature points and weights
   - Calculating stiffness tensor (for elastic case)
   - Loop for all time steps
     - Compute strain energy and check with the previous value and update accordingly
     - Compute stiffness element matrices and residual vectors for every element for phase-field order parameter
     - Assemble element matrices and vectors into global stiffness matrix and residual vector
     - Newton-Raphson iteration loop for phase field order parameter
       * Solve the matrix equation for order parameter values
       * Compute element residual vector and assemble into global residual vector
       * Check for convergence
     - Newton-Raphson iteration loop for displacement

        * Compute stiffness element matrices and residual vectors for every element for phase-field order parameter
        * Assemble element matrices and vectors into global stiffness matrix and residual vector
        * Apply essential boundary conditions to global stiffness matrix and residual vector
        * Slove the matrix equation for displacement values
        * Update displacement values by inclusing essential boundary conditions
        * Compute stress and strain values
        * Check for convergence

3. **Post-processing:** The post-processing stage invloves the following steps,
    - Generating phase field order parameter distribution contour plot
    - Generating load vs displacemnt curve

```
main_staggered.main()
```

2 Inputs

class input_abaqus.**input_parameters**

Bases: `object`

Input parameters class where all inputs are defined. Four different functions are implemented based on the input type.

geometry_parameters()

Input parameters related to geometry are defined. **Nodes** and **elements** of a geometry are generated in ABAQUS.

Returns

- **num_dim** (*int*) – Dimension of a problem.
- **num_node_elem** (*int*) – Number of nodes per element.
- **nodes** (*Array of float64, size(num_node,num_dim)*) – Co-ordinates of all field nodes.
- **num_node** (*int*) – Number of nodes.
- **elements** (*Array of int, size(num_elem,num_node_elem)*) – Element connectivity matrix.
- **num_elem** (*int*) – Total number of elements.
- **num_dof** (*int*) – Total number of DOFs per node.
- **num_Gauss** (*int*) – Total number of Gauss points used for integration in 1-dimension.
- **num_stress** (*int*) – Number of independent stress components.
- **problem** (*str*) – Problem type either 'Elastic' or 'Elastic_Plastic'.

material_parameters_elastic()

Material specific input parameters are defined. This function is for **elastic** problem.

Returns

- **k_const** (*float64*) – Parameter to avoid overflow for cracked elements.
- **G_c** (*float64*) – Critical energy release for unstable crack or damage.
- **l_0** (*float64*) – Length parameter which controls the spread of damage.

- **Young** (*float64*) – Youngs modulus.
- **Poisson** (*float64*) – Poissons ratio.
- **stressState** (*int*) – If stressState == 1 plane stress; if stressState = 2 plane strain.

`material_parameters_elastic_plastic()`

Material specific input parameters are defined. This function is for **elastic-plastic** problem.

Returns

- **k_const** (*float64*) – Parameter to avoid overflow for cracked elements.
- **G_c** (*float64*) – Critical energy release for unstable crack or damage.
- **l_0** (*float64*) – Length parameter which controls the spread of damage.
- **stressState** (*int*) – If stressState == 1 plane stress; if stressState = 2 plane strain.
- **shear** (*float64*) – Shear modulus.
- **bulk** (*float64*) – Bulk modulus.
- **sigma_y** (*float64*) – Yield stress.
- **hardening** (*float64*) – Hardening modulus.

`time_integration_parameters()`

Inputs for time integration parameters are defined.

Returns

- **num_step** (*int*) – Number of time steps.
- **max_iter** (*int*) – Maximum number of Newton-Raphson iterations.
- **max_tol** (*float64*) – Tolerance for iterative solution.
- **disp_inc** (*float64*) – Displacmenet increment per time steps.

3 Gauss points

geometry.**quadrature_coefficients**(*numgauss: int*)

Function to get Gaussian quadrature points and weights.

Gaussian points and weights for 1-Dimension are called from inbuilt function using **numpy**.

Using these points and weights in 1-Dimension, points and weights are computed in 2-Dimension respectively.

Parameters

numgauss (*int*) – number of Guass points for background integration.

Returns

- **Points** (*Array of float64, size(num_dim,num_Gauss**num_dim)*) – Gauss points used for integration.
- **Weights** (*Array of float64, size(num_Gauss**num_dim)*) – Weights for Gauss points used for integration.

4

Material Routine

class material_routine.**material_routine**(*problem: str*)

Bases: object

Depends on the problem type respective variables are returned.

If the problem is **elastic**, stiffness tensor, which is computed either using plane stress or plane strain function, returned.

If the problem is **elastic-plastic**, radial return stress-update algorithm is implemented and updated stress, stiffness tensor and internal variables are returned.

Associative von Mises plasticity with linear isotropic hardening model has been implemented.

material_elasticity()

Stiffness tensor computed using plane strain or plane stress function is called and returned.

Returns
Ce – Stiffness tensor.

Return type
Array of float64, size(3,3)

material_plasticity(*strain: array*, *strain_plas: array*, *alpha: float*)

The radial return stress-update algorithm for a **von Mises** plasticity model with linear **isotropic** hardening is implemented.

This particular code has been used from my plasticity course exercise, where I have implemented it on my own.

Parameters

- **strain** (*Array of float64, size(num_stress)*) – Total strain at current step .
- **strain_plas** (*Array of float64, size(num_stress)*) – Plastic strain at old step.
- **alpha** (*float64*) – Scalar hardening variable at old step.

Returns

- **stress_red** (*Array of float64, size(num_stress)*) – Updated stress at current step .
- **C_red** (*Array of float64, size(3,3)*) – Updated stiffness tensor.
- **strain_plas_red** (*Array of float64, size(num_stress)*) – Plastic strain at current step .
- **alpha** (*float64*) – Scalar hardening variable at current step.

`planestrain`()

Calculates stiffness tensor for plane strain condition.

Returns
Ce – Stiffness tensor for plane strain case.

Return type
Array of float64, size(3,3)

`planestress`()

Calculates stiffness tensor for plane stress condition.

Returns
Ce – Stiffness tensor for plane stress case.

Return type
Array of float64, size(3,3)

5 Tensor algebra

class tensor.**tensor**

Bases: `object`

This class consists of some tensor operations.

This part of code is directly taken from **Prof. Kiefer**, provided in the plasticity exercise.

P4sym()

Function to compute fourth order deviatoric projection tensor.

Returns
P4sym – Fourth order deviatoric projection tensor.

Return type
Array of float64, size(3,3,3,3)

fourth_to_three(*C: array*)

Function to reshape fourth order tensor from (3 x 3 x 3 x 3) to (6 x 6). Then (6 x 6) order tensor is reduced to (3 x 3) order tensor based on plane strain conditions.

Parameters
C (*Array of float64, size(3,3,3,3)*) – Fourth order tensor.

Returns
C_red – Second order tensor.

Return type
Array of float64, size(3,3)

6 Element routine

class element_staggered.**element_staggered**(*elem*, *Points*, *Weights*, *disp*, *Phi*, *stress*, *strain_energy*, *elements*, *nodes*, *num_dof*, *num_node_elem*, *num_Gauss_2D*)

Bases: object

Class to compute element stiffness tensor for displacments and phase field order parameter,

element residual for phase field order parameter, element internal force for displacements.

This element routine is computed based on staggered scheme.

element_residual_field_parameter(*G_c*, *l_0*)

Function for calculating residual vector for order parameter

Parameters

- **G_c** (*float64*) – critical energy release for unstable crack or damage.
- **l_0** (*float64*) – length parameter which controls the spread of damage.

Returns

residual_phi – residual vector for order parameter.

Return type

Array of float64, size(num_elem_var_phi)

element_stiffness_displacement(*C: array*, *k_const: float*)

Function for calculating element stiffness matrix and internal force vector for displacement for elastic case.

Parameters

- **C** (*Array of float64, size(k)*) – Stiffness tensor.
- **k_const** (*float64*) – parameter to avoid overflow for cracked elements.

Returns

- **K_uu** (*Array of float64, size(num_elem_var_u,num_elem_var_u)*) – element stiffness matrix.
- **F_int_elem** (*Array of float64, size(num_elem_var_u)*) – element internal force vector.

element_stiffness_displacement_plasticity(*k_const: float, strain: array, strain_plas: array, alpha: float*)

Function for calculating element stiffness matrix and internal force vector for displacement for elastic-plastic case and updates stress, plastic strain and hardening variable from material routine.

Parameters

- **k_const** (*float64*) – Parameter to avoid overflow for cracked elements.
- **strain** (*Array of float64, size(num_elem,num_Gauss_2D, num_stress)*) – Strain at the integration points for all elements.
- **strain_plas** (*Array of float64, size(num_elem,num_Gauss_2D, num_stress)*) – Plastic strain at the previous step at integration points for all elements.
- **alpha** (*float64*) – Scalar hardening variable at previous step.

Returns

- **K_uu** (*Array of float64, size(num_elem_var_u,num_elem_var_u)*) – element stiffness matrix.
- **F_int_elem** (*Array of float64, size(num_elem_var_u)*) – element internal force vector.
- **stress** (*Array of float64, size(num_elem,num_Gauss_2D,num_stress)*) – Stress at the integration points for all elements.
- **strain_plas** (*Array of float64, size(num_elem,num_Gauss_2D,num_stress)*) – Plastic strain at current step at integration points for all elements.
- **alpha** (*float64*) – Scalar hardening variable at current step.

element_stiffness_field_parameter(*G_c, l_0*)

Function for calculating element stiffness matrix and residual vector for phase field order parameter (phi).

Parameters

- **G_c** (*float64*) – critical energy release for unstable crack or damage.
- **l_0** (*float64*) – length parameter which controls the spread of damage.

Returns

- **K_phiphi** (*Array of float64, size(num_elem_var_phi,num_elem_var_phi)*) – element stiffness matrix for phase-field order parameter.
- **residual_phi** (*Array of float64, size(num_elem_var_phi)*) – element residual vector for phase-field order parameter.

7

Solving stress, strain and strain energy

class solve_stress_strain.**solve_stress_strain**(*num_Gauss_2D*, *num_stress*, *num_node_elem*, *num_dof_u*, *elements*, *disp*, *Points*, *nodes*, *C*, *strain_energy*, *strain_plas*)

Bases: `object`

Class to compute stress, strain and strain energy at integration points for each element.

Strain energy is computed and compared to previous value and updated accordingly. It is computed using elastic strain and stress.

solve()

Function for solving stress, strain and strain energy at integration points for each element

Return type
None.

property solve_strain

Calls solve class and returns strain

Returns
Strain at the integration points for all elements.

Return type
Array of float64, size(num_elem,num_Gauss_2D,num_stress)

property solve_strain_energy

Calls solve class and returns strain energy

Returns
Strain energy at the integration points for all elements.

Return type
Array of float64, size(num_elem,num_Gauss_2D)

property solve_stress

Calls solve class and returns stress

Returns
Stress at the integration points for all elements.

Return type
Array of float64, size(num_elem,num_Gauss_2D,num_stress)

8

Shape function

class shape_function.**shape_function**(*num_node_elem: int*, *gpos: list*, *elem_coord: array*)

Bases: `object`

Class for computing shape function, derivatives of shape function and determinant of Jacobian.

Shape functions are computed using Lagrange interpolant basis.

Derivatives of shape functions are computed using gradients of Lagrange interpolant basis.

Lagrange interpolant basis are implemented for four different cases.

1. Two node line element.
2. Three node triangular element.
3. Four node quadrilateral element.
4. Eight node quadrilateral element.

get_det_Jacobian()

Function to compute determinant of Jacobian matrix.

Raises
ValueError – Determinant of Jacobian matrix should be positive.

Returns
Determinant of Jacobian matrix.

Return type
float64

get_shape_function()

Function to compute shape functions using Lagrange interpolant basis.

Returns
Shape functions.

Return type
Array of float64

get_shape_function_derivative()

Function to compute derivatives of shape functions using inverse of Jacobian matrix and gradients of Lagrange interpolant basis.

Returns

Derivatives of shape functions.

Return type

Array of float64

9

B matrix

class Bmatrix.**Bmatrix**(*dNdX: array*, *num_node_elem: int*)

Bases: `object`

Class for computing B matrix for displacement and phase field order parameter.

B matrix for displacement is the connectivity matrix for strain and displacement.

B matrix for phase-field order parameter is the connectivity matrix for order parameter and its gradient. It is nothing but the derivatives of shape functions.

Bmatrix_disp()

Function for computing B matrix which is strain and displacement connectivity matrix.

Parameters

- **dNdX** (*Array of float64, size(num_dim,num_node_elem)*) – Derivatives of shape functions w.r.t. global coordinates.
- **num_node_elem** (*int*) – Number of nodes per element, possible nodes are 2,3,4 and 8 per element.

Returns

B – Strain and displacement connectivity matrix of a node.

Return type

Array of float64, size(3,2*num_node_elem)

Bmatrix_phase_field()

Function for computing B matrix which is the connectivity matrix for order parameter and its gradient.

Returns

B – Order parameter and its gradient connectivity matrix of a node.

Return type

Array of float64, size(num_dim,num_node_elem)

10

Indices to assemble global matrix

class assembly_index.**assembly**

Bases: `object`

Class for computing indices to assemble global matrices.

assembly_index_phi(*elem: int, num_dof_phi: int, num_node_elem: int, elements: array*)

Function to generate a vector which consists indices of global matrix for respective element belongs to order parameter for staggered scheme.

Parameters

- **elem** (*int*) – Element number.
- **num_dof_phi** (*int*) – Number of degrees of freedom of order parameter.
- **num_node_elem** (*int*) – Number of nodes per element, possible nodes are 2,3,4 and 8 per element.
- **elements** (*Array of float64, size(num_elem,num_node_elem)*) – Element connectivity matrix.

Returns

index – A vector which consists indices of global matrix for respective element belongs to order parameter.

Return type

Array of float64, size(num_dof_elem)

assembly_index_phi_monolithic(*elem: int, num_dof_phi: int, num_tot_var_u: int, num_node_elem: int, elements: array*)

Function to generate a vector which consists indices of global matrix for respective element belongs to order parameter for monolithic scheme.

Parameters

- **elem** (*int*) – Element number.
- **num_dof_phi** (*int*) – Number of degrees of freedom of order parameter.
- **num_tot_var_u** (*int*) – Total number of variables for displacements.

- **num_node_elem** (*int*) – Number of nodes per element, possible nodes are 2,3,4 and 8 per element.
- **elements** (*Array of float64, size(num_elem,num_node_elem)*) – Element connectivity matrix.

Returns
index – A vector which consists indices of global matrix for respective element belongs to order parameter.

Return type
Array of float64, size(num_dof_elem)

assembly_index_u(*elem: int, num_dof_u: int, num_node_elem: int, elements: array*)

Function to generate a vector which consists indices of global matrix for respective element belongs to displacement.

Parameters

- **elem** (*int*) – Element number.
- **num_dof_u** (*int*) – Number of degrees of freedom of displacement.
- **num_node_elem** (*int*) – Number of nodes per element, possible nodes are 2,3,4 and 8 per element.
- **elements** (*Array of float64, size(num_elem,num_node_elem)*) – Element connectivity matrix.

Returns
index – A vector which consists indices of global matrix for respective element belongs to displacement.

Return type
Array of float64, size(num_dof_elem)

11

Post processing

plots.**plot_field_parameter**(*nodes*, *phi*, *tot_inc*)

Function to plot contour plot of field parameter (phi)

Parameters

- **nodes** (*Array of float64, size(numnode,num_dim)*) – coordinates of all field nodes.
- **phi** (*Array of float64, size(num_tot_var_phi)*) – Phase field order parameter.
- **tot_inc** (*float64*) – Applied displacement.

Return type
None.

plots.**plot_field_parameter_mono**(*nodes*, *phi*, *tot_inc*)

Function to plot contour plot of field parameter (phi) for monolithic scheme.

Parameters

- **nodes** (*Array of float64, size(numnode,num_dim)*) – coordinates of all field nodes.
- **phi** (*Array of float64, size(num_tot_var_phi)*) – Phase field order parameter.
- **tot_inc** (*float64*) – Applied displacement.

Return type
None.

plots.**plot_load_displacement**(*disp*, *force*)

Function to generate load vs displacement curve

Parameters

- **disp** (*Array of float64, size(num_step)*) – Displacements.
- **force** (*Array of float64, size(num_step)*) – Load.

Return type
None.

plots.**plot_load_displacement_mono**(*disp, force*)

Function to generate load vs displacement curve for monolithic scheme.

Parameters

- **disp** (*Array of float64, size(num_step)*) – Displacements.
- **force** (*Array of float64, size(num_step)*) – Load.

Return type
None.

plots.**plot_nodes_and_boundary**(*nodes*)

Function to plot field nodes and boundary edges

Parameters
nodes (*Array of float64, size(numnode,nx)*) – coordinates of all field nodes.

Return type
None.

12 Tests

class tests.**tests**

Bases: object

test_B_matrix_true()

UNIT TESTING Aim: Test B matrix which is the strain displacement connectivity matrix in Bmatrix class

Expected result : Array of B matrix

Test command : pytest test.py::test_B_matrix_true()

Remarks : test case passed successfully

test_Jacobian_true()

UNIT TESTING Aim: Test determinant of Jacobian in shape_function class

Expected result : Determinant of a Jacobian matrix

Test command : pytest test.py::test_Jacobian_true()

Remarks : test case passed successfully

test_element_area_true()

UNIT TESTING Aim: Test area of an element using determinant of Jacobian

Expected result : Area of an element

Test command : pytest test.py::test_element_area_true()

Remarks : test case passed successfully

test_plane_strain_true()

UNIT TESTING Aim: Test plane strain stiffness tensor from planestrain method in material_routine class

Expected result : Array of stiffness tensor

Test command : pytest test.py::test_plane_strain_true

Remarks : test case passed successfully

test_plane_stress_true()

UNIT TESTING Aim: Test plane stress stiffness tensor from planestress method in material_routine class

Expected result : Array of stiffness tensor

Test command : pytest test.py::test_plane_stress_true

Remarks : test case passed successfully

test_quadrature_coefficients_true()

UNIT TESTING Aim: Test quadrature points and weights which is calculated using quadrature_coefficients in geometry

Expected result : Array of Guass points and weights

Test command : pytest test.py::test_quadrature_coefficients_true()

Remarks : test case passed successfully

test_shape_function_derivative_true()

UNIT TESTING Aim: Test shape function derivatives in shape_function class

Expected result : Array of shape function derivatives

Test command : pytest test.py::test_shape_function_derivative_true()

Remarks : test case passed successfully

test_shape_function_true()

UNIT TESTING Aim: Test shape functions in shape_function class

Expected result : Array of a shape function

Test command : pytest test.py::test_shape_function_true()

Remarks : test case passed successfully

13

Testing one element

test_one_element.**test_B_matrix_disp_true**()

UNIT TESTING Aim: Test B matrix for displacement which is the strain displacement connectivity matrix

Expected result : Array of B matrix for displacement

Test command : pytest test_one_element.py::test_B_matrix_true()

Remarks : test case passed successfully

test_one_element.**test_B_matrix_phasefield_true**()

UNIT TESTING Aim: Test B matrix for fiels order parameter which is derivative of shape function

Expected result : Array of B matrix

Test command : pytest test_one_element.py::test_B_matrix_phasefield_true()

Remarks : test case passed successfully

test_one_element.**test_det_Jacobian_true**()

UNIT TESTING Aim: Test determinant of Jacobian in shape_function class

Expected result : Determinant of a Jacobian

Test command : pytest test_one_element.py::test_det_Jacobian_true()

Remarks : test case passed successfully

test_one_element.**test_eigen_values_stiffness_disp_positive_true**()

UNIT TESTING Aim: Test sign of eigen values of stiffness matrix for field order parameter

Expected result : Array of sign of eigen values

Test command : pytest test_one_element.py::test_element_area_true()

Remarks : test case passed successfully

test_one_element.**test_eigen_values_stiffness_phasefield_positive_true**()

UNIT TESTING Aim: Test sign of eigen values of stiffness matrix for field order parameter

Expected result : Array of sign of eigen values

Test command : pytest test_one_element.py::test_eigen_values_stiffness_phasefield_positive_true()

Remarks : test case passed successfully

test_one_element.**test_element_area_true**()

UNIT TESTING Aim: Test area of an element using determinant of Jacobian

Expected result : Area of an element

Test command : pytest test_one_element.py::test_element_area_true()

Remarks : test case passed successfully

test_one_element.**test_global_assembly_stiffness_disp_true**()

UNIT TESTING Aim: Test assembly of a global stiffness matrix

Expected result : Array of global stiffness matrix

Test command : pytest test_one_element.py::test_global_assembly_stiffness_disp_true()

Remarks : test case passed successfully

test_one_element.**test_shape_function_derivative_true**()

UNIT TESTING Aim: Test shape function derivatives in shape_function class

Expected result : Array of shape function derivative

Test command : pytest test_one_element.py::test_shape_function_derivative_true()

Remarks : test case passed successfully

test_one_element.**test_shape_function_true**()

UNIT TESTING Aim: Test shape functions in shape_function class

Expected result : Array of shape function

Test command : pytest test_one_element.py::test_shape_function_true()

Remarks : test case passed successfully

test_one_element.**test_stiffness_disp_symmetry**()

UNIT TESTING Aim: Test symmetry of a stiffness matrix

Expected result : Array which has size of stiffness matrix consists of all True

Test command : pytest test_one_element.py::test_element_area_true()

Remarks : test case passed successfully

test_one_element.**test_stiffness_disp_true**()

UNIT TESTING Aim: Test stiffness matrix for displacement

Expected result : Array of stiffness matrix

Test command : pytest test_one_element.py::test_stiffness_disp_true()

Remarks : test case passed successfully

test_one_element.**test_stiffness_phasefield_symmetry**()

UNIT TESTING Aim: Test symmetry of a stiffness matrix

Expected result : Array which has size of stiffness matrix consists of all True

Test command : pytest test.py::test_stiffness_phasefield_symmetry()

Remarks : test case passed successfully

test_one_element.**test_stiffness_phasefield_true**()

UNIT TESTING Aim: Test stiffness matrix of order parameter.

Expected result : Array of stiffness matrix for phase field order parameter

Test command : pytest test_one_element.py::test_stiffness_phasefield_true()

Remarks : test case passed successfully

Python Module Index