Personal Programming Project 2021-22

IMPLEMENTATION OF EXTENDED FINITE ELEMENT METHOD TO ANALYSE THE STRESS FIELD NEAR THE CRACK

TECHNISCHE UNIVERSITÄT BERGAKADEMIE FREIBERG

Die Ressourcenuniversität. Seit 1765.

Vikas Melkot Balasubramanyam

Computational Materials Science

66133

Professors In charge

Dr.-Ing. Arun Prakash

Dr.-Ing. Stefan Prüger

Institution of Mechanics and Fluid Dynamics

March 27, 2022

Contents

List of Figures

List of Tables

1 Abstract

A new and improved technique called Extended Finite Element Method (XFEM) for modelling of cracks in the finite element framework has been presented. Standard displacements are enriched near a crack by incorporating both discontinuous fields and the near tip asymptotic fields through a partition of unity method[3].

The nodes that are present around the crack segments will be enriched giving rise to additional degree of freedoms. These additional DOFs are used to approximate displacements of the corresponding nodes. Displacements and stresses are computed after solving a BVP.

The output from the XFEM will be used as the inputs to interaction integral to compute stress intensity factor. Finally, the stress contour plots have been analysed. Numerical experiments are provided to demonstrate the correctness and robustness of the implemented technique. Time analysis has been done and plotted as a pie chart to check where exactly the bottleneck is present. A table of milestones achieved has been presented. Advantages and disadvantages of the program has been listed. In report has been concluded with some remarks and findings.

2 Introduction

History has witnessed numerous scenarios of structural failure due to existence of cracks. Buildings, dams, bridges, airplanes, railroads, and many other constructions have been impacted with this phenomenon that has the catastrophic effects in the world[4].

Many scientists and engineers have come with the solutions and strategies to decrease the effects of cracks and prevent their occurrences in the structures. One can establish an analytical solution for simple cases. The real-life problems encountered in this front require complex solutions that can be obtained by computational techniques[4]. The Finite element method plays a vital role in this regard. However, it is difficult to model the cracks (discontinuities in the displacements) as accurately as possible using this technique.

In order to model these discontinuities and inaccuracies the extended finite element method (XFEM) was developed in 1999 by Ted Belytschko[2].

2.1 Advantages of XFEM over standard FEM

- Unlike FEM, this method captures the discontinuities in the displacements due to the presence of cracks accurately[3].
- This technique allows the entire crack to be represented independently of the mesh, and so re-meshing is not necessary to model crack growth[3].
- Computational effort is significantly less compared to usual FEM.

3 Partition of Unity Finite Element Method, PU-FEM

A Partition of Unity can be defined as a collection of global functions $f_i(x)$ whose support is contained in a cloud and whose value sum to unity at each point x in the solution domain as see. Equation 1 [1].

$$\sum_{i=1}^{N} f_i(x) = 1, \quad \forall x \in \Omega \tag{1}$$

By choosing an arbitrary function $\psi(x)$ defined on domain Ω^{PU}, the following property can be observed as[3] see. Equation 2.

$$\sum_{i=1}^{N} f_i(x)\psi(x_i) = \psi(x) \tag{2}$$

In the FE approach, the shape functions are usually represented as a PU-FEM. Based on the concept of the PU-FEM, the displacement field $u(x)$ can be discretized over the domain of the problem [3] by taking $f_i(x) \equiv N_i(x)$ as see. Equation 3

$$\mathbf{u}(x) = \sum_{i=1}^{\mathcal{N}} N_i(x)u_i \tag{3}$$

wherein $\mathcal{N}$ is the number of nodal points for each finite element and $N_i(x)$ is the standard FEM shape function.

The incorporation of local enrichment to approximate displacement was originally introduced by Melenk and Babuška (1996) through the PU FEM [1]. The essential feature is based on the multiplication of enrichment functions by nodal shape functions[1].It should be noted that only certain nodes are selected for the enrichment and the FE approximation of enriched domain takes the form see. Equation 4

$$u(x) = \sum_{I \in \mathcal{N}} N_I(x) u_I + \sum_{J \in \mathcal{N}^{enr}} N_J(x) \psi(x) u_J \tag{4}$$

Expanding the above equation leads to see. Equation 5

$$u(x) = \sum_{i=1}^{n} N_i(x) u_i + \sum_{j^{heavy}=1}^{P} N_j(x)[\psi(x)] a_j + \sum_{k^{tip}=1}^{4} N_k(x)[\beta_\alpha(x)] b_{k^\alpha} \tag{5}$$

wherein, $H(x)$ is the Heaviside function, a_j are the additional degree of freedoms associated with Heaviside jump function, $\beta(x)$ represents the branch function and b_k are the additional degree of freedoms associated with branch function and α takes the values from 1 to 4 for each node[3].

4 Extended Finite Element Method (XFEM) - Realization in 2D

Extended finite element method is a partition of unity based method which helps us to model discontinuities that are arbitrarily aligned with the mesh. The main idea behind XFEM is, to enrich the crack containing elements using enrichment functions in standard finite element framework[1].
In this project, Heaviside step function and Asymptotic branch functions have been used as the extrinsic enrichment functions for the displacement approximation[1] . The method entails selection of enriched nodes and elements, a usual FEM procedure such as defining the quadratic shape functions, defining the Iso-parametric elements, converting local coordinate system to global coordinate system using Jacobin matrix, computing, strain-displacement matrix, applying Gauss type integration method, and generating element stiffness matrix. All the necessary steps that are required to solve a BVP using XFEM will be discussed in the following sub-sections.

4.1 Selection of enriched nodes and elements

It has been assumed that the cracks propagate through the elements [2] hence, it is very important to identify properly the elements and nodes that are to be enriched. If an element is completely cut by the crack [12,13,19,18] see Figure 1, then that element should be enriched using Heaviside step function[3].

On the other hand, should the element [14,15,21,20] contains the crack tip, then the corresponding element should be enriched using Asymptotic branch functions[7]. The element [13,14,20,19] which is lying exactly behind the crack tip containing element, will have mixed enrichment[3]. It means that the element will be enriched using both Heaviside and Asymptotic branch functions.

4.2 Selection of Blended elements

In the below illustration see Figure 2, all the elements that are surrounding the crack containing elements will be considered as the blended elements[3]. These elements share their nodes with the enriched elements. The shared nodes will also be enriched accordingly. The blended elements could have 12, 18 and 24 DOFs. This includes 8 normal DOFs and the rest are considered to be the additional DOFs.

4.3 Selection of Mixed Enriched elements

In the below illustration see Figure 3, node no [13,14,20,19] should be mixed enriched because Node numbers [14,20] are sharing their positions with the next element [14,15,21,20] which is Tip enriched and [13,19] are sharing their positions with the element [12,13,19,18] which is completely Heaviside enriched. Hence, node numbers [13,19] will be Heaviside enriched and node numbers [14,20] will be Tip enriched. This type of elements could have 22,28,34 degree of freedoms that leads to a element stiffness matrix of 22x22, 28x28 and 34x34.

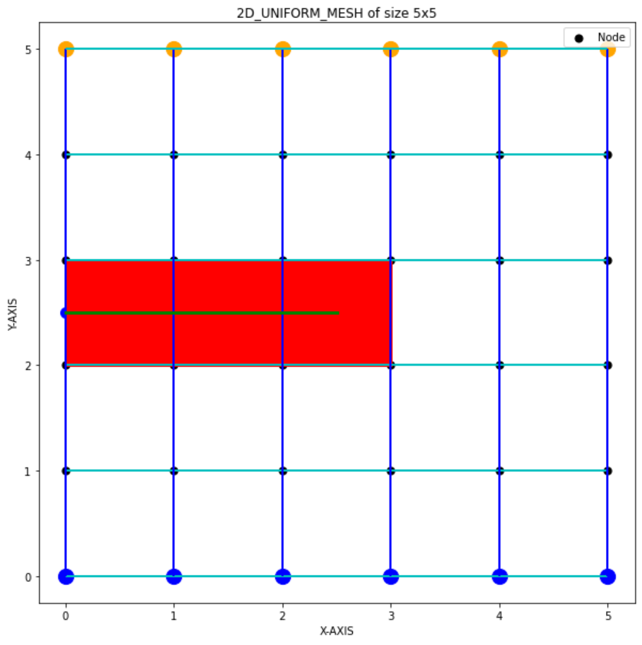


Figure 1: Elements selected for the enrichment

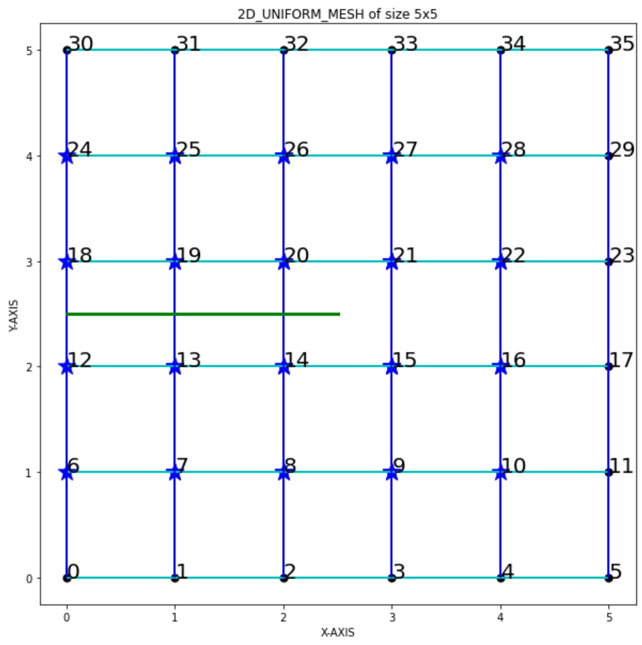


Figure 2: Blended-elements

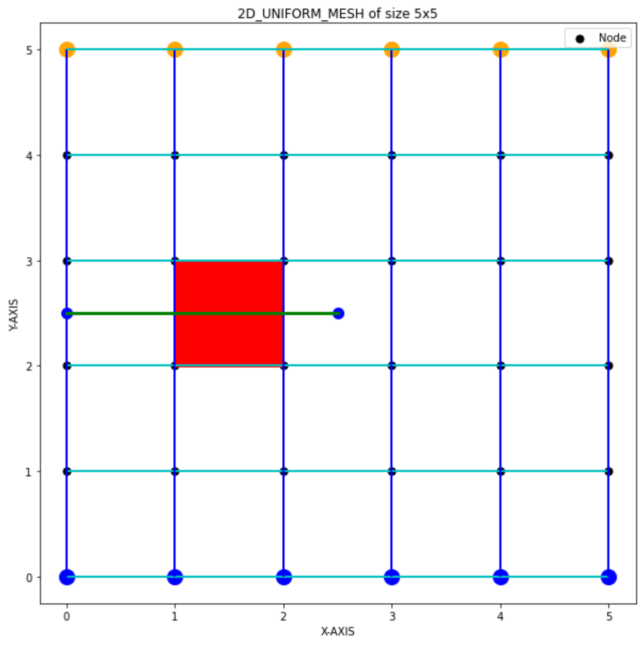


Figure 3: Mixed enriched element

4.4 Heaviside step function

The function usually takes the value of either $+1$ or 0 based on the function evaluated. the illustration is shown below see Figure 4. For an element completely cut by the crack, Heaviside function takes the value of $+1$, if the Gauss point is above the crack segment and -1 if the Gauss point is below the crack segment and will be 0 if it is lying on the crack segment. The step function is easy to calculate based on the sign of the Δ see. Equation 6. One of the easiest ways is the Orientation test[1].

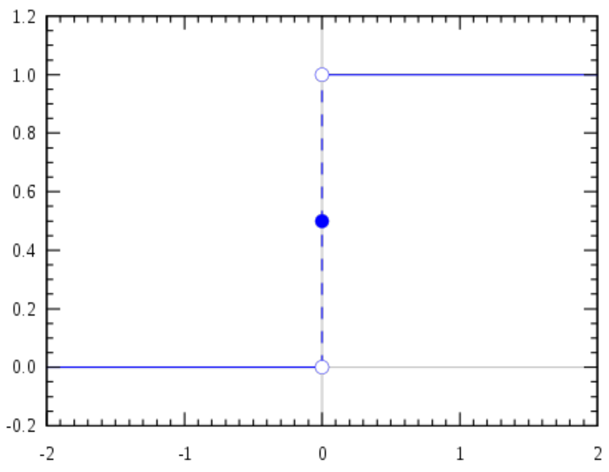


Figure 4: Heaviside step-function
[1]

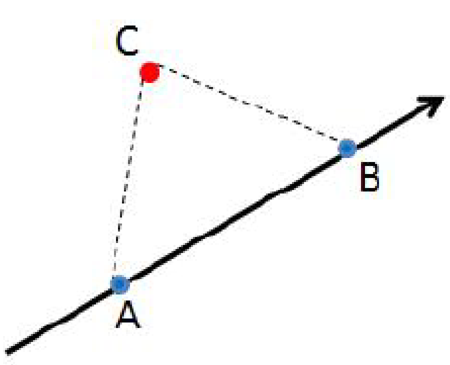
$$H(x)[1] = \begin{cases} +1 & sign|\Delta| > 0 \\ 0 & sign|\Delta| = 0 \\ -1 & sign|\Delta| < 0 \end{cases} \tag{6}$$

$|\Delta|$, represents the determinant value of the matrix Δ which is calculated using orientation test

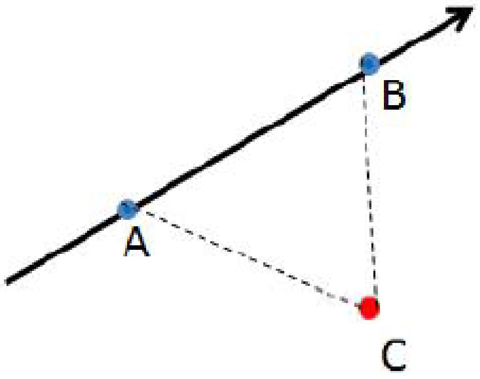
4.5 Orientation test

Orientation test determines whether the point under consideration is above or below the given line segments see Figure 5. The test is performed by evaluating a sign of the determinant. We formulate a triangle using 2 points from the crack segment and 1 query point and evaluating determinant will give the twice the area of a triangle.

It is obvious that, if the a triangle is formed above the crack segment, the sign of the determinant will be positive and should the triangle is formed below the crack segment, the sign of the determinant will be negative, and if the query point falls on the line, the determinant will have a zero value. Mathematically it can be expressed as see. Equation 7



(a) Query point "c" above



(b) Query point "c" below

Figure 5: Orientation test [1]

$$\Delta[1] = \begin{bmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{bmatrix} \tag{7}$$

In the above matrix see. Equation 7, 'a' and 'b' are the coordinates of the crack segment and 'c' is the coordinates of the point under query. This methodology is applied for all the crack segments in a loop and the determinant value which has the least magnitude will be considered and the sign of that value is calculated. -0.1cm

4.6 Heaviside Enrichment

The nodes, that are enriched using Heaviside step function are called Heaviside enriched node. Each node that is Heaviside enriched will have 2 additional DOFs. Should all the nodes of the element are Heaviside enriched then the element will have a 16x16 element stiffness matrix. Below figure see Figure 6 which element has been chosen for the heaviside enrichment.

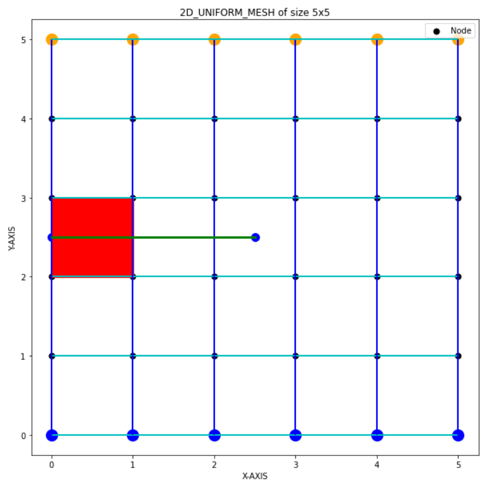


Figure 6: Heaviside enriched element

4.7 Near-tip enrichment functions

This is one of the extrinsic enrichment functions, which is used to enrich an element containing the crack tip. Branch functions are derived from the analytical solution using linear elastic fracture mechanics theory, and they are given as see. Equation 8. The element containing crack tip marked in red is shown in Figure 7.

$$\beta_\alpha[7] = \{\beta_1, \beta_2, \beta_3, \beta_4\} = \sqrt{r}\left[\cos\frac{\theta}{2}, \sin\frac{\theta}{2}, \cos\frac{\theta}{2}\sin\theta, \sin\frac{\theta}{2}\sin\theta\right] \tag{8}$$

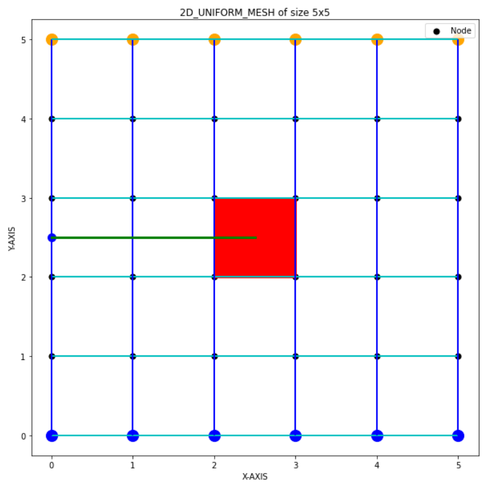


Figure 7: Tip enriched element

It should be mentioned here that r and θ are in polar coordinates w.r.t crack tip. r is the distance measure from the point of the query to the crack tip see. Equation 10 Ex: Node to crack tip, Gauss point to crack tip. It is vital to mention here that the distance measured, r, should be transformed from global coordinate system to crack tip coordinate system.

$$\begin{bmatrix} X_{ctcs} \\ Y_{ctcs} \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} * \begin{bmatrix} x_{global} - X_{tip} \\ y_{global} - Y_{tip} \end{bmatrix} \tag{9}$$

$$r = \sqrt{X_{ctcs}^2 + Y_{ctcs}^2} \qquad \theta = \arctan\left[\frac{Y_{ctcs}}{X_{ctcs}}\right] \tag{10}$$

α is the angle made by the crack w.r.t x-axis, θ should lie in the range of 0 to $\pm\pi$, 'ctcs' is defined as crack tip coordinate system.

The nodes that are enriched using Near-tip enrichment functions are called tip enriched nodes/elements will have 8 additional DOFs these nodes are called Tip enriched nodes. The above figure properly illustrates the scenario. Suppose all the nodes of the element are Tip enriched then the element stiffness matrix will have a size 40x40 for the corresponding element. (8-classical DOFs, $\beta_1 - 8DOFs, \beta_2 - 8DOFs, \beta_3 - 8DOFs, \beta_4 - 8DOFs = 40DOFs)$

4.8 Derivatives of the near-tip enrichment functions

The near-tip enrichment functions are represented in local polar coordinates and hence it has to be transformed to global Cartesian coordinate system. The derivatives of the enrichment functions with regard to global coordinates can be evaluated using the chain rule see. Equation 11 and eq (12) [5]

$$\frac{dF}{dX} = \frac{\partial F}{\partial r} \cdot \frac{\partial r}{\partial X} + \frac{\partial F}{\partial \theta} \cdot \frac{\partial \theta}{\partial X} \tag{11}$$

$$\frac{dF}{dY} = \frac{\partial F}{\partial r} \cdot \frac{\partial r}{\partial Y} + \frac{\partial F}{\partial \theta} \cdot \frac{\partial \theta}{\partial Y} \tag{12}$$

Derivatives of $F\alpha$ (r,θ) with respect to the crack tip polar coordinates (r,θ) is represented as see. Equation 13 to see. Equation 13

$$F_{1,r} = \frac{1}{2\sqrt{r}} \sin\frac{\theta}{2}, \qquad F_{1,\theta} = \frac{\sqrt{r}}{2} \cos\frac{\theta}{2} \tag{13}$$

$$F_{2,r} = \frac{1}{2\sqrt{r}} \cos\frac{\theta}{2}, \qquad F_{2,\theta} = -\frac{\sqrt{r}}{2} \sin\frac{\theta}{2} \tag{14}$$

$$F_{3,r} = \frac{1}{2\sqrt{r}} \sin\frac{\theta}{2} \sin\theta, \quad F_{3,\theta} = \sqrt{r}\left[\frac{1}{2}\cos\frac{\theta}{2}\sin\theta + \sin\frac{\theta}{2}\cos\theta\right] \tag{15}$$

$$F_{4,r} = \frac{1}{2\sqrt{r}} \cos\frac{\theta}{2} \sin\theta, \quad F_{4,\theta} = \sqrt{r}\left[-\frac{1}{2}\sin\frac{\theta}{2}\sin\theta + \cos\frac{\theta}{2}\cos\theta\right] \tag{16}$$

and the derivatives of F_α (r,θ)[5] with respect to the local crack coordinate system (x', y') can then be defined as see. Equation 17 to see. Equation 20

$$F_{1,x'} = -\frac{1}{2\sqrt{r}} \sin\frac{\theta}{2}, \qquad F_{1,y'} = \frac{1}{2\sqrt{r}} \cos\frac{\theta}{2} \tag{17}$$

$$F_{2,x'} = \frac{1}{2\sqrt{r}} \cos\frac{\theta}{2}, \qquad F_{2,y'} = \frac{1}{2\sqrt{r}} \sin\frac{\theta}{2} \tag{18}$$

$$F_{3,x'} = -\frac{1}{2\sqrt{r}} \sin\frac{3\theta}{2} \sin\theta, \quad F_{3,y'} = \frac{1}{2\sqrt{r}}\left[\sin\frac{\theta}{2} + \sin\frac{3\theta}{2}\cos\theta\right] \tag{19}$$

$$F_{4,x'} = -\frac{1}{2\sqrt{r}} \cos\frac{3\theta}{2} \sin\theta, \quad F_{4,y'} = \frac{1}{2\sqrt{r}}\left[\cos\frac{\theta}{2} + \cos\frac{3\theta}{2}\cos\theta\right] \tag{20}$$

Finally, the derivatives of the asymptotic functions F_α [5] in the global coordinate system are obtained by using the below Equation 21 to Equation 24.

$$\frac{\partial r}{\partial X} = \frac{\partial r}{\partial x} \cdot \frac{\partial x}{\partial X} + \frac{\partial r}{\partial y} \cdot \frac{\partial y}{\partial X} \tag{21}$$

$$\frac{\partial r}{\partial Y} = \frac{\partial r}{\partial x} \cdot \frac{\partial x}{\partial Y} + \frac{\partial r}{\partial y} \cdot \frac{\partial y}{\partial X} \tag{22}$$

$$\frac{\partial \theta}{\partial X} = \frac{\partial \theta}{\partial x} \cdot \frac{\partial x}{\partial X} + \frac{\partial \theta}{\partial y} \cdot \frac{\partial y}{\partial X} \tag{23}$$

$$\frac{\partial \theta}{\partial Y} = \frac{\partial \theta}{\partial x} \cdot \frac{\partial x}{\partial Y} + \frac{\partial \theta}{\partial y} \cdot \frac{\partial y}{\partial Y} \tag{24}$$

where the derivatives of r and θ w.r.t to x,y [5] can be written as Equation 25 and Equation 26

$$\frac{\partial r}{\partial x} = \cos\theta \quad , \frac{\partial \theta}{\partial x} = -\frac{\sin\theta}{r} \tag{25}$$

$$\frac{\partial r}{\partial y} = \sin\theta, \quad \frac{\partial \theta}{\partial y} = \frac{\cos\theta}{r} \tag{26}$$

Using the transformation relationship between the global and crack tip coordinates we have Equation 27 and Equation 28

$$\frac{\partial x}{\partial X} = \cos\alpha, \quad \frac{\partial x}{\partial Y} = \sin\alpha \tag{27}$$

$$\frac{\partial y}{\partial X} = -\sin\alpha, \quad \frac{\partial y}{\partial Y} = \cos\alpha \tag{28}$$

5 Formulation of XFEM shape functions and B-matrix in the framework of Finite Element Method

Construction of XFEM shape function N matrix see. Equation 30 and strain-displacement matrix B see. Equation 29 is straight forward. This section comprises of the detailed information regarding the generation of N and B matrices.

$$[B] = \begin{bmatrix} B_{STD} & B_{enr} \end{bmatrix} \tag{29}$$

$$[N] = \begin{bmatrix} N_{STD} & N_{enr} \end{bmatrix} \tag{30}$$

5.1 Shape functions

The shape functions that are used in XFEM are same as the shape functions used in FEM. For a four noded iso-parametric quadrilateral element see Figure 8, the standard FEM bi-linear shape functions [3]associated

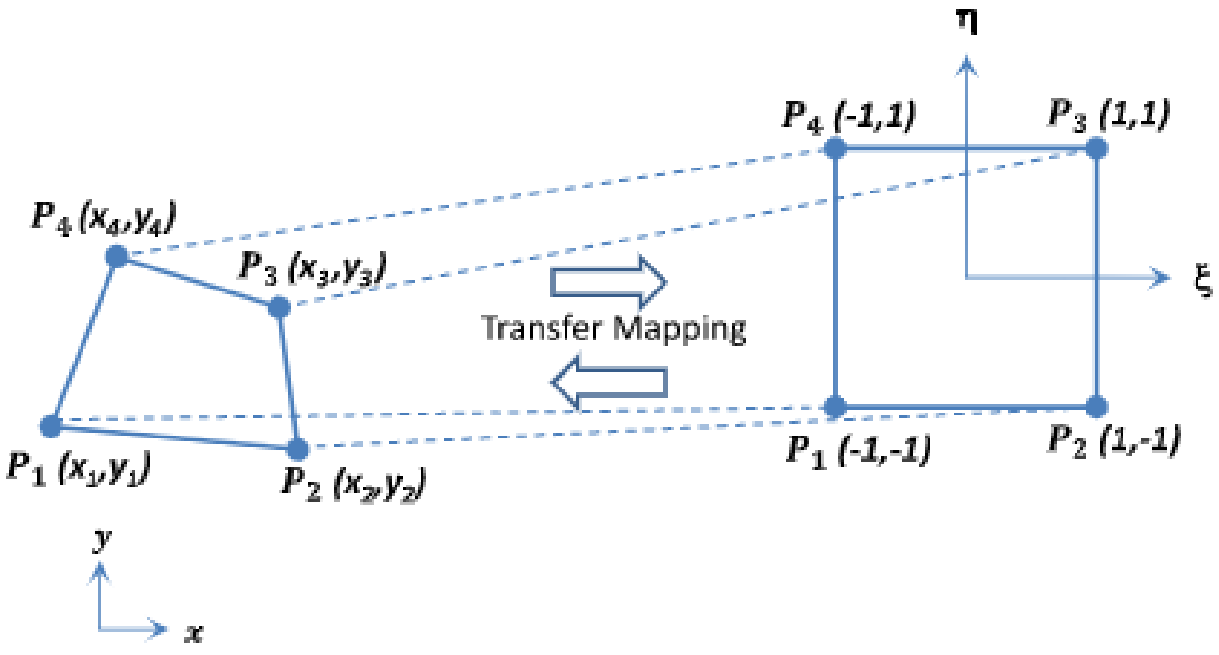


Figure 8: Iso Parametric Element Mapping
[8]

with each node are given as see. Equation 31 to eq (34).

$$N1 = (1 - \xi_1)(1 - \xi_2) \tag{31}$$

$$N2 = (1 + \xi_1)(1 - \xi_2) \tag{32}$$

$$N1 = (1 - \xi_1)(1 - \xi_2) \tag{33}$$

$$N2 = (1 + \xi_1)(1 - \xi_2) \tag{34}$$

The displacement approximation [3] can then be written in the form of see. Equation 35

$$(X) = \begin{bmatrix} N1 & 0 & N2 & 0 & N3 & 0 & N4 & 0 \\ 0 & N1 & 0 & N2 & 0 & N3 & 0 & N4 \end{bmatrix} \cdot \begin{bmatrix} u_{x1} \\ u_{y1} \\ u_{x2} \\ u_{y2} \\ u_{x3} \\ u_{y3} \\ u_{x4} \\ u_{y4} \end{bmatrix} = N_{std}u \tag{35}$$

Where u represents displacements of the given element. Each node has 2 DOFs hence the displacement matrix has a shape of 8x1. The standard FEM shape function matrix is given as see. Equation 36

$$N_{STD} = \begin{bmatrix} N1 & 0 & N2 & 0 & N3 & 0 & N4 & 0 \\ 0 & N1 & 0 & N2 & 0 & N3 & 0 & N4 \end{bmatrix} \tag{36}$$

for a generic enrichment function, g(X), the enriched shape function matrix[3] will be in the form of see. Equation 37

$$N_{ENR} = \begin{bmatrix} [5](N_1 g),x & 0 & (N_2 g),x & 0 & (N_3 g),x & 0 & (N_4 g),x & 0 \\ 0 & (N_1 g),y & 0 & (N_2 g),y & 0 & (N_3 g),y & 0 & (N_4 g),y \end{bmatrix} \tag{37}$$

The shape functions are in local coordinate system and it should be transformed into global coordinate system. This is accomplished by using Jacobian matrix [3] and the steps are as follows

- Take the derivatives of the shape functions w.r.t ξ_1 and ξ_2 see. Equation 38 to eq (41) and see. Equation 42

$$N_{1,\xi_1} = -\frac{1}{4}(1-\xi_2), \quad N_{1,\xi_2} = -\frac{1}{4}(1-\xi_1) \tag{38}$$

$$N_{2,\xi_1} = \frac{1}{4}(1-\xi_2), \quad N_{2,\xi_2} = -\frac{1}{4}(1+\xi_1) \tag{39}$$

$$N_{3,\xi_1} = \frac{1}{4}(1+\xi_2), \quad N_{3,\xi_2} = \frac{1}{4}(1+\xi_1) \tag{40}$$

$$N_{4,\xi_1} = -\frac{1}{4}(1+\xi_2), \quad N_{4,\xi_2} = \frac{1}{4}(1-\xi_1) \tag{41}$$

$$\frac{\partial[N]}{\partial\xi} = \begin{bmatrix} \frac{dN_i}{d\xi_1} \\ \\ \frac{dN_i}{d\xi_2} \end{bmatrix} = 0.25 * \begin{bmatrix} -(1-\xi_2) & 1-\xi_2 & 1+\xi_2 & -(1+\xi_2) \\ -(1-\xi_1) & -(1+\xi_1) & 1+\xi_1 & 1-\xi_1 \end{bmatrix} \tag{42}$$

- Multiply the derivatives with the corresponding nodal coordinates see. Equation 43 this gives the Jacobi matrix of 2x2.

$$J = \frac{\partial[N]}{\partial\xi} \cdot (X^e)^T \tag{43}$$

- Take the inverse of the Jacobi matrix

- Multiply the inverse of the Jacobi matrix with the differentiated shape functions from step 1 generating a 2x4 matrix see. Equation 44.

$$\frac{\partial[N]}{\partial x} = J^{-1} \cdot \frac{\partial[N]}{\partial\xi} \tag{44}$$

- The elements in this matrix are arranged in a specified manner to get B-matrix see. Equation 45.

5.2 B-matrix

This is also called as a Strain-Displacement matrix which gives a relation between strains and displacements see. Equation 45[3].

$$B_{std} = \begin{bmatrix} N_{1,x} & 0 & N_{2,x} & 0 & N_{3,x} & 0 & N_{4,x} & 0 \\ 0 & N_{1,y} & 0 & N_{2,y} & 0 & N_{3,y} & 0 & N_{4,y} \\ N_{1,y} & N_{1,x} & N_{2,y} & N_{2,x} & N_{3,y} & N_{3,x} & N_{4,y} & N_{4,x} \end{bmatrix} \tag{45}$$

Enriched B-matrix [3] is given by see. Equation 46

$$B_{enr} = \begin{bmatrix} (N_1g),x & 0 & (N_2g),x & 0 & (N_3g),x & 0 & (N_4g),x & 0 \\ 0 & (N_1g),y & 0 & (N_2g),y & 0 & (N_3g),y & 0 & (N_4g),y \\ (N_4g),y & (N_1g),x & (N_4g),y & (N_4g),x & (N_4g),y & (N_4g),x & (N_4g),y & (N_4g),x \end{bmatrix} \tag{46}$$

$g(x)$ could be either Heaviside step function or Near Tip function. The below Table 1 gives the information on the additional degrees of freedom generated due to the enrichment

g(x)	**Enrichment type**	**Degrees of freedom**
H(X)	Heaviside function or Heaviside step function	2 DOFs per node
$F^4(r, \theta)$	near-tip enrichment function	8 DOFs per node

Table 1: Table showing the additional DOFs for an enriched node

If the node is Heaviside enriched, the B-matrix[3] is given by see. Equation 47

$$B_{Heavy} = \begin{bmatrix} (N_iH(x))_x & 0 \\ 0 & (N_iH(x))_y \\ (N_iH(x))_y & (N_iH(x))_x \end{bmatrix} \quad i = 1,2,3,4 \tag{47}$$

The derivative of Heaviside function is the Dirac delta function[3], that is see. Equation 48

$$(N_iH(x))_x = N_{i,x}H_{,x}(x) = \delta \tag{48}$$

If the node is tip enriched, the B-matrix [3] is given by see. Equation 49

$$B_{tip} = \begin{bmatrix} (N_i F_\alpha),x & 0 \\ 0 & (N_i F_\alpha),y \\ (N_i F_\alpha),y & (N_i F_\alpha),x \end{bmatrix} \quad \alpha = 1,2,3,4 \tag{49}$$

The derivative of near-tip enrichment function[3] is given by see. Equation 50.

$$(N_i F_\alpha),x = F_\alpha N_{i,x} + F_{\alpha,x} N_i \tag{50}$$

$$(N_i F_\alpha),y = F_\alpha N_{i,y} + F_{\alpha,y} N_i \tag{51}$$

when $i = 1, \alpha = 1,2,3,4$

$$(N_1 F_1),x = F_1 N_{1,x} + F_{1,x} N_1 \tag{52}$$

$$(N_1 F_2),x = F_2 N_{1,x} + F_{2,x} N_1 \tag{53}$$

$$(N_1 F_3),x = F_3 N_{1,x} + F_{3,x} N_1 \tag{54}$$

$$(N_1 F_4),x = F_4 N_{1,x} + F_{4,x} N_1 \tag{55}$$

Then the tip enriched B-matrix for 1 node is represented as see. Equation 56,

$$B_{tip} = \begin{bmatrix} F_1 N_{1,x} + F_{1,x} N_1 & 0 & \dots & F_4 N_{1,x} + F_{4,x} N_1 & 0 \\ 0 & F_1 N_{1,y} + F_{1,y} N_1 & \dots & 0 & F_4 N_{1,y} + F_{4,y} N_1 \\ F_1 N_{1,y} + F_{1,y} N_1 & F_1 N_{1,x} + F_{1,x} & \dots & F_4 N_{1,y} + F_{4,y} N_1 & F_4 N_{1,x} + F_{4,x} N_1 \end{bmatrix} \quad \alpha = 1,2,3,4 \tag{56}$$

the same has to be implemented for the other 3 nodes if all 4 nodes in the element are tip enriched.

5.3 Numerical Integration

In the classical FEM, the standard shape functions are of the polynomial order and the Gauss integration rule can be used to evaluate the integral of stiffness matrix. However, in the X-FEM the enriched shape functions are obtained in terms of non-polynomial order. Moreover, the enrichment functions may not be smooth over an enriched element due to presence of the weak or strong discontinuity inside the element. Hence, the standard Gauss quadrature rule cannot be used if an element is crossed over by a crack, and necessary modifications are necessary for numerical integration over an enriched element[3].

The suggested approach is based on the increase of the number of Gauss integration points[3], as shown in the below Figure Figure 9, however, this may also result in a substantial loss of accuracy. In order to overcome these difficulties, the enriched element will be divided into n-number of sub-polygons as shown in Figure, and the Gauss integration rule is performed over each sub-polygons. The sub-polygons do not produce additional DOFs[3]. In the end all the K-matrices of the sub polygons will be summed up to get a K-matrix of the corresponding element.

Consider an element cut by an interface into two distinct parts, Ω+ and Ω−; the integration of stiffness matrix over a domain (element) Ω can be performed using see. Equation 57

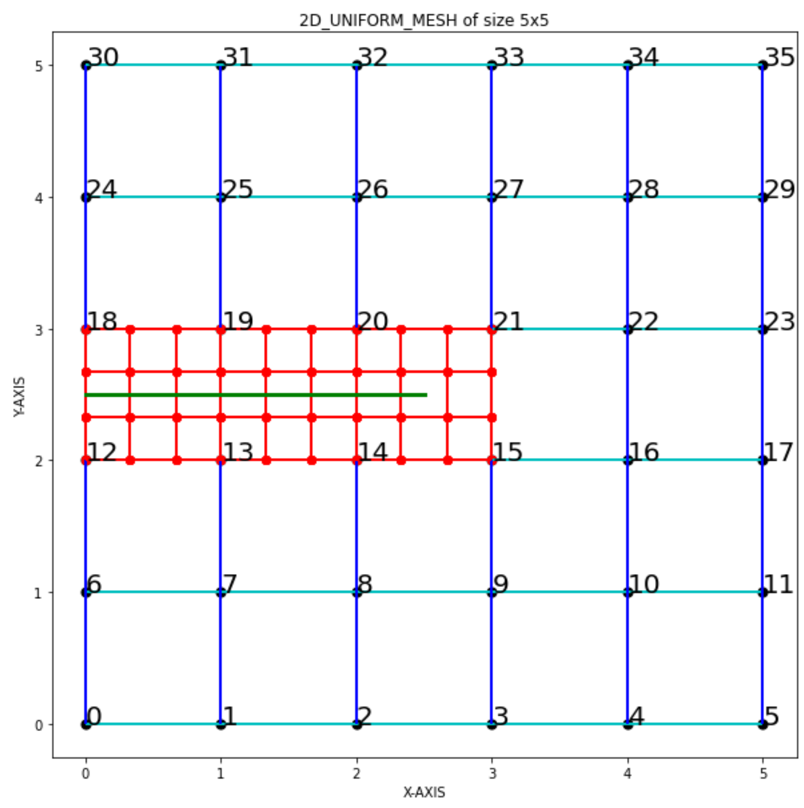


Figure 9: 9 Sub-polygons per element

$$K_{ij}^{\alpha\beta} = \int_{\Omega} B^T_{i(\xi_1,\xi_2)} * D * B_{j(\xi_1,\xi_2)} * |J| d\xi_1 d\xi_2$$

$$= \int_{\Omega+} B^T_{i(\xi_1,\xi_2)} * D * B_{j(\xi_1,\xi_2)} * |J| d\xi_1 d\xi_2 + \int_{\Omega-} B^T_{i(\xi_1,\xi_2)} * D * B_{j(\xi_1,\xi_2)} * |J| d\xi_1 d\xi_2$$

$$= \sum_{l=1}^{\mathcal{N}^{SUB+}} \left(\sum_{k=1}^{\mathcal{N}^{GP+}} B^T_{i(\xi_{1k},\xi_{2k})} * D * B_{j(\xi_{1k},\xi_{2k})} * w_k \right)_l + \sum_{l=1}^{\mathcal{N}^{SUB-}} \left(\sum_{k=1}^{\mathcal{N}^{GP-}} B^T_{i(\xi_{1k},\xi_{2k})} * D * B_{j(\xi_{1k},\xi_{2k})} * w_k \right)_l \quad (57)$$

where in $\mathcal{N}^{SUB+}$ and $\mathcal{N}^{SUB-}$ are the number of sub-polygons in $\Omega+$ and $\Omega-$, and $\mathcal{N}^{GP+}$ and $\mathcal{N}^{GP-}$ are the number of Gauss points at each sub-polygon in $\Omega+$ and $\Omega-$, respectively.In this relation, w_k is the weight of quadrature point. Therefore, Gauss quadrature can be employed, which allows integration of polynomials up to a certain order in the given element.

5.4 Element connectivity matrix

The matrix which is used to assemble all the element stiffness matrix. The matrix consists of only 0s and 1s. The size of the connectivity matrix is decided based on the number of DOFs available in the geometry. The columns of the matrix is the maximum DOFs available in the geometry and the rows of the matrix is the maximum DOFs available in the single element. As mentioned in the previous section, the additional DOFs will be placed in the last columns and rows after allotting rows and columns for normal DOFs. The equation for generating an assignment matrix is given by see. Equation 58

$$A_{matrix} = [Max\ DOFs\ availabe\ in\ element] \ * \ [Max\ DOFs\ availabe\ in\ the\ geometry] \quad (58)$$

5.5 Element stiffness matrix(K-matrix)

The element stiffness matrix for classical element is given by see. Equation 59 and see. Equation 60[3]

$$K = \int_{-1}^{1} \int_{-1}^{1} B^T_{\xi_1,\xi_2} * D * B_{\xi_1,\xi_2} * |J| d\xi_1 d\xi_2 \quad (59)$$

Wherein

B^T = Strain displacement matrix

D = Plane stress relation/plane strain relation

J = Jacobian matrix

The BVP is solved using

$$[K][u] = [F] \tag{60}$$

Wherein

K = K_{global}, is called the global stiffness matrix

u = Displacement matrix

F = Force vector

5.6 Nodal Displacements

After solving BVP, a 1D displacement vector is obtained. This vector contains both classical and enriched displacements. It should be noted that if there a node is enriched, then displacement of that node will be summed with the enriched displacements and multiplied with the enrichment function value. Using the displacement approximation equation, corresponding nodal displacements are calculated. The relation handles both classical and enriched displacements see. Equation 61.

$$u(x) = \sum_{i=1}^{n} N_i(x)u_i + \sum_{j^{heavy}=1}^{P} N_j(x)[\psi(x)]a_j + \sum_{k^{tip}=1}^{4} N_k(x)[\beta_\alpha(x)]b_{k^\alpha} \tag{61}$$

The displacements that are calculated using XFEM should be treated the same way as the stresses. But before transforming, one should calculate the gradient of the XFEM displacements [1] and it is given by see. Equation 62

$$\begin{bmatrix} u_{x(XFEM)} & u_{y(XFEM)} \\ v_{x(XFEM)} & v_{y(XFEM)} \end{bmatrix} = B_{std} = \begin{bmatrix} N_{1,x} & 0 & N_{2,x} & 0 & N_{3,x} & 0 & N_{4,x} & 0 \\ 0 & N_{1,y} & 0 & N_{2,y} & 0 & N_{3,y} & 0 & N_{4,y} \end{bmatrix} * \begin{bmatrix} u_{x1} \\ u_{y1} \\ u_{x2} \\ u_{y2} \\ u_{x3} \\ u_{y3} \\ u_{x4} \\ u_{y4} \end{bmatrix} \tag{62}$$

Transformation of displacement gradients to crack tip coordinate system see. Equation 63

$$\begin{bmatrix} U_{X(CTCS)} & U_{Y(CTCS)} \\ V_{X(CTCS)} & V_{Y(CTCS)} \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} * \begin{bmatrix} u_{x(XFEM)} & u_{y(XFEM)} \\ v_{x(XFEM)} & v_{y(XFEM)} \end{bmatrix} * \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \tag{63}$$

5.7 Calculation of Stresses and Strains

After the computation of displacements, strains must be calculated using strain-displacement matrix (B-matrix) and it is given by see. Equation 64 and see. Equation 65

$$\epsilon = [B][u] \tag{64}$$

wherein ϵ is the strain and B is B-matrix and u is the nodal displacements.

$$\begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} N_{1,x} & 0 & N_{2,x} & 0 & N_{3,x} & 0 & N_{4,x} & 0 \\ 0 & N_{1,y} & 0 & N_{2,y} & 0 & N_{3,y} & 0 & N_{4,y} \\ N_{1,y} & N_{1,x} & N_{2,y} & N_{2,x} & N_{3,y} & N_{3,x} & N_{4,y} & N_{4,x} \end{bmatrix} * \begin{bmatrix} u_{x1} \\ u_{y1} \\ u_{x2} \\ u_{y2} \\ u_{x3} \\ u_{y3} \\ u_{x4} \\ u_{y4} \end{bmatrix} \tag{65}$$

Stresses are calculated using the see. Equation 66 and plane stress relation see. Equation 67

$$\sigma = [D][\epsilon] \tag{66}$$

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} * \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} \tag{67}$$

The stresses, that are calculated using XFEM should be transformed from global coordinate system to crack tip coordinate system using appropriate rotation matrix [1] and the equation is given by see. Equation 68

$$\begin{bmatrix} \sigma_{XX(CTCS)} & \sigma_{XY(CTCS)} \\ \sigma_{XY(CTCS)} & \sigma_{YY(CTCS)} \end{bmatrix} = e \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} * \begin{bmatrix} \sigma_{xx(XFEM)} & \sigma_{xy(XFEM)} \\ \sigma_{yx(XFEM)} & \sigma_{yy(XFEM)} \end{bmatrix} * \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \tag{68}$$

5.8 Displacement Boundary conditions

This approach is adapted when displacements boundary conditions are applied instead of traction boundary conditions.[1]. Procedure has been described below in detail. The approach splits the displacement matrix and K-matrix into 2 parts one with known displacements U_D and another w with unknown displacements U_f Equation 69

$$A = \begin{pmatrix} A_{DD} & A_{DD} \\ A_{fD} & A_{ff} \end{pmatrix} \quad \text{and} \quad f = \begin{pmatrix} f_D \\ f_f \end{pmatrix} \tag{69}$$

After splitting the K-matrix and displacement vector, we have a system of linear equation Equation 70

$$\begin{pmatrix} I & 0 \\ A_{fD} & A_{ff} \end{pmatrix} \begin{pmatrix} u_D \\ u_f \end{pmatrix} = \begin{pmatrix} u_D \\ f_f \end{pmatrix} \tag{70}$$

The above equation Equation 70 is used to compute the displacements.

6 Linear Elastic Fracture mechanics

6.1 Introduction

Strength of the materials were computed in the past based on two possible theories [Griffith 1921]. A material is said to fracture if maximum tensile stress or maximum extension in a body exceeds a certain threshold value[4]. Hence the strength of the material is basically considered to be an intrinsic property.
One of earliest recorded incidents of brittle fracture failure was the Montrose bridges 1830 [Erdogan 2000]. There have been many incidents due to fracture failure after that e.g the event of Tay Rail Bridge failure in 1879 [4]. These incidents led the scientists and engineers to exploit the Fracture mechanics domain. It is said that Griffith's and Irwin's work has led the foundations for a new engineering branch "Engineering Fracture Mechanics" to flourish, and soon after that Fracture mechanics evolved as an important branch in the engineering realm. A very good review on fracture mechanics can be found in Erdogan [2000]. More details on engineering fracture mechanics can also be found in [4]

6.2 Modes of Failure

Failure is defined as the rupture of the sample under the applied load. There are three modes of failure, namely Mode I, Mode II, and Mode III [4].

- Mode I: is the opening type Figure 10 wherein a tensile load is applied normal to the crack surfaces and crack opens perpendicular to the crack plane i.e. the crack propagation angle is 0.

- Mode II: A shear load is applied parallel to the crack surfaces Figure 11 and the crack is allowed to propagate. The crack propagation angle varies from +70 degrees to -70 degrees. The crack faces are found to be sliding in the direction of applied load.
- Mode III: a shear stress is applied perpendicular to the plane of the crack Figure 12. This is also called as Out-of-plane tearing mode.

The three modes of failures are shown schematically in the below figures.

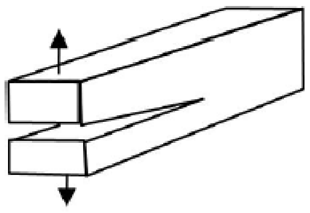


Figure 10: Mode-I Type failure [4]

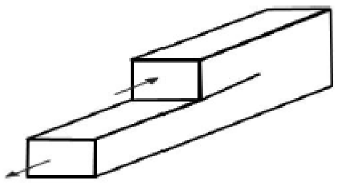


Figure 11: Mode-II Type failure [4]

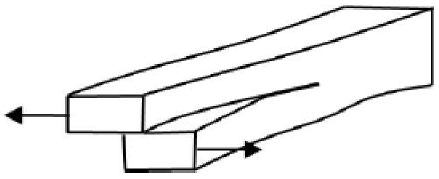


Figure 12: Mode-III Type failure [4]

7 Displacement and Stress Fields at the Crack Tip Area

The importance of stress singularity at the crack tip area was presented by Westergaard (1939) and Williams (1957) by evaluation of the displacement and stress fields around the crack tip area Figure 13. Williams (1957) used the Airy stress function to capture the singularity at the crack tip area using a polar coordinate system [3] (r, θ) as given in see. Equation 71

$$\Phi = r^{\lambda+1}(c_1 \sin(\lambda+1)\hat{\theta} + c_2 \cos(\lambda+1)\hat{\theta} + c_3 \sin(\lambda-1)\hat{\theta} + c_4 \sin(\lambda-1)\hat{\theta}) \tag{71}$$

wherein c_i are the coefficients and $\hat{\theta}$ is depicted in the below Figure

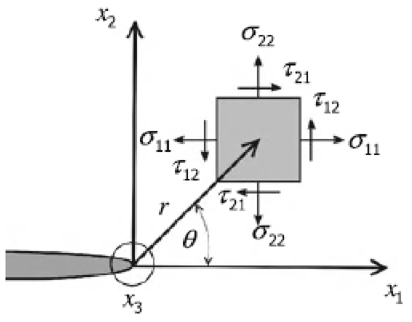


Figure 13: Stresses at the crack tip in Cartesian coordinates [4]

7.1 Auxiliary displacements for mode-I

Substituting the Airy stress function EQUATION in the equilibrium equation of the system, that is, $\bigtriangledown^2 \bigtriangledown^2 \Phi = 0$ [3] in the absence of body forces, applying the traction-free boundary conditions at the crack faces. It should be noted here that the displacement field is a function of crack tip polar coordinate system and we are required to find the spatial derivatives according to the local crack tip Cartesian coordinate system [6]. This will be evaluated as see. Equation 72 to see. Equation 74

$$U_x = \frac{K_I(1+\nu)}{E}\sqrt{\frac{r}{2\pi}}\cos\frac{\theta}{2}\left[k - 1 + 2\sin^2\frac{\theta}{2}\right] \tag{72}$$

$$U_y = \frac{K_I(1+\nu)}{E}\sqrt{\frac{r}{2\pi}}\sin\frac{\theta}{2}\left[k + 1 - 2\cos^2\frac{\theta}{2}\right] \tag{73}$$

$$U_z = 0 \tag{74}$$

Derivatives of the displacement U_x and U_y w.r.t to x and y [6] is given in see. Equation 75 to eq (78)

$$\frac{dU_x}{dX} = \frac{dU_x}{dr} * \frac{dr}{dX} + \frac{dU_x}{d\theta} * \frac{d\theta}{dX} \tag{75}$$

$$\frac{dU_x}{dY} = \frac{dU_x}{dr} * \frac{dr}{dY} + \frac{dU_x}{d\theta} * \frac{d\theta}{dY} \tag{76}$$

$$\frac{dU_y}{dX} = \frac{dU_y}{dr} * \frac{dr}{dX} + \frac{dU_y}{d\theta} * \frac{d\theta}{dX} \tag{77}$$

$$\frac{dU_y}{dY} = \frac{dU_y}{dr} * \frac{dr}{dY} + \frac{dU_y}{d\theta} * \frac{d\theta}{dY} \tag{78}$$

Derivatives of the displacement U_x and U_y w.r.t to r and θ [6]is given by see. Equation 79 to see. Equation 82

$$\frac{dU_x}{dr} = \frac{K_I(1+\nu)}{2D} \frac{1}{\sqrt{2\pi r}} \cos\frac{\theta}{2}(k - \cos\theta) \tag{79}$$

$$\frac{dU_x}{d\theta} = \frac{K_I(1+\nu)}{D} \sqrt{\frac{r}{2\pi}} \left[-0.5 \sin\frac{\theta}{2}(k - \cos\theta) + \sin\theta\cos\frac{\theta}{2}\right] \tag{80}$$

$$\frac{dU_y}{dr} = \frac{K_I(1+\nu)}{2D} \frac{1}{\sqrt{2\pi r}} \sin\frac{\theta}{2}(k - \cos\theta) \tag{81}$$

$$\frac{dU_x}{d\theta} = \frac{K_I(1+\nu)}{D} \sqrt{\frac{r}{2\pi}} \left[0.5 \cos\frac{\theta}{2}(k - \cos\theta) + \sin\theta\sin\frac{\theta}{2}\right] \tag{82}$$

Derivatives r and θ w.r.t to x and y [3] is given by see. Equation 83 and eq(84)

$$\frac{dr}{dX} = \cos\theta, \qquad \frac{dr}{dY} = \sin\theta \tag{83}$$

$$\frac{d\theta}{dX} = -\sin\frac{\theta}{r}, \qquad \frac{d\theta}{dY} = \cos\frac{\theta}{r} \tag{84}$$

7.2 Auxiliary stresses for mode-I

The Auxiliary stresses in case of mode I is given by Equation 85 to Equation 88 [3]

$$\sigma_x = \frac{K_I}{\sqrt{2\pi r}} \cos\frac{\theta}{2}\left[1 - \sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right] \tag{85}$$

$$\sigma_y = \frac{K_I}{\sqrt{2\pi r}} \cos\frac{\theta}{2}\left[1 + \sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right] \tag{86}$$

$$\tau_{xy} = \frac{K_I}{\sqrt{2\pi r}} \sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\frac{3\theta}{2} \tag{87}$$

$$\sigma_z = \begin{cases} \nu(\sigma_x + \sigma_y) & \text{Plane strain} \\ 0 & \text{Plane stress} \end{cases} \tag{88}$$

7.3 Auxiliary displacements for mode-II

The displacements for mode-II is given by Equation 89 to Equation 91 [3]

$$U_x = \frac{K_{II}(1+\nu)}{E}\sqrt{\frac{r}{2\pi}}\sin\frac{\theta}{2}\left[k + 1 + 2\cos^2\frac{\theta}{2}\right] \tag{89}$$

$$U_y = \frac{K_{II}(1+\nu)}{E}\sqrt{\frac{r}{2\pi}}\cos\frac{\theta}{2}\left[k - 1 - 2\sin^2\frac{\theta}{2}\right] \tag{90}$$

$$U_z = 0 \tag{91}$$

Derivatives of the displacement U_x and U_y w.r.t to r and θ [6] is given in Equation 92 to Equation 95

$$\frac{dU_x}{dr} = \frac{K_{II}(1+\nu)}{2D}\frac{1}{\sqrt{2\pi r}}\cos\frac{\theta}{2}(k + 2 + \cos\theta) \tag{92}$$

$$\frac{dU_x}{d\theta} = \frac{K_{II}(1+\nu)}{D}\sqrt{\frac{r}{2\pi}}\left[0.5\cos\frac{\theta}{2}(k + 2 + \cos\theta) - \sin\theta\sin\frac{\theta}{2}\right] \tag{93}$$

$$\frac{dU_y}{dr} = \frac{K_{II}(1+\nu)}{2D}\frac{1}{\sqrt{2\pi r}}\cos\frac{\theta}{2}(k - 2 + \cos\theta) \tag{94}$$

$$\frac{dU_x}{d\theta} = -\frac{K_{II}(1+\nu)}{D}\sqrt{\frac{r}{2\pi}}\left[-0.5\sin\frac{\theta}{2}(k - 2 + \cos\theta) + -\sin\theta\cos\frac{\theta}{2}\right] \tag{95}$$

7.4 Auxiliary stresses for mode-II

The Auxiliary stresses in case of mode II is given by Equation 96 to Equation 99 [3]

$$\sigma_x[1] = -\frac{K_{II}}{\sqrt{2\pi r}} \sin\frac{\theta}{2}\left[2 + \cos\frac{\theta}{2}\cos\frac{3\theta}{2}\right] \tag{96}$$

$$\sigma_y[1] = \frac{K_{II}}{\sqrt{2\pi r}} \sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\frac{3\theta}{2} \tag{97}$$

$$\tau_{xy}[1] = \frac{K_{II}}{\sqrt{2\pi r}} \cos\frac{\theta}{2}\left[1 - \sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right] \tag{98}$$

$$\sigma_z[1] = \begin{cases} \nu(\sigma_x + \sigma_y) & \text{Plane strain} \\ 0 & \text{Plane stress} \end{cases} \tag{99}$$

Wherein "k" in the above equations is called Kolosov constant and it is defined as $k = (3-\nu)/(1+\nu)$ for plane stress and $k = 3-4\nu$ for plane strain problems and ν is the Poisson ratio. D is Young's modulus. In the above relations, K_I, and K_{II} are the SIFs for mode-I and mode-II respectively and they are given by Equation 100 and Equation 101.

$$K_I = \sigma_y\sqrt{2 * \pi r} \quad MPa\sqrt{m} \tag{100}$$

$$K_{II} = \sigma_{xy}\sqrt{2 * \pi r} \quad MPa\sqrt{m} \tag{101}$$

wherein

$\boldsymbol{\sigma_x}$ = Tensile Stress in MPa

$\boldsymbol{\sigma_{xy}}$ = Shear Stress in MPa

r = crack length in m

8 Stress Intensity Factors

In the linear elastic fracture mechanics (LEFM), the stress, strain, and displacement fields can be determined by employing the concept of the SIFs near the crack tip region. It is therefore important to accurately evaluate the SIFs for the FE analysis of LEFM.

There are many computational algorithms available to evaluate the SIFs. The approaches can be categorized into two groups; the "direct" approach and the "energy" approach. The direct approach relates the SIFs with the FEM results directly, while the energy approach is based on the computation of energy release rate. In general, the energy approaches are more accurate than the direct procedures. However, the direct approaches are more popular and are usually used to verify the results of energy approaches, since their expressions are simple. The most popular energy approach is the J-integral technique[3]. In the following section the J-integral method is discussed in detail

8.1 J-integral

In Fracture mechanics computations, SIFs are the most important fracture parameters used for the determination of mixed-mode near tip stress, strain, and displacement fields. In order to evaluate the SIFs, the area J–integral is defined in relation Equation 102 and see. Equation 103, which can be computed over an area of the FE mesh refer to the figure below Figure 14. In XFEM modeling, the area for the integration is calculated by assuming a virtual circle with a specific radius around the crack tip, and the integration is performed over the elements that lie inside the circle [4], as shown in Figure 7.16.

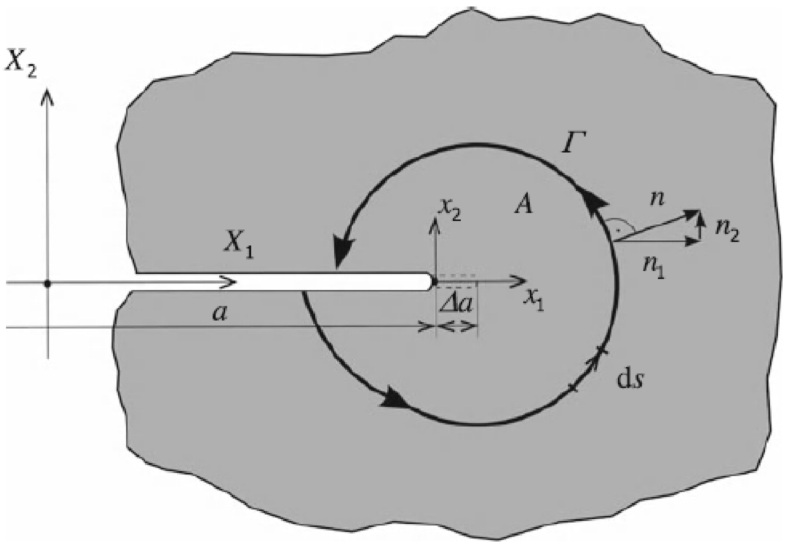


Figure 14: Elements inside the area are integrated[4]

Basically, J-integral is calculated by adding the integrals computed for a sample without a crack (Auxiliary state / state (2)) and the same sample with crack (current configuration/ state(1)). For this purpose, BVP is solved using XFEM to obtain to obtain the displacement, strain, and stress fields of state (1) that is, $u_i^{(1)}$, $\epsilon_{ij}^{(1)}$, and $\sigma_{ij}^{(1)}$ and these values are then transferred from the global to local crack tip coordinate system (x_1, x_2) by using an appropriate transformation.

$$J = \int_A \left(\sigma_{ij} \frac{\partial u_i}{\partial x_1} - \mathcal{W} \delta_{ij} \right) \frac{\partial q}{\partial x_j} dA \tag{102}$$

$$I^{(1,2)} = \int_A \left(-\mathcal{W}^{(1,2)} \delta_{1j} + \sigma_{ij}^{(1)} \frac{\partial u_i^{(2)}}{\partial x_1} + \sigma_{ij}^{(2)} \frac{\partial u_i^{(1)}}{\partial x_1} \right) \frac{\partial q}{\partial x_j} dA \tag{103}$$

Wherein $\mathcal{W}^{(1,2)}$ is the interaction strain energy [4] defined by see. Equation 104

$$\mathcal{W}^{(1,2)} = \sigma_{ij}^{(1)} \epsilon_{ij}^{(2)} = \sigma_{ij}^{(2)} \epsilon_{ij}^{(1)} \tag{104}$$

The super scripts (2), and (1) implicates auxiliary state(material without crack) and current state(after the introduction of the crack) respectively. The distribution of weighting function q in the above relation can be obtained for an element using the standard FE interpolation [4] as see. Equation 105

$$q = \sum_{I=1}^{\mathcal{N}^{elem}} N_I(x) q_I \tag{105}$$

where $\mathcal{N}^{elem}$ is the number of nodes of an element and q_I are the nodal values of q. The derivation of weighting function can be obtained as see. Equation 106

$$\frac{\partial q}{\partial x} = \sum_{I=1}^{\mathcal{N}^{elem}} \frac{\partial N_I}{\partial x} q_I \tag{106}$$

After calculating the derivative of the weight function see. Equation 106, $\frac{\partial q}{\partial x}$ is converted to local crack tip coordinate system using appropriate transformation see. Equation 107.

$$\left[\frac{\partial q}{\partial X} \right] = [R] * \left[\frac{\partial q}{\partial x} \right] \tag{107}$$

where R in the above equation is called rotation matrix and it is defined as see. Equation 108

$$\begin{bmatrix} \frac{\partial q}{\partial X} \\ \frac{\partial q}{\partial Y} \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} * \begin{bmatrix} \frac{\partial q}{\partial x} \\ \frac{\partial q}{\partial y} \end{bmatrix} \tag{108}$$

Equation 102 is a part of the J-integral and its is called as interaction integral and the terms in the equation can be expanded as follows

The first term in the Equation 102, $\mathcal{W}^{(1,2)}$, is called the strain energy function and it can be expanded as Equation 109

$$\mathcal{W}^{(1,2)} = ((\sigma_{11}^{(1)} * \epsilon_{11}^{(2)}) + (\sigma_{12}^{(1)} * \epsilon_{12}^{(2)}) + (\sigma_{21}^{(1)} * \epsilon_{21}^{(2)}) + (\sigma_{22}^{(1)} * \epsilon_{22}^{(2)}))\frac{\partial Q}{\partial x_1} \tag{109}$$

The second term in the Equation 102, $\left[\sigma_{ij}^{(1)} \frac{\partial u_i^{(2)}}{\partial x_1}\right]$, can be expanded as Equation 110

$$\sigma_{ij}^{(1)} \frac{\partial u_i^{(2)}}{\partial x_1} = (\sigma_{11}^{(1)} * \frac{\partial u_i^{(2)}}{\partial x_1} * \frac{\partial Q}{\partial x_1}) + (\sigma_{12}^{(1)} * \frac{\partial u_i^{(2)}}{\partial x_2} * \frac{\partial Q}{\partial x_2}) + (\sigma_{21}^{(1)} * \frac{\partial u_i^{(2)}}{\partial x_1} * \frac{\partial Q}{\partial x_1}) + (\sigma_{22}^{(1)} * \frac{\partial u_i^{(2)}}{\partial x_2} * \frac{\partial Q}{\partial x_2}) \tag{110}$$

The third term in the Equation 102, $\left[\sigma_{ij}^{(2)} \frac{\partial u_i^{(1)}}{\partial x_1}\right]$, can be expanded as Equation 111

$$\sigma_{ij}^{(2)} \frac{\partial u_i^{(1)}}{\partial x_1} = (\sigma_{11}^{(2)} * \frac{\partial u_i^{(1)}}{\partial x_1} * \frac{\partial Q}{\partial x_1}) + (\sigma_{12}^{(2)} * \frac{\partial u_i^{(1)}}{\partial x_2} * \frac{\partial Q}{\partial x_2}) + (\sigma_{21}^{(2)} * \frac{\partial u_i^{(1)}}{\partial x_1} * \frac{\partial Q}{\partial x_1}) + (\sigma_{22}^{(2)} * \frac{\partial u_i^{(1)}}{\partial x_2} * \frac{\partial Q}{\partial x_2}) \tag{111}$$

8.2 The Maximum Energy Release Rate

It is the amount of total stored energy being released per area of crack growth. After evaluating Equation 103, K_I and K_{II} can be conputed using the below equation Equation 112

$$K_I, \quad K_{II} = \frac{I^(1,2) * D}{2} \tag{112}$$

Then the computed SIFs are used to compute the energy release rate Equation 113.

$$G = \frac{K_I^2 + K_{II}^2}{E} \tag{113}$$

Wherein E = Young's modulus for a plane stress case, G is expressed in $\frac{N}{m}$.

Function Body of the Interaction integral

```
def Interaction_integral(Nodes, displacements, stress, strains, set4Gauss, c_2,
GaussPoint_1to4, l_x, l_y, D_plane_stress, alpha, CL, A, force, D, scale,
increment)
The function computes the SIFs using domain integral technique
Parameters:
-----------
Nodes :  List of 4 nodal coordinates
displacements :  List of Element displacements (8 per element)
stress :  List of 3 stress values
strains:  List of 3 strain values
set4Gauss:  Gauss coordinates in global system
c_2:  Crack tip coordinates
GaussPoint_1to4 :  4 Gauss coordinates
l_x, l_y:  Length of Element along x and y axes
D_plane_stress:  Plane stress relation
alpha:  Angle made by crack tip w.r.t x-axis
CL, A: Crack length, Geometry length along X
force:  Applied force
D: Young's modulus
scale:  Scaling factor for the domain
Returns:  KI, KII
```

9 User Manual

The User manual provides the detailed information on how to run and compile the program. The manual includes information on some main user defined variables, files necessary for compiling the program, libraries and their versions used, and commands to run the main program and testing files.

9.1 User Defined Variables

nu = Poisson's Ratio

Nodes_elements = number of nodes per element (4 nodes per element)

NL = Nodes List

EL = Elements List

A = Length of the Geometry along X-axis

B = Length of the Geometry along Y-axis

x = no of divisions in X and Y directions

D = Elastic Modulus in GPa

D_planestress = Plane stress relation

GaussPoint1to4 = Gauss points (For full integration 4 Gauss points are considered)

geomns = list of coordinates of all the elements in CCW direction

UNIT = List of Element number

diag = diagonal length of the element

Klassic_mat = Classica elemental stiffness matrices holder

Verschiebung = Displacements holder

Belastung = Stress holder

Spannung = Strain holder

gamma, beta, cracks = list of all the crack segments

T_Nodelist = list of Nodes which are tip enriched (list of 4 nodes)

Tipmatrix = Tip enriched Matrix

T_Elem = Element number which is tip enriched

Enr_matrix = Holder for enriched stiffness matrices

H_Nodelist = list of Nodes which are Heaviside enriched (list of 4 nodes)

H_matrix = Heaviside enriched Matrix

H_Elem = Element number which is Heaviside enriched

MIX_N = List of nodal coordinates except for the enriched nodes

MIX_E = List of element numbers except for the enriched element numbers

PT_matrix = Pre tip enriched element's stiffness matrix

NON_N,NON_E = Non enriched nodes and elements list

N_matrix, B_matrix = Matrices of non enriched and Blended elements respectively

N_elements, B_elements = list of non enriched element numbers and Blended element numbers list respectively

N_nodes, B_nodes = list of non enriched nodes and Blended nodal list respectively

K_global = Global stiffness matrix

CLASS_DOFs = Only classical degrees of freedom (8 per element)

Tip, Heavy = sorted and unique list of enriched element numbers

Total_Dofs = Total geometry degrees of freedom (classical DOFs and additional DOFs due to enrichments)

H1, H2, H3, H4 = Heaviside function values

B_std = Standard B-matrix

N = Shape functions list

dN = differentiated shape functions

jacobi = Jacobian matrix

dN_en = Enriched shape functions list

F11, F21, F31, F41 = Asymptotic function evaluations for tip containing elements

3rd letter in the each variable denotes node number

Bt_D_B_TH = B-matrix * plane stress relation * B-matrix'

r1, theta1 = distance and angle from the crack tip to the point under query (number 1 denotes the node that is under query)

xi_1, xi_2 = shape function variables

TL, TR = Top left and top right of the matrix

BL, BR = bottom left and top right of the matrix

dNdxi = differentiation of shape functions w.r.t x and y

F1x1, F1y1 = differentiation of asymptotic function w.r.t x and y

dummy = matrix whose size in not equal to 40x40

Tside = unique list of tip enriched element numbers

Hside = unique list of Heaviside enriched element numbers

A_matrix = Assignment matrix of each elememt

H_dof, T_dof = Heaviside and Tip enriched degrees of freedom

Ux_Uy = displacements along x and y direction

Epsilon_ij = Strain matrix generated using small strain thoery

scale = scaling factor of drawing the circle

Auxiliary_disps = Auxiliary displacements

CTCS = crack tip coordinate system

alpha = angle made by the crack w.r.t to global x-axis

q1, q2, q3, q4 = corresponding nodal values

Ux, Uy, Vx, Vy = displacement gradients w.r.t x and y

CTCS_displacements = Transformed displacements

Aux_stress11 = Auxiliary stress component

I1, I2 = first and second part of the interaction integral for mode 1 crack

K1, K2 = SIFs for mode I and mode II

9.2 Libraries

The list of inbuilt libraries and their versions used in the program is given below

Python:3.9.5

numpy: 1.18.5

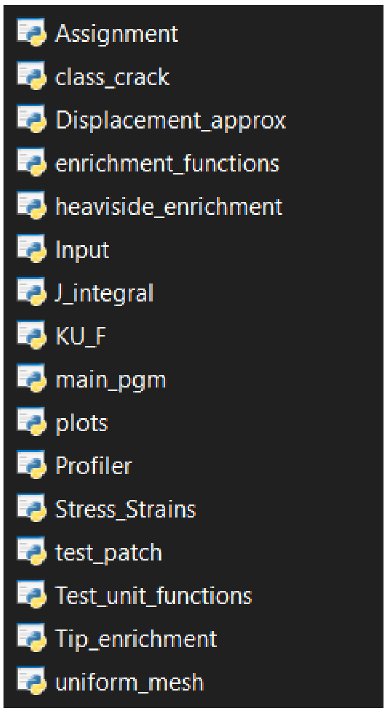
scipy: 1.5.0

matplotlib: 3.2.2

shapely: 1.8.0

9.3 Necessary Files

The list of files that is necessary to run and compile the program is shown below in Figure 15



```
Assignment
class_crack
Displacement_approx
enrichment_functions
heaviside_enrichment
Input
J_integral
KU_F
main_pgm
plots
Profiler
Stress_Strains
test_patch
Test_unit_functions
Tip_enrichment
uniform_mesh
```

Figure 15: Files necessary to run the program

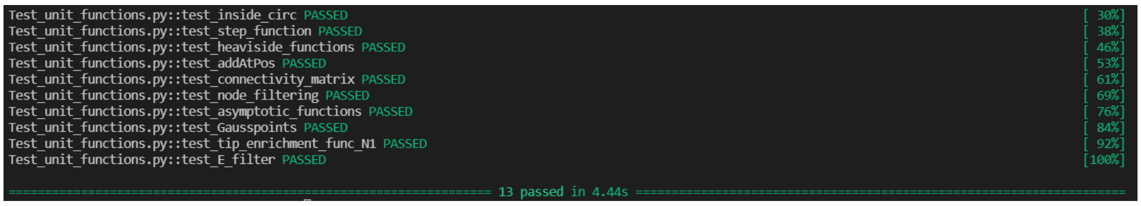
9.4 Commands to run the program and test

Command to run the program: python -c 'import Input; Input.MainFunction()'

command to run all the patch tests: pytest -v test_patch.py

command to run all the unit tests : pytest -v Test_unit_functions.py

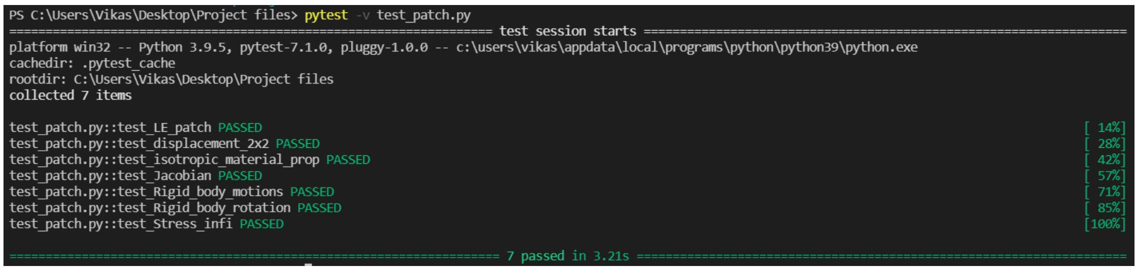
command to run time analysis: python Profiler.py



```
Test_unit_functions.py::test_inside_circ PASSED                                  [ 30%]
Test_unit_functions.py::test_step_function PASSED                                [ 38%]
Test_unit_functions.py::test_heaviside_functions PASSED                          [ 46%]
Test_unit_functions.py::test_addAtPos PASSED                                     [ 53%]
Test_unit_functions.py::test_connectivity_matrix PASSED                          [ 61%]
Test_unit_functions.py::test_node_filtering PASSED                               [ 69%]
Test_unit_functions.py::test_asymptotic_functions PASSED                         [ 76%]
Test_unit_functions.py::test_Gausspoints PASSED                                  [ 84%]
Test_unit_functions.py::test_tip_enrichment_func_N1 PASSED                       [ 92%]
Test_unit_functions.py::test_E_filter PASSED                                     [100%]

=================================== 13 passed in 4.44s ===================================
```

Figure 16: Message of all the unit tests passed



```
PS C:\Users\Vikas\Desktop\Project files> pytest -v test_patch.py
================================================================ test session starts =================================================================
platform win32 -- Python 3.9.5, pytest-7.1.0, pluggy-1.0.0 -- c:\users\vikas\appdata\local\programs\python\python39\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Vikas\Desktop\Project files
collected 7 items

test_patch.py::test_LE_patch PASSED                                                                                                            [ 14%]
test_patch.py::test_displacement_2x2 PASSED                                                                                                    [ 28%]
test_patch.py::test_isotropic_material_prop PASSED                                                                                             [ 42%]
test_patch.py::test_Jacobian PASSED                                                                                                            [ 57%]
test_patch.py::test_Rigid_body_motions PASSED                                                                                                  [ 71%]
test_patch.py::test_Rigid_body_rotation PASSED                                                                                                 [ 85%]
test_patch.py::test_Stress_infi PASSED                                                                                                         [100%]

================================================================= 7 passed in 3.21s ==================================================================
```

Figure 17: Message of all the patch tests passed

10 Testing and verification

All the tests that have been carried out are discussed in detail in this section.

10.1 Unit tests

10.1.1 Sanity check for Step function

Aim: To check if the step function is producing proper outputs

Procedure: 4 nodal coordinates, and crack coordinates are required to carryout the test see. Figure 18. The output will be 1,-1 or 0 based on the location of the nodes w.r.t the crack segment. The outputs act as the inputs in generating Heaviside B-matrix For the below shown figure, the expected outputs are listed below.

FUNCTION BODY FOR HEAVISIDE STEP FUNCTION

```
def step_function(A, cracks)
Parameters
A : point under query
cracks :  all the crack segment coordinates
Returns:  1 or 0 or -1
```

Expected output-1: For inclined crack, [-1,-1, 1,-1]

Expected output-2: For straight crack, [-1,-1, 1, 1]

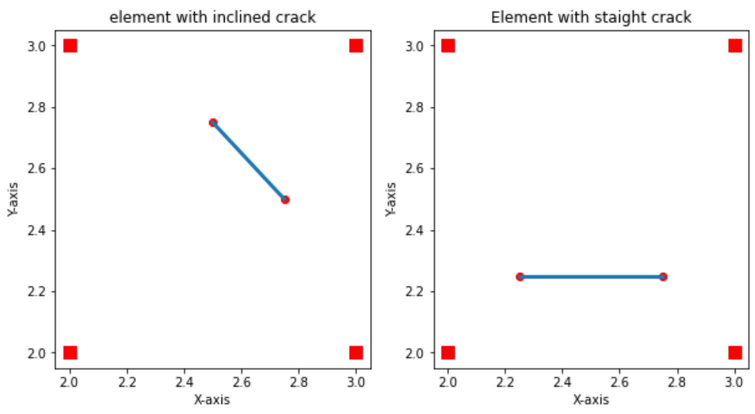


Figure 18: Crack segments inside an element
Left: inclined crack, Right : straight crack

Result:Test case passed

---

command to run the test: `py.test -v Test_unit_functions.py::test_step_function`

command to check the output: `python -c 'import Test_unit_functions;`
`Test_unit_functions.test_step_function()'`

10.1.2 Sanity check for Heaviside B-matrix

Aim: To check if Heaviside function is producing proper outputs as per the corresponding Heaviside function values

Procedure: After generating a enriched shape function matrix of shape 2x4, these functions are called to generate individual B-matrices of size 3x2. For the below enriched shape function matrix, a sample B-matrix is as expected.

Function Body For Heaviside B-matrix Generation

`def heaviside_function1(dN_en, H1)`

`The function generates a B-matrix for the "lower left" node w.r.t crack tip`

```
Parameters
-----
dN_en :  differentiated shape function W.r.t x and y (2x4 shape)
H1 :  step function value for the "lower left" node
Returns
----
B_enriched1 :  Enriched B-matrix for lower left node
```

Input:

$$\frac{\partial[N]}{\partial(x,y)} = \begin{bmatrix} \frac{dN_i}{dx} \\ \\ \frac{dN_i}{dy} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Expected output for node 1:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Result:Test case passed

---

command to run the test: `py.test -v Test_unit_functions.py::test_heaviside_functions`

command to check the output: `python -c 'import Test_unit_functions; Test_unit_functions.test_heaviside_functions()'`

10.1.3 Sanity check for Connectivity matrix

Aim: To check if the function is producing an identity matrix of size 8x8 for the particular input

Procedure: For a given element number (list of 4 numbers), the function generates a connectivity matrix of size 8x8.

Function Body For Reshaping The Matrix

```
def connectivity_matrix(EL, KL, length_nodes, Hside, Tside)
The function computes Assignment matrix for all the types of the elements
Parameters
EL : Element list [1,2,3,4]
KL : List of stiffness matrices
length_nodes :  Total nodes present in the geometry
Hside :  List of nodes that are heaviside enriched
Tside :  List of nodes that are tip enriched
Returns
----
K_global :  assembled stiffness matrix
Total_Normal_DOFs :  Total classical DOFs
Total_Dofs :  Total Geometry DOFs
Tside = sorted list of nodes that are tip enriched
Hside = sorted list of nodes that are Heaviside enriched
```

Input: [0,1,3,2]

Expected output:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Result:Test case passed

---

command to run the test: `py.test -v Test_unit_functions.py::test_connectivity_matrix`

command to check the output: `python -c 'import Test_unit_functions;`

```
Test_unit_functions.test_connectivity_matrix()
```

10.1.4 Sanity check for addAtpos function

Aim: To check if the function is producing a matrix of desired order

Procedure: For a given matrix, the function changes the order of the matrix by adding extra zeros.

Function Body For Reshaping The Matrix

```
def addAtPos(dummy, matrix)
This function is created to balance the size of the matrices.  This function is
called when the size of the matrix is not 40X40, the highest possible size for a
matrix in the sample.  The function adds two matrices of different sizes in place,
offset by xy
coordinates.
Usage:
-----
matrix:  base matrix
dummy:  add this matrix to base matrix
pos:  tuple (x,y) containing the location where the matrix to be added
Returns
result :  Matrix
```

Input: Input matrix shape 4x4

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix}$$

Expected output: Output size 5x5

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 7 & 0 \\ 8 & 9 & 10 & 11 & 0 \\ 12 & 13 & 14 & 15 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Result:Test case passed

---

command to run the test: `py.test -v Test_unit_functions.py::test_addAtPos`

command to check the output: `python -c 'import Test_unit_functions; Test_unit_functions.test_addAtPos()'`

10.1.5 Sanity check for node filtering

Aim: To check if the function is filtering the unwanted lists properly

Procedure: For a given list of nodal coordinates, the function filters out the inner lists which are enriched.

FUNCTION BODY FOR NODE FILTERING

```
def filtering(EN_list, GL_nodes)
This function filters the nodes which do not require enrichment
Parameters
-----
EN_list :  List of nodes which has been enriched
GL_nodes :  list containing all the nodes
Returns
----
result :  Filtered nodes
```

Input: List of lists of nodal coordinates

((2.0, 2.0), (3.0, 2.0), (3.0, 3.0), (2.0, 3.0))

((0.0, 2.0), (1.0, 2.0), (1.0, 3.0), (0.0, 3.0))

((1.0, 2.0), (2.0, 2.0), (2.0, 3.0), (1.0, 3.0))

((0.0, 0.0), (1.0, 0.0), (1.0, 1.0), (0.0, 1.0))

((1.0, 0.0), (2.0, 0.0), (2.0, 1.0), (1.0, 1.0))

((2.0, 0.0), (3.0, 0.0), (3.0, 1.0), (2.0, 1.0))

((3.0, 0.0), (4.0, 0.0), (4.0, 1.0), (3.0, 1.0))

Enriched nodal list:

((3.0, 0.0), (4.0, 0.0), (4.0, 1.0), (3.0, 1.0))

Expected output: After filtering

((2.0, 2.0), (3.0, 2.0), (3.0, 3.0), (2.0, 3.0))

((0.0, 2.0), (1.0, 2.0), (1.0, 3.0), (0.0, 3.0))

((1.0, 2.0), (2.0, 2.0), (2.0, 3.0), (1.0, 3.0))

((0.0, 0.0), (1.0, 0.0), (1.0, 1.0), (0.0, 1.0))

((1.0, 0.0), (2.0, 0.0), (2.0, 1.0), (1.0, 1.0))

((2.0, 0.0), (3.0, 0.0), (3.0, 1.0), (2.0, 1.0))

Result: Test case passed

---

command to run the test: `py.test -v Test_unit_functions.py::test_node_filtering`

command to check the output: `python -c 'import Test_unit_functions; Test_unit_functions.test_node_filtering()'`

10.1.6 Sanity check for element filtering

Aim: To check if the function is filtering the unwanted lists properly

Procedure: For a given list of element numbers, the function filters out the inner lists which are enriched.

Function Body For Element Filtering

```
def E_filter(EN_list, GL_elements)
This function filters the elements which do not require enrichment
Parameters
-----
EN_list :  Enriched element list
GL_elements :  list containing all the elements
Returns
----
result :  filtered elements
```

Input: List of lists of element numbers

(14.0, 15.0, 21.0, 20.0), (12.0, 13.0, 19.0, 18.0)

(13.0, 14.0, 20.0, 19.0), (10.0, 11.0, 17.0, 16.0)

(16.0, 17.0, 23.0, 22.0),**(22.0, 23.0, 29.0, 28.0)**

Enriched element list:

((22.0, 23.0, 29.0, 28.0), (16.0, 17.0, 23.0, 22.0))

Expected output: After removing the enriched element list

(14.0, 15.0, 21.0, 20.0)

(12.0, 13.0, 19.0, 18.0)

(13.0, 14.0, 20.0, 19.0)

(10.0, 11.0, 17.0, 16.0)

Result: Test case passed

Command to run the test: `py.test -v Test_unit_functions.py::test_E_filter`

Command to check the output: `python -c 'import Test_unit_functions; Test_unit_functions.test_E_filter()'`

10.1.7 Sanity check for Gauss points generation

Aim: To check if the function is generating the Gauss point coordinates properly

Procedure: For a given list of nodal coordinates, the function generates Gauss points at the calculated distance.

FUNCTION BODY FOR GAUSS POINTS GENERATION

```
def G_points(SE_ME)
The function plots the Gauss points in the global coordinate system
Parameters
-----
SE_ME : list of nodes to calculate the coordinates of new Gauss points
Returns
----
G : List of 4 Gauss points
GPs :  returns 1 Gauss point
```

Input: List of nodal coordinates

$$((2.0, 2.0), (3.0, 2.0), (3.0, 3.0), (2.0, 3.0))$$

Expected coordinates:

$$((2.25, 2.25), (2.75, 2.25), (2.75, 2.75), (2.25, 2.75))$$

Result: Test case passed

Command to run the test: `py.test -v Test_unit_functions.py::test_Gausspoints`

Command to check the output: `python -c 'import Test_unit_functions; Test_unit_functions.test_Gausspoints()'`

10.1.8 Sanity check for Asymptotic functions

Aim: To check if the function is generating right values for the given r and θ, α values

Procedure: For a given list of nodal coordinates, the function generates Gauss points at the calculated distance.

FUNCTION BODY FOR ASYMPTOTIC FUNCTIONS GENERATION

```
def asymptotic_functions(r, theta, alpha)
This function generates the necessary terms required for generating tip enriched
B-matrix.
Parameters
-----
r :  polar coordinate of the point under query, measured from the crack tip
theta :  angle of the point under query, measured from the crack tip (-pi to +pi)
alpha :  angle made by crack w.r.t global x-axis
Returns
----
F1 :  asymptotic_function1
F2 :  asymptotic_function2
F3 :  asymptotic_function3
F4 :  asymptotic_function4
dF : Differentiation of the enrichment functions associated with Shape functions
```

Input: $r = 0.25$ and $\theta = 0.15935$, $\alpha = 0$

Expected output:

(0.98916395, -0.14681514, -0.14681514, -0.98916395, -0.26253043, -1.85407775, -0.12425015, -0.5561607)

Result: Test case passed

---

command to run the test: `py.test -v Test_unit_functions.py::test_asymptotic_functions`

command to check the output: `python -c 'import Test_unit_functions; Test_unit_functions.test_asymptotic_functions()`

10.1.9 Sanity check for Tip enriched B-matrix

Aim: To check if the function is generating proper B-matrix values for the given input

Procedure: For a given list of nodal coordinates, shape functions, Jacobian matrix, differentiated shape functions, the function generates a B-matrix of shape 8x8.

Function Body For Tip Enriched B-matrix Generation

```
def tip_enrichment_func_N1(F1, F2, F3, F4, dN, dF, N)
The function generates a B-matrix for the "Lower left" node w.r.t the
corresponding element
Parameters
-----
F1 :  asymptotic_function1
F2 :  asymptotic_function2
F3 :  asymptotic_function3
F4 :  asymptotic_function4
dF : Differentiation of the enrichment functions associated with Shape functions
dN : Differentiation of Shape functions w.r.t x and y (2x4 shape)
N : Shape function values
Returns
----
B_tip1 :  B-matrix for "Lower left" node (3x8 shape)
```

Input:

((2.0, 2.0), (3.0, 2.0), (3.0, 3.0), (2.0, 3.0))

Shape functions: N1, N2, N3, N4

Asymptotic function values: F1, F2, F3, F4

Expected output:

((0.14869175, 0, 0.00895635, 0, 0.0186328, 0. -0.00104311, 0)

(0, 0.09796097, 0,-0.02866144, 0,-0.05244283, 0, -0.02033145)

(0.09796097, 0.14869175, -0.02866144, 0.00895635, -0.05244283, 0.0186328, -0.02033145, -0.00104311))

Result: Test case passed

Command to run the test: `py.test -v Test_unit_functions.py::test_tip_enrichment_func_N1`

Command to check the output: `python -c 'import Test_unit_functions; Test_unit_functions.test_tip_enrichment_func_N1()'`

10.1.10 Sanity check for Uniform mesh

Aim: To check if the function is producing proper nodal coordinates and element numbering in CCW direction

---

FUNCTION BODY FOR UNIFORM MESH GENERATION

```
def uniform_mesh(A,B,x)
This function computes all the prerequisites for the nodes and elements generations
Parameters
------
A : length along x_axis
B : length along y_axis
```

```
x :  number of elements per row
Returns
NL : list of nodes
EL : list of elements
```

Inputs: length of the geometry along x and y axes A = 1unit, B = 1unit.

Divisions along x and y axis, x = 1

Expected output-1: Nodal coordinates

$$((0, 0),(1, 0),(0, 1),(1, 1))$$

Expected output-2: Element numbering

$$(1,2,4,3)$$

Result: Test case passed

---

command to run the test: `py.test -v Test_unit_functions.py::test_uniform_mesh`

command to check the output: `python -c 'import Test_unit_functions; Test_unit_functions.test_uniform_mesh()'`

10.1.11 Sanity check for Strain matrix generator

Aim: To check if the function is producing proper strains using small strain theory see. Equation 114 [3]. The equation is given below

---

Function Body For Strain Matrix Generation

```
def Kinematics(grad_disps):
Applying small strain theory, the function computes auxiliary strains from the
auxiliary displacements
ϵij = 0.5*(Ui,j + Uj,i)
Parameters
```

```
-----
grad_disps :  Auxiliary displacements
Returns:  Auxiliary strains
```

$$\epsilon_{ij} = 0.5 * \begin{bmatrix} u_{i,i} + u_{i,i} & u_{i,j} + u_{j,i} \\ u_{j,i} + u_{i,j} & u_{j,j} + u_{j,j} \end{bmatrix} \tag{114}$$

Inputs:Displacement matrix of shape 2x2

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$$

Expected output-1: Strains computed considering small strain theory

$$\begin{bmatrix} 1 & 1.5 \\ 1.5 & 2 \end{bmatrix}$$

Result: Test case passed

---

command to run the test: `py.test -v Test_unit_functions.py::test_kinematics`

command to check the output: `python -c 'import Test_unit_functions; Test_unit_functions.test_kinematics()'`

10.1.12 Sanity check for point inside the circle

Aim: To check if the nodal coordinates lie inside or outside the prescribed circle

---

Function Body To Check of Point Lie Outside The Domain

```
def inside_circ(c_2, length_element_x, length_element_y, P, scale)
```

```
The function checks if any of the nodes is outside the domain under consideration.
If yes, it will return True, False otherwise
Parameters
-----
c_2 :  right crack tip
length_element_x :  length of the element along x-direction in "meters"
length_element_y :  length of the element along y-direction in "meters"
P : point under query (nodes)
scale :  scaling factor of the domain used for interaction integral
Returns
----
nodal value (q) 1 or 0 based on the condition
```

Inputs: Crack tip = (0.5,0.5)

length of element along x and y = 1

Nodal coordinates = [2,2]

scale of the circle = 1.5

Expected output: False (point lie outside the circle for the given inputs)

Result: Test case passed

---

command to run the test: `py.test -v Test_unit_functions.py::test_inside_circ`

command to check the output: `python -c 'import Test_unit_functions;`
`Test_unit_functions.test_inside_circ()'`

10.1.13 Sanity check matrix transformation

Aim: To check if the function is able to properly transform the matrix from global system to local crack tip coordinate system see. Equation 115

Function Body For Matrix Transformation

```
def Global_to_local_CT(S, CTCS)

CTCS = Rotation Matrix

Parameters

-----

S : stresses in global co-ordinate system

CTCS : crack tip coordinate system

Returns

----

Stresses in local crack coordinate system
```

Inputs: crack angle w.r.t x-axis, $\alpha = 90^o$

random matrix = [1,2,3]

Rotation matrix =

$$\begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \tag{115}$$

Expected output: For the current input, the corresponding output is as expected

$$\begin{bmatrix} -0.60422787 & -2.19595653 \\ -2.19595653 & 3.60422787 \end{bmatrix}$$

Result: Test case passed

command to run the test: `py.test -v Test_unit_functions.py::test_Global_to_local_CT`

command to check the output: `python -c 'import Test_unit_functions; Test_unit_functions.test_Global_to_local_CT()'`

10.2 Patch tests

10.2.1 Rigid body translation test

Aim: To check if the inner node has the same displacement as the outer nodes [10].

Procedure: In this case, we consider a 2x2 elements and prescribe a displacements on the outer nodes of the geometry see. Figure 19 and then we check the displacements of the inner node.

Example: A geometry consisting of 2x2 elements with 9 nodes see. Figure 18, node number 5 is considered as the inner node. It shares its position with all the 4 elements. Here, we prescribe displacements on the 4 corner nodes of the geometry. One should make sure that the geometry is in equilibrium.

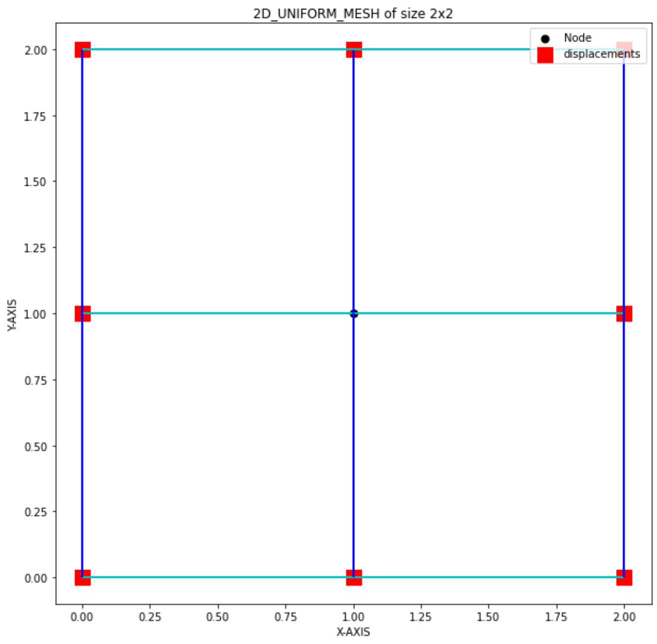


Figure 19: Sample under translation patch test
Red squares: Prescribed displacements

Expected output: If a displacement of 0.5mm is prescribed in both x and y directions, the inner node should displace at about the same (0.5mm) as the corner nodes.

Result: Test case passed

Command to run the test: `py.test -v test_patch.py::test_displacement_2x2`

Command to check the output: `python -c 'import test_patch; test_patch.test_displacement_2x2()'`

10.2.2 Linear Elastic Material Response test

This test validates if the element under experiment is a Linear elastic or not. The easiest method is to implement Newton's method [10].

Aim: To check if the material response in linear elastic in nature by implementing Newton's method.

Procedure: Here, 4 elements are considered for the test which are skewed towards the right. All the nodes in the extreme right are prescribed with traction boundary conditions and the nodes that are in the extreme left are prescribed with essential boundary conditions.

Example: In the below figure Figure 20, the bottom right node is prescribed with 2.5N, the top right node is prescribed with 2.5N, and the middle node is prescribed with 5N forces. Both the degrees of freedom are arrested for the bottom right node. u_x is arrested for the other 2 nodes. The basic idea of the implementation is given below.

Consider the equilibrium equation see. Equation 116

$$\mathbf{R}(\mathbf{u}) = \mathbf{F} - \mathbf{P}(\mathbf{u}) \tag{116}$$

wherein **F** is the vector of applied nodal forces, u is the vector of nodal displacements, and for a static linear elastic problem P is defined as see. Equation 117

$$\mathbf{P} = \mathbf{K}\mathbf{u} \tag{117}$$

in which **K** is the global stiffness matrix and u is the initial displacement vector. A solution for the see. Equation 117 is defined by requiring the residuals $\mathbf{R} = 0$.

For the first iteration prescribe displacements as zero and calculate vector P and using the P vector, calculate residuals R using the Equation 116. With the new residuals compute a new displacement vector and add the new displacement vector to the older vector Equation 118 then evaluate the above equation but

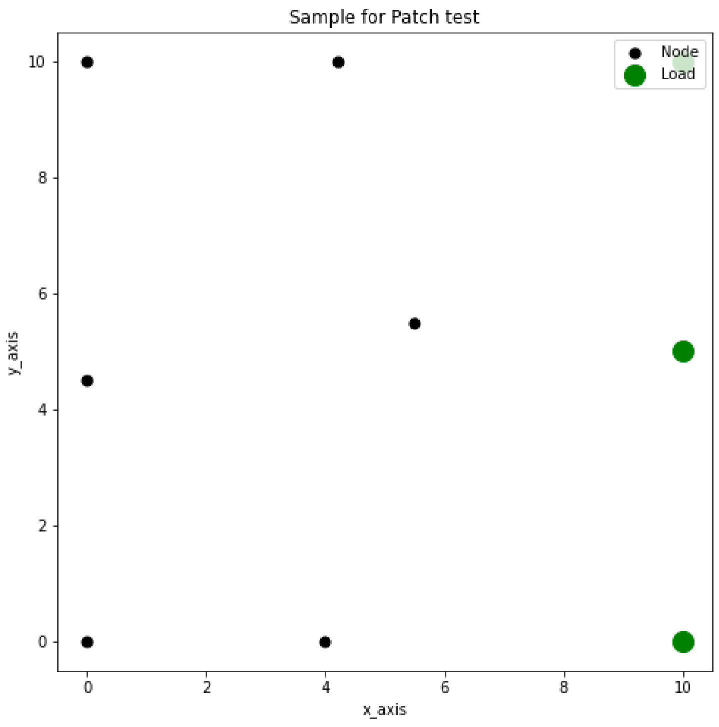


Figure 20: Sample under patch test
Green circles: Prescribed force

now the residuals will be a zero vector.

$$\mathbf{u} \leftarrow \mathbf{u} + \Delta\mathbf{u} \tag{118}$$

Expected output : Convergence after one iteration

Result: Test case passed.

Command to run the test: `py.test -v test_patch.py::test_LE_patch`

Command to check the output: `python -c 'import test_patch; test_patch.test_LE_patch()'`

10.2.3 Sanity check for Isotropic material property

Aim: To check if the element is producing same stresses and strains at all the 4 Gauss points[10].

Procedure: Here, a single element is considered for the test see. Figure 21. The element is at equilibrium position under tensile load applied in Newtons. the equilibrium equation $\mathbf{Ku} = \mathbf{f}$, is solved to get the displacements and using these displacements, the stresses and strains are calculated at 4 Gauss points. The stresses and strains that are obtained must have same values at all the Gauss points. In the below figure stresses and strains have been computed at the red points.

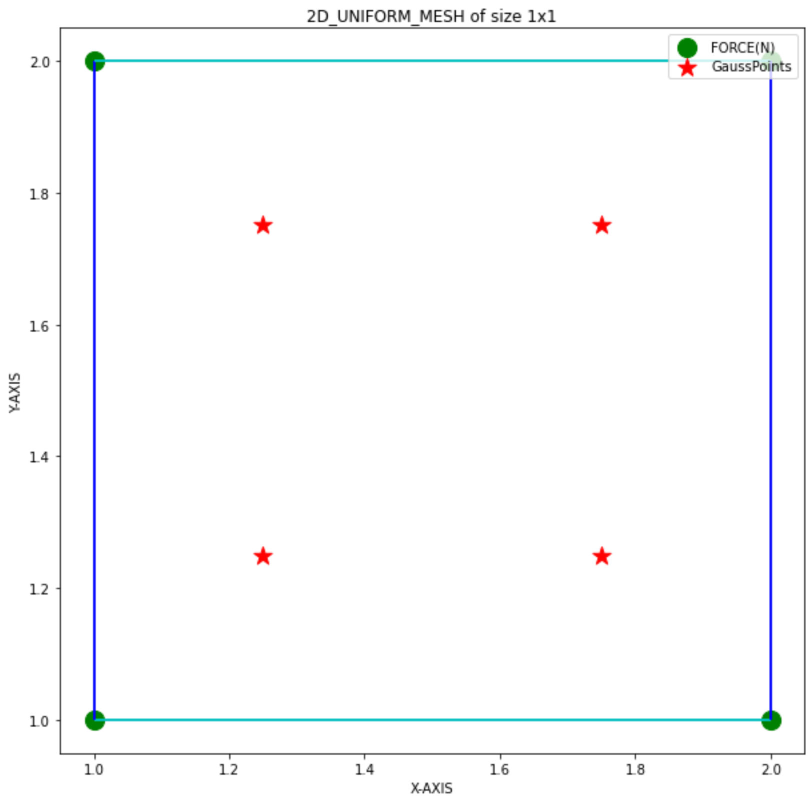


Figure 21: Stresses and strains for 1 element at 4 Gauss points

Expected output: Same stresses and strains values at all the 4 Gauss points

Result: Test case passed.

Command to run the test: `py.test -v test_patch.py::test_isotropic_material_prop`

Command to check the output: `python -c 'import test_patch; test_patch.test_isotropic_material_prop()'`

10.2.4 Sanity check for Shape functions

Aim: To check the properties of the shape functions[7].

Procedure: Here, for a 2d full integration scheme, 4 shape functions are taken into account. Initially, the summation of the shape functions are checked for one random Gauss point (Ex: $\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$) and it should be 1 and the summation of the differentiated shape functions should be 0

Expected output-1:

$$\sum_{i=1}^{4} N_i = 1$$

Expected output-2:

$$\sum_{i=1}^{4} \frac{\partial N_i}{\partial x} = 0 \qquad \sum_{i=1}^{4} \frac{\partial N_i}{\partial y} = 0$$

Result: Test case passed.

10.2.5 Sanity check for Jacobian matrix

Aim: To check if the Jacobian matrix obtained, is volume conserving or not irrespective of the rotation angle[10].

Procedure: For a given element coordinate system and Gauss points, a Jacobian matrix and its determinant is computed computed. Using Euler angles, the Jacobian matrix is rotated and then the determinant is computed.

Expected output: The determinant values before and after the rotation should remain same. This proves that Jacobian matrix is volume conserving.

Result: Test case passed.

Command to run the test: `py.test -v test_patch.py::test_Jacobian`

Command to check the output: `python -c 'import test_patch; test_patch.test_Jacobian()'`

10.2.6 Sanity check for Rigid body motions

Aim: To check if the global stiffness matrix is producing prescribed zeros for rotation and translation motion for the constrained and unconstrained structures. [10].

Procedure: For a given structure (1x1 element), global stiffness matrix is computed and then Eigenvalues are computed. If the 2D structure is unconstrained, then the number of zero Eigenvalues should be 3 (2-rotation, 1-translation). If one of the corners of the structure is constrained, then the number of zero Eigenvalues should be 0.

Expected output:Unconstrained structure=3

constrained structure = 0

Result: Test case passed.

Command to run the test: `py.test -v test_patch.py::test_Rigid_body_motions`

10.2.7 Sanity check for Rigid body rotation

Aim: After performing the transformation, global stiffness matrix should be able to represent rigid body rotation [10].

Procedure: For a given structure (2x2 elements), all the nodes are rotated using the rotation matrix leading to the new positions. Now, the external nodes are rotated using transformation matrix. The new positions of the nodes acts as an input to the displacement vector. It should be mentioned here that the rotation is done w,r,t to the center node therefor, the displacement values of the center node is [0,0]. Using the available displacement and global stiffness matrix, the displacements of the inner node is calculated and compared with the initially rotated nodes.

Expected output:Inner node should have the displacement value of the initially rotated nodes

Result: Test case passed.

Command to run the test: `py.test -v test_patch.py::test_Rigid_body_rotation`

10.2.8 Sanity check for infinite stress at the crack tip

Aim:To check if the stress near the crack tip is infinity [4]

Procedure: For a given structure (1x1 element), compute displacements by solving the BVP. This time only one Gauss point is used at (0,0). Compute the global coordinate of the Gauss point and then calculate the distance from the Gauss point to the crack tip it should be 0. Now, if any stress component is calculated, it should be infinity. This phenomenon is observed because of the term $\frac{1}{\sqrt{(r)}}$, is present in Equation 85 from which the stresses are calculated.

Expected output:The stress near the crack tip is infinity

Result: Test case passed.

Command to run the test: `py.test -v test_patch.py::test_Stress_infi`

11 Results and Observations

Both XFEM and Fracture mechanics have been implemented using python. The robustness and correctness of the implementation will be validated using 3 experiments and the observations are recorded. Following are material properties and dimensions taken into consideration.

Young's Modulus : $200x10^3$MPa

Poisson's ratio, $\nu = 0.25$

Length of the geometry along x-direction: 2000mm

Length of the geometry along y-direction: 2000mm

No of elements: 20x20

Length of each element in x and y directions: 100mm

Total area of the geometry:$4x10^6$mm^2

$cracklength : 950mm(halfofthegeometrylength)$

$BoundaryValueproblem : PlaneStress$

$cracklocation : centerofthegeometryanditisanedgecrack$

$Boundaryconditions : Stress(MPa)orDisplacement(mm)Assumption : Plasticzoneissmallercomparedtothethick$

11.1 Stress analysis in pure mode I loading

The sample is under pure tension and the illustration is depicted in Figure 22 in which the bottom edge has been completely arrested (0 DOfs) and it has been represented using blue points and a displacement of 5mm has been applied perpendicular to top edge of the geometry, represented using yellow points. After solving the BVP, the stresses are calculated at the Gauss points of each element, the distribution of σ_x, σ_y and σ_{xy} has been shown in Figure 23, Figure 24 and Figure 25 respectively. Using Interaction integral, energy release rate and stress intensity factors K_I and K_{II} are computed.

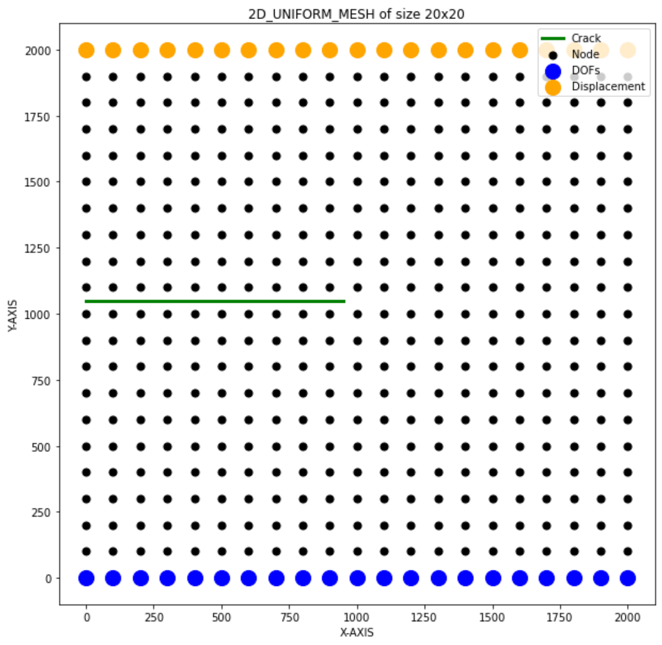


Figure 22: Sample for Mode I and Mode II failure

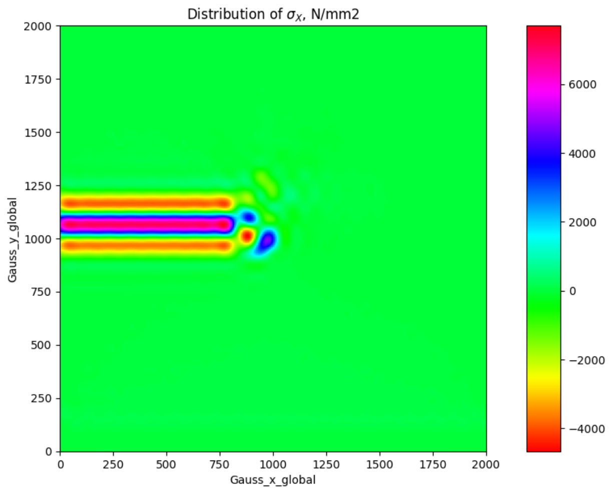


Figure 23: Distribution of σ_x under mode I

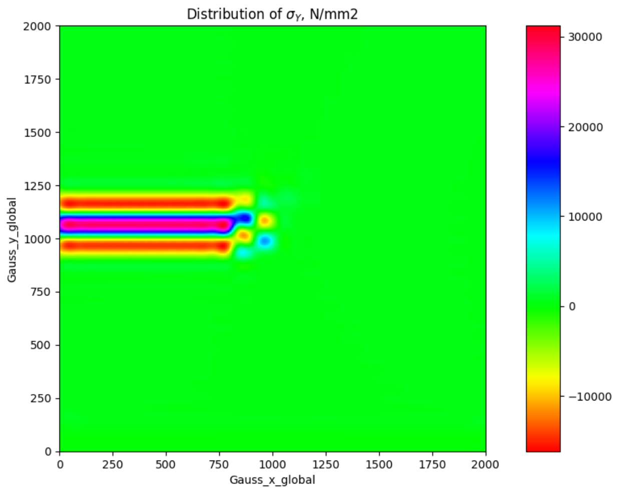


Figure 24: Distribution of σ_y in mode I

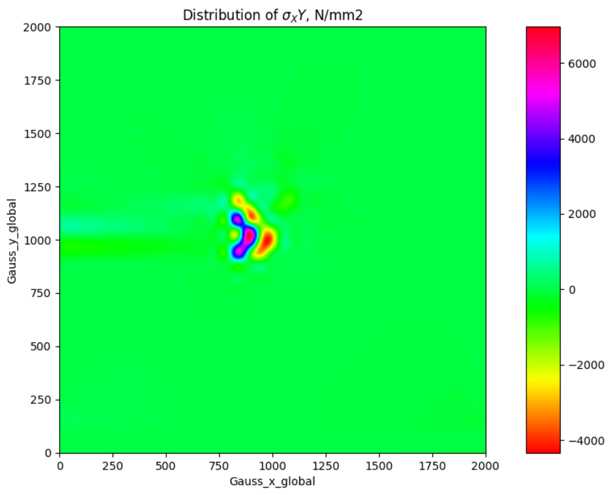


Figure 25: Distribution of σ_{xy} under mode I

11.1.1 Observations

From Figure 23, it is observed that the stresses σ_x and σ_y are not generated on the path rather they are concentrated at the crack tip. This is obtained because of the loading conditions. As, the displacements are applied parallel to crack face, the shear stress σ_{xy} generated is more compared to σ_x and σ_y. The recorded stress σ_x near the crack tip is 6000 $\frac{N}{mm^2}$ which is less compared to the σ_y in see Figure 24 which is 30000 $\frac{N}{mm^2}$.

It is observed that the redistribution of the stresses near the crack tip is lesser compared to mode I failure. Because it is assumed that the model has undergone ductile fracture analysis and therefore the mechanism does not encompass severe plastic deformations. It is observed form Figure 25, that due to the loading conditions, the shear stress generated along the crack path is minimal. The energy release rate and Stress intensity factors obtained for the mode I failure are given in Table 2

11.2 Stress analysis in pure mode II loading

The sample is under simple shear and the illustration is depicted in Figure 22.The loading conditions are similar to the mode I failure except that the displacements are applied parallel to the crack face. The distribution of σ_x, σ_y and σ_{xy} has been shown in Figure 26, Figure 27 and Figure 28 respectively.

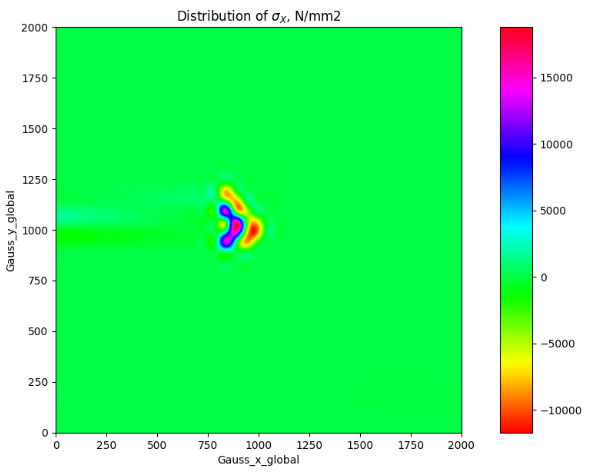


Figure 26: Distribution of σ_x under mode II

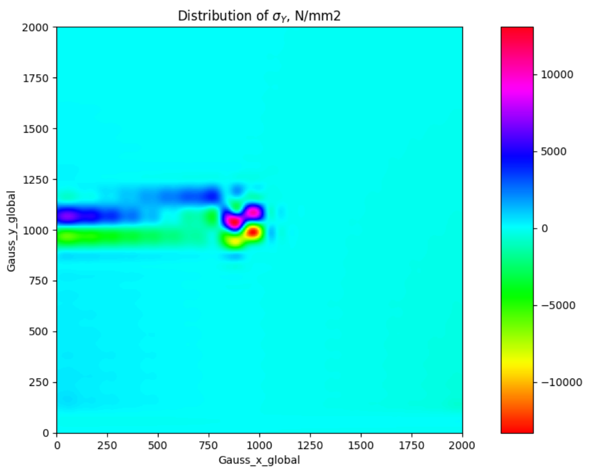


Figure 27: Distribution of σ_y in mode II

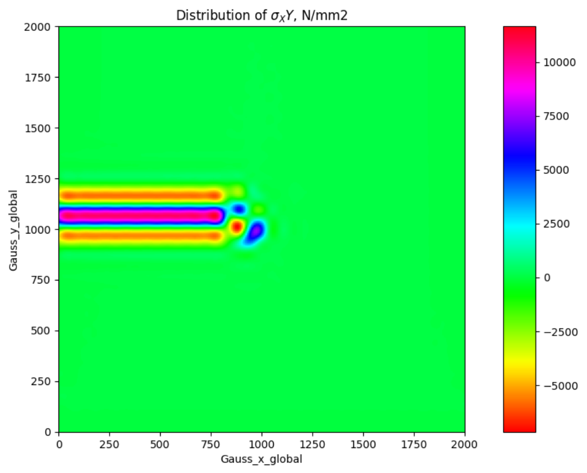


Figure 28: Distribution of σ_{xy} in mode II

11.2.1 Observations

From Figure 26, one can extrapolate that the stresses are generated on the path of the crack, rather it is concentrated near the crack tip and stress has been redistributed to the material farther out. This is observe because the material has yielded and cannot tolerate stresses much above the yield stress and therefore the true size of the plastic zone is larger than it was assumed. The maximum recorded stress σ_x near the crack tip is 15000 $\frac{N}{mm^2}$ and σ_y in see Figure 27 which is 10000 $\frac{N}{mm^2}$.
It is observed that the redistribution of the stresses near the crack tip is minimal. Because it is assumed that the model has undergone ductile fracture analysis and therefore the mechanism does not encompass severe plastic deformations. The energy release rate and Stress intensity factors obtained for the mode II failure are provided in Table 2

11.3 Stress analysis in Mixed mode loading

The sample is under mixed mode loading and the illustration is depicted in Figure 29.In this experiment, the stress is applied at some distance Y_0 from the left crack tip. A stress magnitude of 500 $\frac{N}{mm^2}$ is applied perpendicular to the node and a same magnitude of stress is applied parallel to the node. The example has been taken from [9]. The SIF values obtained after the computation is compared with the theoretical SIF values. The distribution of σ_x, σ_y and σ_{xy} has been shown in Figure 30, Figure 31 and Figure 32 respectively.

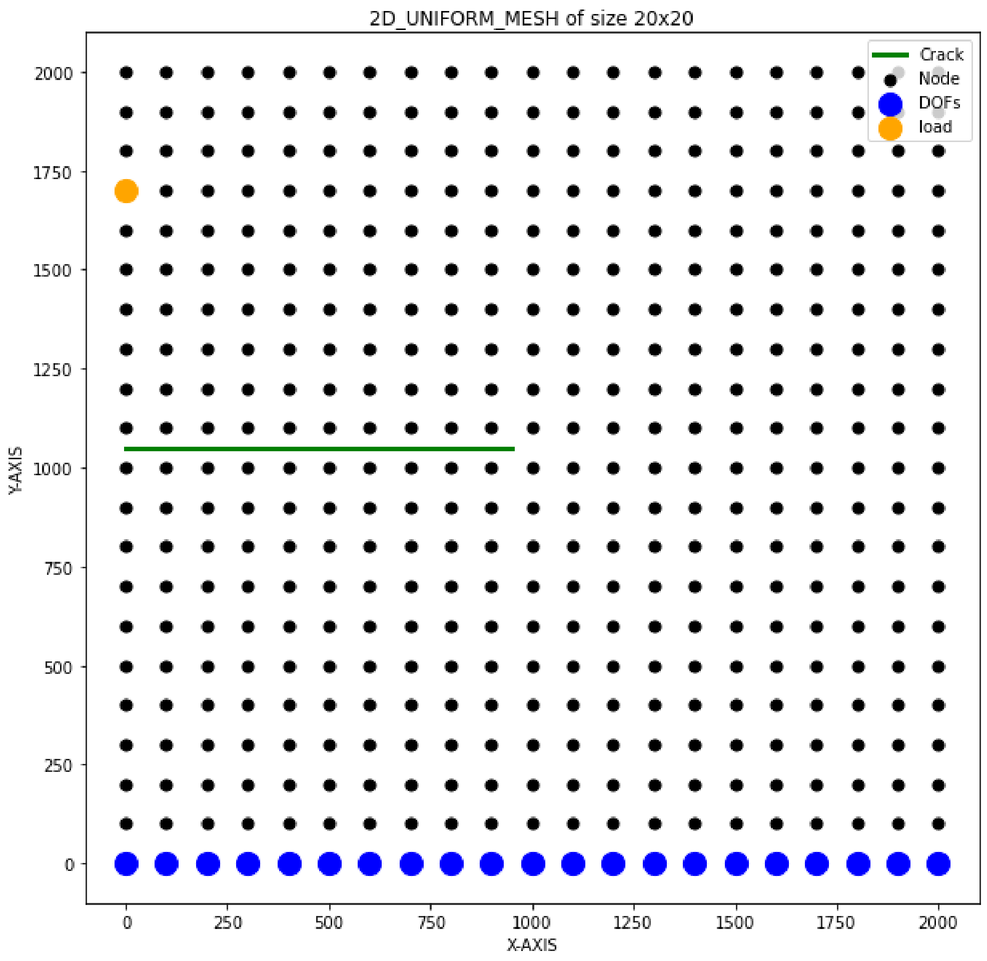


Figure 29: Sample for Mixed-mode failure

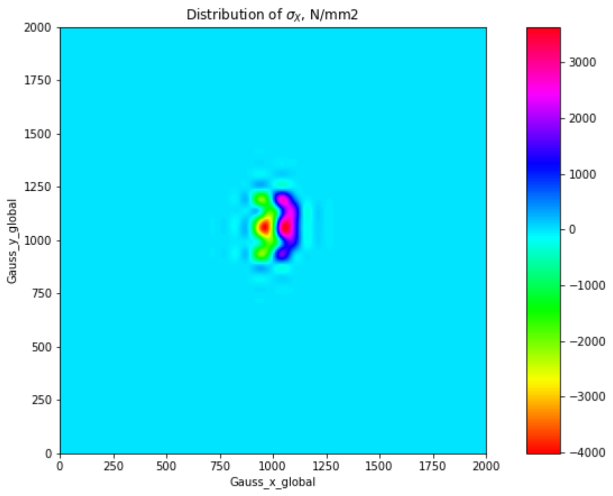


Figure 30: Distribution of σ_x in mixed-mode

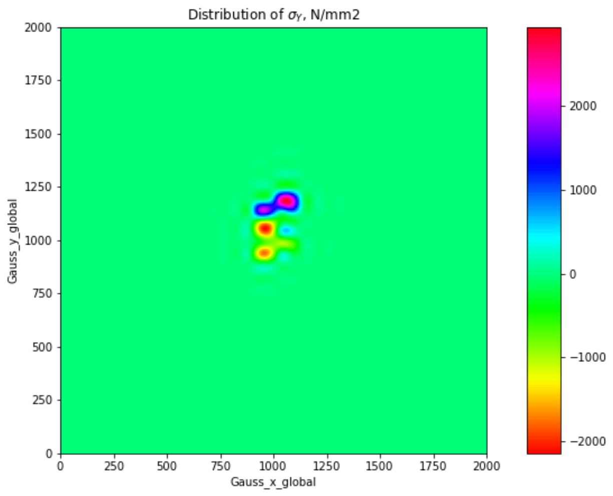


Figure 31: Distribution of σ_y in mixed-mode

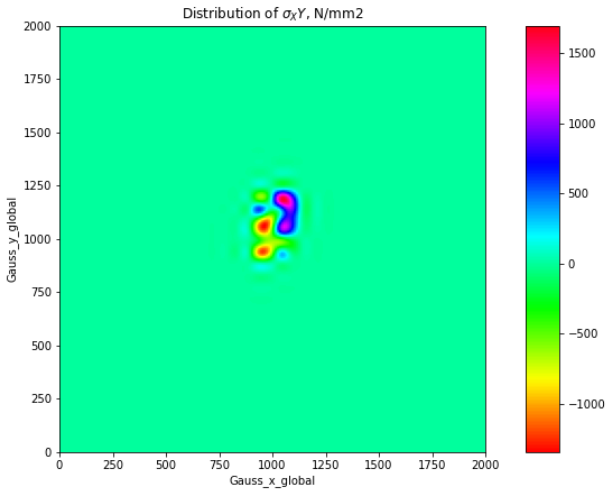


Figure 32: Distribution of σ_{xy} in mixed-mode

11.3.1 Observations

From Figure 30, Figure 31 and Figure 32 one can extrapolate that the stresses are not generated on the path of the crack, rather it is concentrated near the crack tip and stress has been redistributed to the material farther out. This is observe because the material has yielded and cannot tolerate stresses much above the yield stress and therefore the true size of the plastic zone is larger than it was assumed. The maximum recorded stresses σ_x near the crack tip is 3000 $\frac{N}{mm^2}$, σ_y in Figure 30 which is 2000 $\frac{N}{mm^2}$, and σ_{xy} in Figure 32 is 1500 σ_{xy} .

It is observed that the redistribution of the stresses near the crack tip is minimal. Because it is assumed that the model has undergone ductile fracture and therefore the mechanism does not encompass severe plastic deformations. The energy release rate and Stress intensity factors obtained for the mixed mode loading are mentioned in Table 2

11.3.2 Analytical solution for mixed mode loading

The above mention mixed mode example see Figure 29 and Figure 33[9] and the theoretical results have been calculated using the below relations.

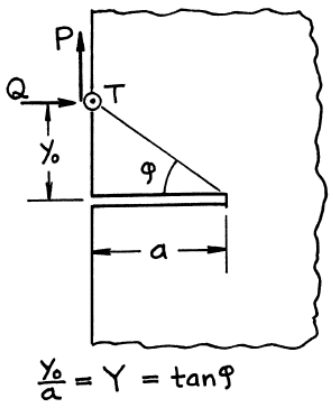


Figure 33: Mixed mode loading[9]

$$\begin{Bmatrix} K_I \\ K_{II} \end{Bmatrix} = \frac{1}{\sqrt{\pi a}} \begin{Bmatrix} P \cdot F_1(Y) \\ Q \cdot F_3(Y) \end{Bmatrix} - \frac{1}{\sqrt{\pi a}} \cdot \frac{2}{\pi} \begin{Bmatrix} Q \\ P \end{Bmatrix} \cdot F_2(Y) \tag{119}$$

$$
\begin{aligned}
F_1(Y) &= \frac{1+2Y^2}{(1+Y^2)^{3/2}}\left[1.3-.3\left(\frac{Y}{\sqrt{1+Y^2}}\right)^{5/4}\left\{.665-.267\left(\frac{Y}{\sqrt{1+Y^2}}\right)^{5/4}\left(\frac{Y}{\sqrt{1+Y^2}}-.73\right)\right\}\right.\\
F_2(Y) &= \left[\frac{1}{1+Y^2}+\frac{Y^2}{(1+Y^2)^{3/2}}\tanh^{-1}\left(\frac{1}{\sqrt{1+Y^2}}\right)\right]\left[1.3-.375\frac{Y}{\sqrt{1+Y^2}}\left(1-.4\frac{Y}{\sqrt{1+Y^2}}\right)\right]\\
F_3(Y) &= \frac{1}{(1+Y^2)^{3/2}}\left[1.3-.75\frac{Y}{\sqrt{1+Y^2}}\left(1-1.184\frac{Y}{\sqrt{1+Y^2}}+.512\frac{Y^2}{1+Y^2}\right)\right]
\end{aligned}
\tag{120}
$$

Kth_I : 1.3351026274999267 $\quad$ MPa$\sqrt{mm}$

Kth_{II} : -0.9419196333744306 $\quad$ MPa$\sqrt{mm}$

Theoretical energy release rate, Gth: 1.3348558108467148e-05 $\quad \frac{N}{mm}$

The theoretical energy release rate has been calculated from Equation 113

Type of Loading	No of Elements	Load	K_I MPa$\sqrt{(mm)}$	K_II MPa$\sqrt{(mm)}$	G (N/mm)
Pure Mode I	20x20	5mm	43.54863434	-299.14214843	0.45691254
Pure Mode II	20x20	5mm	13.84484329	-26521.68345064	3516.99846528
Mixed Mode	20x20	500 N/mm^2	-0.02073148	1.15505595	0.01651803

Table 2: Table of results for computed values

12 Time Analysis

The first 3 lines of the log file has been used to create the time analysis section. The first line gives the information on the name of the function. Second line gives the information on date, time and data (file name.prof). Third line indicates the number of function calls and time consumed by the CPU in seconds.

```
    ==========test_uniform_mesh==========
Fri Mar 25 07:51:34 2022        F1.prof
657 function calls (640 primitive calls) in 0.002 seconds


    ==========test_kinematics==========
Fri Mar 25 07:51:34 2022        F2.prof
1118 function calls (1089 primitive calls) in 0.004 seconds


    ==========test_Global_to_local_CT==========
Fri Mar 25 07:51:34 2022        F3.prof
1583 function calls (1542 primitive calls) in 0.005 seconds


    ==========test_inside_circ==========
Fri Mar 25 07:51:34 2022        F4.prof
1640 function calls (1599 primitive calls) in 0.006 seconds


    ==========test_step_function==========
Fri Mar 25 07:51:35 2022        F5.prof
357435 function calls (350183 primitive calls) in 0.702 seconds


    ==========test_heaviside_functions==========
Fri Mar 25 07:51:35 2022        F6.prof
357826 function calls (350556 primitive calls) in 0.703 seconds
```

```
==========test_addAtPos==========
Fri Mar 25 07:51:35 2022    F7.prof
358804 function calls (351474 primitive calls) in 0.704 seconds

==========test_connectivity_matrix==========
Fri Mar 25 07:51:35 2022    F8.prof
361449 function calls (353975 primitive calls) in 0.707 seconds

==========test_node_filtering==========
Fri Mar 25 07:51:35 2022    F9.prof
361592 function calls (354118 primitive calls) in 0.707 seconds

==========test_asymptotic_functions==========
Fri Mar 25 07:51:35 2022    F10.prof
362160 function calls (354670 primitive calls) in 0.709 seconds

==========test_Gausspoints==========
Fri Mar 25 07:51:35 2022    F11.prof
362791 function calls (355293 primitive calls) in 0.710 seconds

==========test_tip_enrichment_func_N1==========
Fri Mar 25 07:51:35 2022    F12.prof
363995 function calls (356443 primitive calls) in 0.713 seconds

==========test_E_filter==========
Fri Mar 25 07:51:35 2022    F13.prof
364112 function calls (356560 primitive calls) in 0.713 seconds

==========test_LE_patch==========
Fri Mar 25 07:51:35 2022    F14.prof

440101 function calls (432078 primitive calls) in 6.792 seconds


    ==========test_displacement_2x2==========

Fri Mar 25 07:51:35 2022    F15.prof

365116 function calls (357554 primitive calls) in 0.717 seconds


    ==========test_isotropic_material_prop==========

Fri Mar 25 07:51:35 2022    F16.prof

366298 function calls (358724 primitive calls) in 0.722 seconds


    ==========test_Jacobian==========

Fri Mar 25 07:51:35 2022    F17.prof

368182 function calls (360552 primitive calls) in 0.727 seconds


    ==========MainFunction==========

Fri Mar 25 20:25:37 2022    F18.prof

2519091 function calls (2494063 primitive calls) in 24.391 seconds
    ==========test_Rigid_body_motion==========

Fri Mar 25 07:51:45 2022    F19.prof

1892899 function calls (1856279 primitive calls) in 10.609 seconds


    ==========test_Rigid_body_rotation==========

Fri Mar 25 07:51:45 2022    F20.prof

1893919 function calls (1857289 primitive calls) in 10.612 seconds
```

The below pie chart Figure 34 gives you an idea of the time elapsed by all the test functions and the main function.

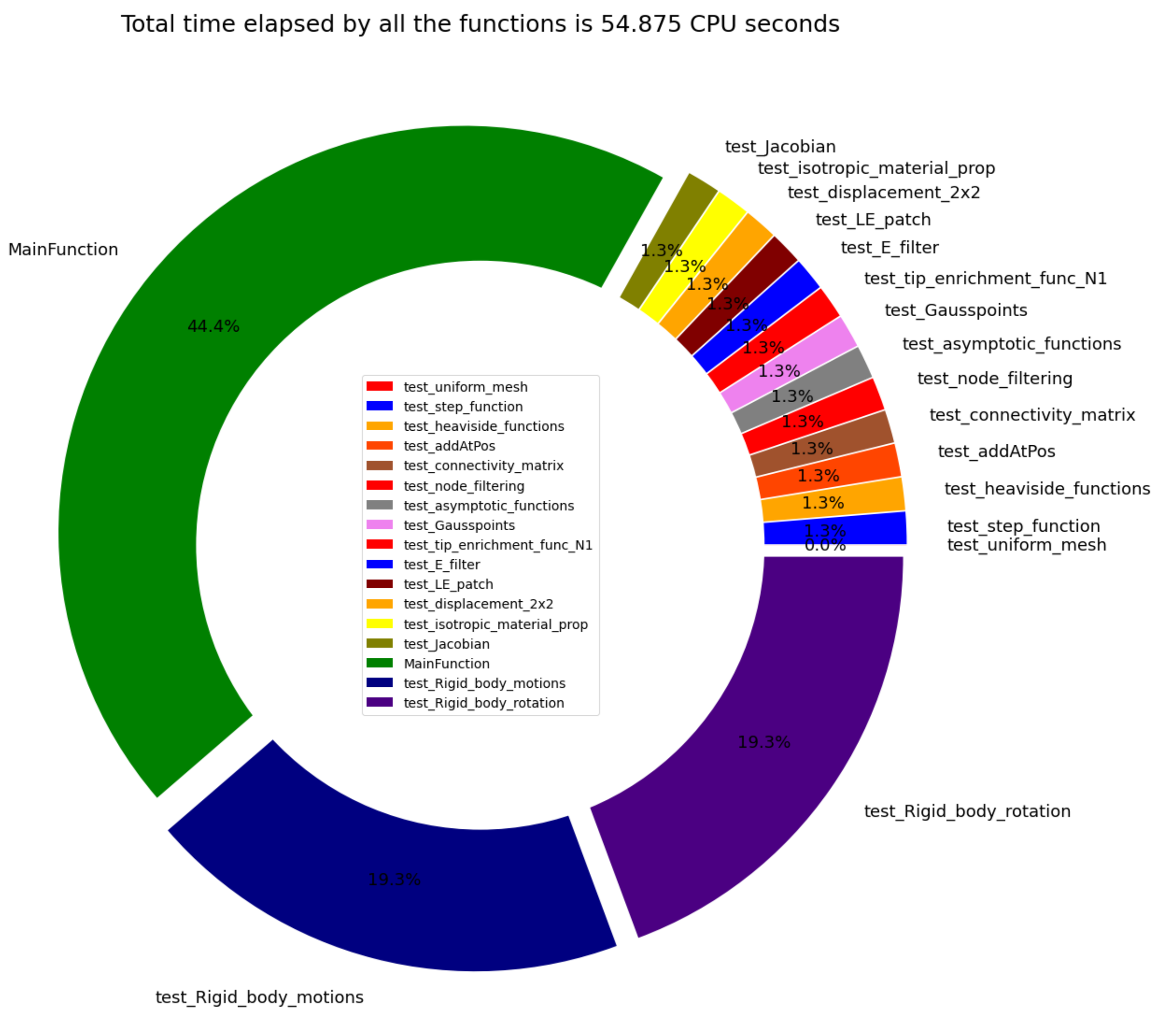


Figure 34: Time Analysis Pie Chart

13 Git Log

Some of the git commit logs have been listed here.

```
commit 93af61b2e865a1c0e6551f69575f248157ad5f57

Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>

Date:  Sat Feb 26 22:19:49 2022 +0100

completely updated


   commit 632fa39a16acae9766c4709c4c5741824c62d7d7

Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>

Date:  Sat Feb 26 22:19:19 2022 +0100

J-integral flowchart


   commit 7ce99f9cca2da60c59f216c5695f079b25771589

Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>

Date:  Sat Feb 26 22:18:51 2022 +0100

flowchart added


   commit b1f856d8b4d382cf427a341485c114dacef4e4f1

Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>

Date:  Fri Feb 25 17:28:04 2022 +0100

draft report

Necessary files to run .tex file.


   commit 8c84ab4f2d3c78da1fa34ea6c12e5dd4f798093d

Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>

Date:  Fri Feb 25 17:26:21 2022 +0100

Delete Documents directory
```

```
  commit 43aa8b7f760136580435729aa143d8643198fc48
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Fri Feb 25 17:24:28 2022 +0100
Report draft

  commit 1f7bd85d4a2757a6373bbaa4046b5247e5a8dfbd
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Thu Feb 17 12:36:59 2022 +0100
updated

  commit 3f8a7549a0e4b5ee2038d9e17a340f21a5d0adbd
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Thu Feb 17 12:36:18 2022 +0100
Delete important_patch test.py

  commit 8d96977de11684a6bc953f3b6c53d4eb64f4ae8c
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Fri Feb 11 19:11:04 2022 +0100 Delete affected_enrichments.py

  commit 43df3cc875dc40ca764a4eb715656e77294155cf
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Fri Feb 11 19:10:27 2022 +0100
Delete enrichment_functions.py

  commit 0dc37e9546a13659dd1be440f9241e3ccc910f14
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Sat Feb 5 23:25:18 2022 +0100
Delete Assignment.py
```

```
  commit 708195d78b40b5900b5c01b73376eb9e91c988e2
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Sat Feb 5 23:09:49 2022 +0100
Delete Tip_enrichment.py

  commit cee05729644ee169d554c8fba5a64b41b0aeb3ea
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Sat Feb 5 23:09:20 2022 +0100
Delete Stress_Strains.py

  commit 5f3ad85e148f497f9f143598935def28020ea967
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Sat Feb 5 13:07:30 2022 +0100
Delete class_crack.py

  commit 631ee1a3e2058e98b80721058d7e200850819a51
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Sat Feb 5 13:06:57 2022 +0100
Delete new_crack.py

  commit fd5c81ad1bea5391250bf39851cc61a7eeff65f9
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Sat Feb 5 13:06:28 2022 +0100
Delete J_integral.py

  commit d543572aba18685500c6a658cf6ab6ca03817db8
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Mon Jan 17 22:28:24 2022 +0100
Delete Tip_enrichment.py
```

```
   commit 352e6e70d314a165a39f00c615c54c1c9e0522d1
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Mon Jan 17 22:28:00 2022 +0100
Delete uniform_mesh.py

   commit 8554bb529eee80758442408bd3b2a82812452ae2
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Mon Jan 17 22:27:27 2022 +0100
Delete Assignment.py

   commit db44b4db2adb38b3d263989e9b6cb87d2d8a64c0
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Sun Jan 9 10:15:19 2022 +0100
new plots

   commit d877ebf14bcfd8ebf4c75088f18b2a43a750291a
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Sun Jan 9 10:15:03 2022 +0100
2 forces applied

   commit 40c97f20f7415681b1b99d45b88b5e3e93e44faf
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Sun Jan 9 10:14:13 2022 +0100
Delete Stress_Strains.py

   commit 6b880ce4db796537d1e07ce0f48f9b3b1984ba8c
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Sun Jan 9 10:13:56 2022 +0100
```

```
Delete plots.py

   commit f9c16de643cf3c659e0339d043dfba744962a4b9
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Sun Jan 9 10:13:47 2022 +0100
Delete uniform_mesh.py

   commit 8477c784a2a47551bb87cbc3f1ecac41430d5fee
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Fri Jan 7 22:00:58 2022 +0100
Add files via upload

   commit 7a4a83a21871e6239395a7ced05f5f59e0395051
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Fri Jan 7 22:00:17 2022 +0100
new functions

   commit c754da15abe0fbcbabad7c1d97d7f8710fddbbfa
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Fri Jan 7 21:58:10 2022 +0100
Delete Displacement_approx.py

   commit cd25e07fbe2dde8fc8d1045e4b2edbd068183b24
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Fri Jan 7 21:57:48 2022 +0100
Delete new_crack.py

   commit fae0e5484006cfde18c46d9c87871ff3f376b2ce
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
```

```
Date:  Fri Jan 7 21:57:27 2022 +0100
Delete

  commit cd3e902cefc72a3f7870fb5390bae8eaf341e237
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Fri Jan 7 21:57:03 2022 +0100
new plots

  commit 524547af60a201ff4a5c9174ca517fbf68365551
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Fri Jan 7 21:56:40 2022 +0100
new update

  commit 78cca7adc5489878a0a9793e35d166a25437a840
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Fri Jan 7 21:56:29 2022 +0100
new changes

  commit 27041128204ead18e132a817842feb914e3d403c
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Tue Jan 4 20:44:20 2022 +0100
updated

  commit e3a5131d7386038586685326f15304e3129f45c7
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>
Date:  Tue Jan 4 20:44:02 2022 +0100
updated

  commit 2372d1fc3f1e35c568ed4a6f1168ac69460f6be8
```

```
Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>

Date:  Tue Jan 4 20:43:21 2022 +0100

new update


   commit 27cb4ef6b1ddfee19d5091afcf76c623eb55d829

Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>

Date:  Tue Jan 4 20:43:07 2022 +0100

new update


   commit 930fcf83e61b162f7699e08762a08fadb7f4a76f

Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>

Date:  Mon Jan 3 23:48:25 2022 +0100

Added new functions


   commit 3d5cc4fc617a6eba5bdef5c6080cd6f65e4c1340

Author:  vikas0296 <91872517+vikas0296@users.noreply.github.com>

Date:  Mon Jan 3 23:45:33 2022 +0100

Updated
```

14 Milestones

The following table contains the proposed milestones and their status.

Activities	**Status**
Implementation of XFEM	Yes
Implementation of Fracture Mechanics	Yes
Analysis of stress for mode I, mode II and mixed mode	Yes
Unit tests and Patch tests	Yes
Flow charts	Yes
Git commits log	Yes
Time analysis is performed for all test functions and a log file is generated	Yes
Implementation of Damage Mechanics	No
Crack propagation	No

Table 3: Table showing the Milestones status

14.1 Advantages of the program

∗ The program is well structured.

∗ The program is well documented.

∗ The program runs quickly.

∗ Unit and Patch tests have been conducted to check the sanity of the code.

∗ Time analysis for all the main functions has been done .

∗ Most of the milestones has been achieved.

14.2 Disadvantages of the program

∗ The program solves only plane stress problems.

∗ The program is capable of analysing only the straight edge crack.

∗ The program cannot handle more than one crack.

∗ The crack doesn't propagate.

∗ The program contains some redundant codes.

∗ The results are unreliable.

∗ The program is not user friendly.

∗ Not many examples have been solved.

15 Conclusion

In this project, extended finite element method and fracture mechanics have been coupled to analyse the stress field around the crack tip. One of the major advantages of the XFEM which has been exploited here. Using a course mesh, stress analysis of a cracked component has been done. Using displacement boundary conditions 2 examples are solved and another example is solved using traction boundary conditions. Stress intensity factors and energy release rates have been recorded and discussed. It is observed that the stress redistribution is similar in all the examples as it is a plane stress case. These quantities depend on the applied load and the crack length. Unit tests and patch tests have been conducted to check the sanity of the code. It is evident that the results are not as expected because of numerous reasons one of them could be the number of Gauss points.

References

[1] Awais Ahmed and Ferdinando Auricchio. Extended finite element method (xfem)-modeling arbitrary discontinuities and failure analysis, 2009.

[2] Ted Belytschko and Tom Black. Elastic crack growth in finite elements with minimal remeshing. *International journal for numerical methods in engineering*, 45(5):601–620, 1999.

[3] Amir R Khoei. *Extended finite element method: theory and applications.* John Wiley & Sons, 2014.

[4] Meinhard Kuna. Finite elements in fracture mechanics. *Solid Mechanics and Its Applications*, 201:153–192, 2013.

[5] Soheil Mohammadi. *Extended finite element method: for fracture analysis of structures.* John Wiley & Sons, 2008.

[6] Toshio Nagashima, Youhei Omoto, and Shuichi Tani. Stress intensity factor analysis of interface cracks using x-fem. *International Journal for Numerical Methods in Engineering*, 56(8):1151–1173, 2003.

[7] VB Pandey, IV Singh, BK Mishra, S Ahmad, A Venugopal Rao, and Vikas Kumar. A new framework based on continuum damage mechanics and xfem. *Engineering Fracture Mechanics*, 206:172–200, 2019.

[8] N Sukumar and J-H Prévost. Modeling quasi-static crack growth with the extended finite element method part i: Computer implementation. *International journal of solids and structures*, 40(26):7513–7537, 2003.

[9] Hiroshi Tada, Paul C Paris, and George R Irwin. The stress analysis of cracks. *Handbook, Del Research Corporation*, 34, 1973.

[10] Robert L Taylor. Feap-a finite element analysis program, 2014.

Additional Information

All the files pertaining to the programming project can be found here:

https://github.com/vikas0296/Vikas$_6$6133/tree/main.

1. Tried to implement Damage Mechanics but was unsure of the results. To compensate this failure, fracture mechanics has been implemented.

2. As, there there was no loading history, the crack is not allowed to propagate.

3. All the plots are saved in the same directory.

4. While performing the time analysis 20 files are generated with extension 'file <.prof>'

5. A log file is generated after the execution of profiler.py file.