

Contents

List of Figures

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[?].

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 crack is allowed to propagate. Numerical experiments are provided to demonstrate the correctness and robustness of the implemented technique.

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[?].

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[?]. 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[?].

2.1 Advantages of XFEM over standard FEM

- Unlike FEM, this method captures the discontinuities in the displacements due to the presence of cracks accurately[?].
- This technique allows the entire crack to be represented independently of the mesh, and so re-meshing is not necessary to model crack growth[?].
- 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 $f_i(\mathbf{x})$ whose support is contained in a cloud and whose value sum to unity at each point \mathbf{x} in the solution domain as see. ?? [?].

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

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

$$\sum_{i=1}^N f_i(\mathbf{x})\psi(\mathbf{x}_i) = \psi(\mathbf{x}) \quad (2)$$

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 $\mathbf{u}(\mathbf{x})$ can be discretized over the domain of the problem [?] by taking $f_i(\mathbf{x}) \equiv N_i(\mathbf{x})$ as see. ??

$$\mathbf{u}(\mathbf{x}) = \sum_{i=1}^N N_i(\mathbf{x})\mathbf{u}_i \quad (3)$$

wherein N is the number of nodal points for each finite element and $N_i(\mathbf{x})$ 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 [?]. The essential feature is based on the multiplication of enrichment functions by nodal shape functions[?]. It should be noted that only certain nodes are selected for the enrichment and the FE approximation of enriched domain takes the form see. ??

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

Expanding the above equation leads to see. ??

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

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[?].

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[?].

In this project, Heaviside step function and Asymptotic branch functions have been used as the extrinsic enrichment functions for the displacement approximation[?] . 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 [?] 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 ??, then that element should be enriched using Heaviside step function[?].

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[?]. The element [13,14,20,19] which is lying exactly behind the crack tip containing element, will have mixed enrichment[?]. 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 ??, all the elements that are surrounding the crack containing elements will be considered as the blended elements[?]. 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 ??, 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.

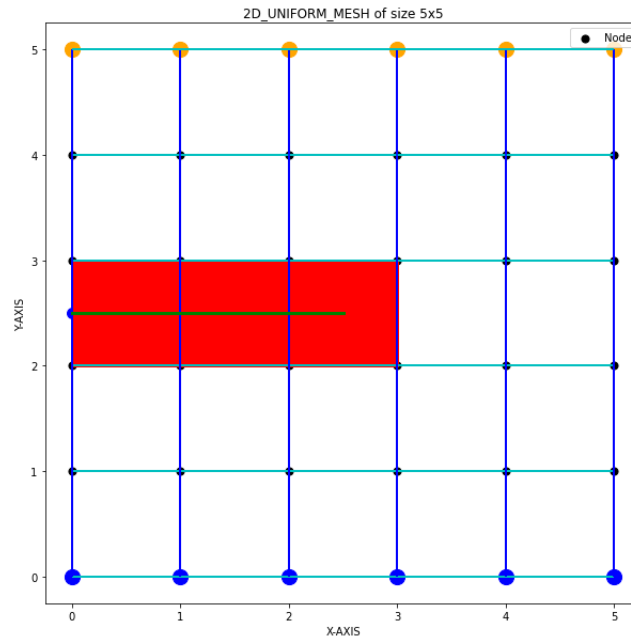


Figure 1: Elements selected for the enrichment

Blend.png

Figure 2: Blended-elements

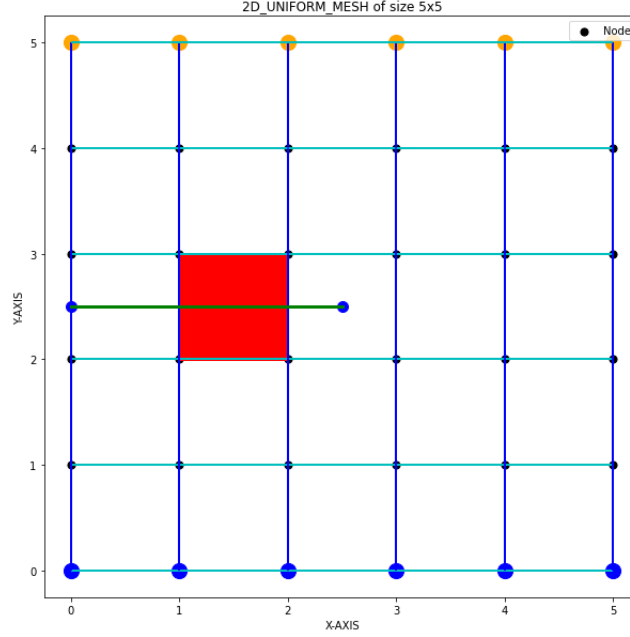


Figure 3: Mixed enriched element

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 ???. 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. ??. One of the easiest ways is the Orientation test[?].

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

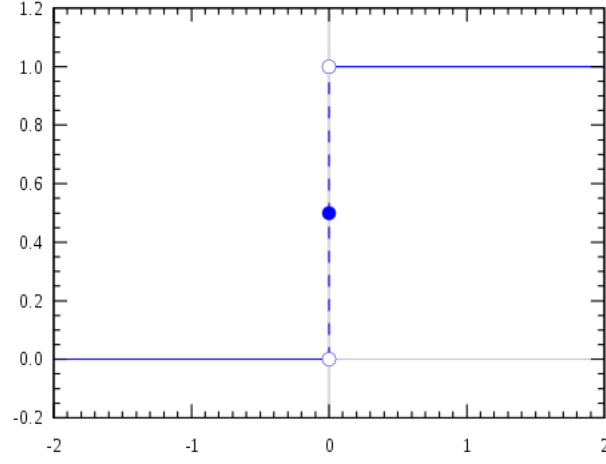


Figure 4: Heaviside step-function
[?]

$|\Delta|$, 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 ???. 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. ??

$$\Delta[?] = \begin{bmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{bmatrix} \quad (7)$$

In the above matrix see. ??, '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

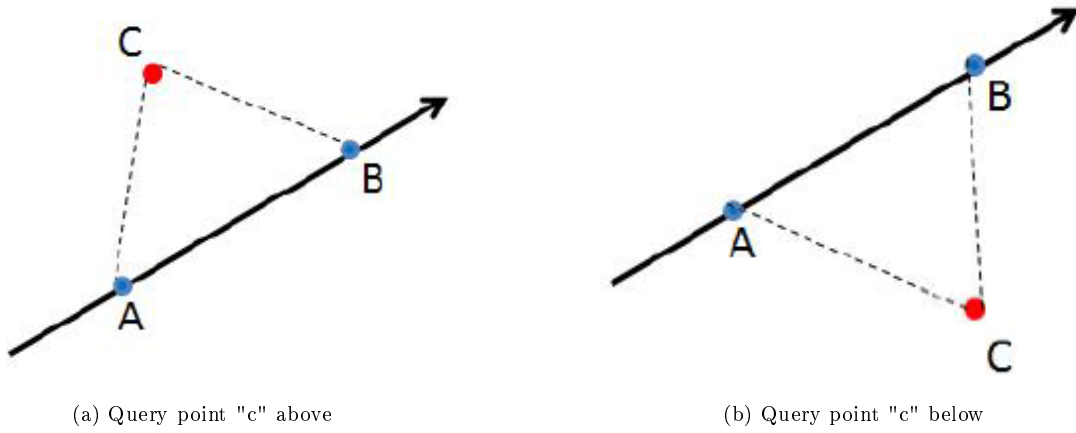


Figure 5: Orientation test
[?]

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 ?? which element has been chosen for the heaviside enrichment.

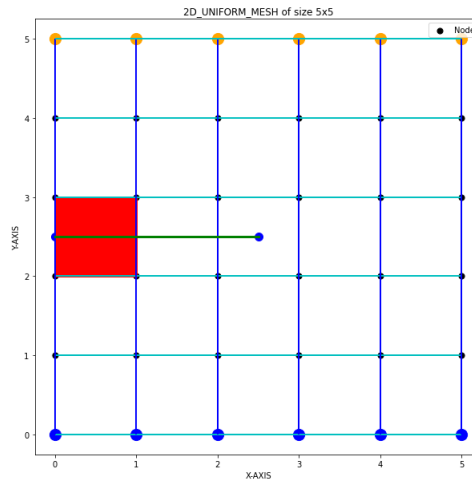


Figure 6: Heaviside enriched element

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. ???. The element containing crack tip marked in red is shown in ??.

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

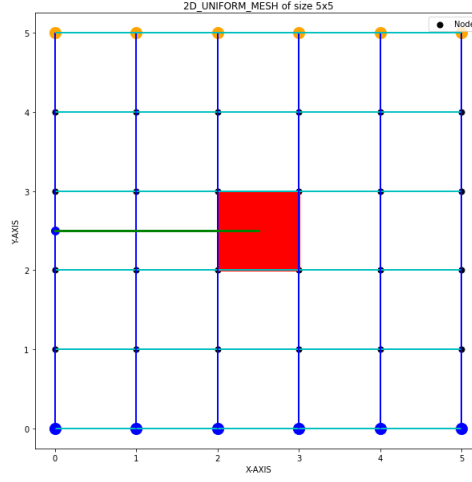


Figure 7: Tip enriched element

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. ??? 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.

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

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

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, $\beta_1 - 8\text{DOFs}$, $\beta_2 - 8\text{DOFs}$, $\beta_3 - 8\text{DOFs}$, $\beta_4 - 8\text{DOFs} = 40\text{DOFs}$)

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. ?? and eq (12) [?]

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

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

Derivatives of $F_\alpha(r, \theta)$ with respect to the crack tip polar coordinates (r, θ) is represented as see. ?? to see. ??

$$F_{1,r} = \frac{1}{2\sqrt{r}} \sin \frac{\theta}{2}, \quad F_{1,\theta} = \frac{\sqrt{r}}{2} \cos \frac{\theta}{2} \quad (13)$$

$$F_{2,r} = \frac{1}{2\sqrt{r}} \cos \frac{\theta}{2}, \quad F_{2,\theta} = -\frac{\sqrt{r}}{2} \sin \frac{\theta}{2} \quad (14)$$

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

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

and the derivatives of $F_\alpha(r, \theta)$ with respect to the local crack coordinate system (x', y') can then be defined as see. ?? to see. ??

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

$$F_{2,x'} = \frac{1}{2\sqrt{r}} \cos \frac{\theta}{2}, \quad F_{2,y'} = \frac{1}{2\sqrt{r}} \sin \frac{\theta}{2} \quad (18)$$

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

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

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

$$\frac{\partial r}{\partial X} = \frac{\partial r}{\partial x} \cdot \frac{\partial x}{\partial X} + \frac{\partial r}{\partial y} \cdot \frac{\partial y}{\partial X} \quad (21)$$

$$\frac{\partial r}{\partial Y} = \frac{\partial r}{\partial x} \cdot \frac{\partial x}{\partial Y} + \frac{\partial r}{\partial y} \cdot \frac{\partial y}{\partial Y} \quad (22)$$

$$\frac{\partial \theta}{\partial X} = \frac{\partial \theta}{\partial x} \cdot \frac{\partial x}{\partial X} + \frac{\partial \theta}{\partial y} \cdot \frac{\partial y}{\partial X} \quad (23)$$

$$\frac{\partial \theta}{\partial Y} = \frac{\partial \theta}{\partial x} \cdot \frac{\partial x}{\partial Y} + \frac{\partial \theta}{\partial y} \cdot \frac{\partial y}{\partial Y} \quad (24)$$

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

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

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

Using the transformation relationship between the global and crack tip coordinates we have ?? and ??

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

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

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

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

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

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

5.1 Shape functions

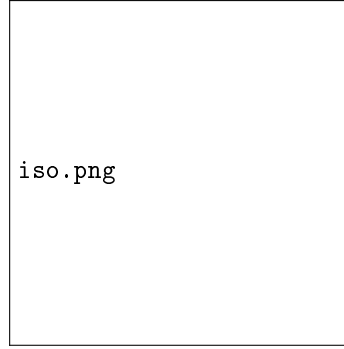


Figure 8: Iso Parametric Element Mapping
[?]

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 ??, the standard FEM bi-linear shape functions [?] associated with each node are given as see. ?? to eq (34).

$$N1 = (1 - \xi_1)(1 - \xi_2) \quad (31)$$

$$N2 = (1 + \xi_1)(1 - \xi_2) \quad (32)$$

$$N1 = (1 - \xi_1)(1 - \xi_2) \quad (33)$$

$$N2 = (1 + \xi_1)(1 - \xi_2) \quad (34)$$

The displacement approximation [?] can then be written in the form of see. ??

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

Where u represents displacements of the given element. Each node has 2 DOFs hence the displacement matrix has a shape of 8×1 . The standard FEM shape function matrix is given as see. ??

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

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

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

The shape functions are in local coordinate system and it should be transformed into global coordinate

system. This is accomplished by using Jacobian matrix [?] and the steps are as follows

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

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

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

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

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

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

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

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

- 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. ??.

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

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

5.2 B-matrix

This is also called as a Strain-Displacement matrix which gives a relation between strains and displacements see. [1].

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

Enriched B-matrix [2] is given by see. [2]

$$B_{enr} = \begin{bmatrix} (N_1 g),_x & 0 & (N_2 g),_x & 0 & (N_3 g),_x & 0 & (N_4 g),_x & 0 \\ 0 & (N_1 g),_y & 0 & (N_2 g),_y & 0 & (N_3 g),_y & 0 & (N_4 g),_y \\ (N_4 g),_y & (N_1 g),_x & (N_4 g),_y & (N_4 g),_x & (N_4 g),_y & (N_4 g),_x & (N_4 g),_y & (N_4 g),_x \end{bmatrix} \quad (46)$$

$g(x)$ could be either Heaviside step function or Near Tip function. The below [2] 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[2] is given by see. [2]

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

The derivative of Heaviside function is the Dirac delta function[2], that is see. [2]

$$(N_i H(x))_x = N_{i,x} H(x) = \delta \quad (48)$$

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

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

The derivative of near-tip enrichment function[?] is given by see. ??.

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

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

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

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

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

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

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

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

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

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[?].

The suggested approach is based on the increase of the number of Gauss integration points[?], as shown in the below Figure ??, 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[?]. 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. ??

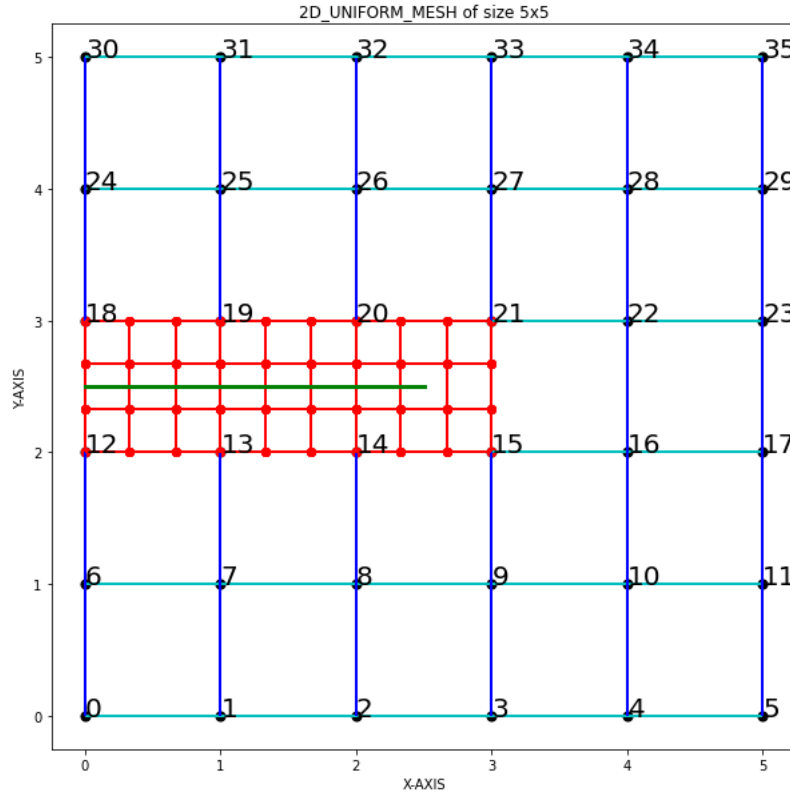


Figure 9: 9 Sub-polygons per element

$$\begin{aligned}
 K_{ij}^{\alpha\beta} &= \int_{\Omega} B_{i(\xi_1, \xi_2)}^T * D * B_{j(\xi_1, \xi_2)} * |J| d\xi_1 d\xi_2 \\
 &= \int_{\Omega+} B_{i(\xi_1, \xi_2)}^T * D * B_{j(\xi_1, \xi_2)} * |J| d\xi_1 d\xi_2 + \int_{\Omega-} B_{i(\xi_1, \xi_2)}^T * D * B_{j(\xi_1, \xi_2)} * |J| d\xi_1 d\xi_2 \\
 &= \sum_{l=1}^{\mathcal{N}^{\text{SUB}+}} \left(\sum_{k=1}^{\mathcal{N}^{\text{GP}+}} B_{i(\xi_{1k}, \xi_{2k})}^T * D * B_{j(\xi_{1k}, \xi_{2k})} * w_k \right)_l + \sum_{l=1}^{\mathcal{N}^{\text{SUB}-}} \left(\sum_{k=1}^{\mathcal{N}^{\text{GP}-}} B_{i(\xi_{1k}, \xi_{2k})}^T * D * B_{j(\xi_{1k}, \xi_{2k})} * w_k \right)_l \quad (57)
 \end{aligned}$$

where in $\mathcal{N}^{\text{SUB}+}$ and $\mathcal{N}^{\text{SUB}-}$ are the number of sub-polygons in $\Omega+$ and $\Omega-$, and $\mathcal{N}^{\text{GP}+}$ and $\mathcal{N}^{\text{GP}-}$ 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. ??

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

5.5 Element stiffness matrix

The element stiffness matrix for classical element is given by see. ?? and see. ??[?]

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

The BVP is solved using

$$[K][u] = [F] \quad (60)$$

Wherein

$K = K_{\text{global}}$, 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. ??.

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

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 [?] and it is given by see. ??

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

Transformation of displacement gradients to crack tip coordinate system see. ??

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

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. ?? and see. ??

$$\epsilon = [B][u] \quad (64)$$

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

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

Stresses are calculated using the see. ?? and plane stress relation see. ??

$$\sigma = [D][\epsilon] \quad (66)$$

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

The stresses, that are calculated using XFEM should be transformed from global coordinate system to

crack tip coordinate system using appropriate rotation matrix [?] and the equation is given by see. ??

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

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[?]. 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 [?]. 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 [?]

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 [?].

- Mode I: is the opening type ?? 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 ?? 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 ?. This is also called as Out-of-plane tearing mode.

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

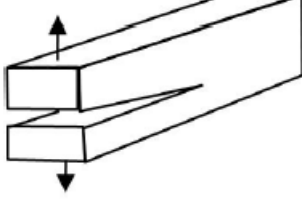


Figure 10: Mode-I Type failure [?]

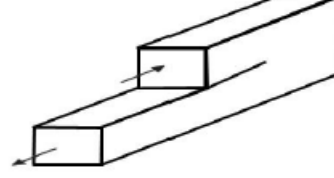


Figure 11: Mode-II Type failure [?]

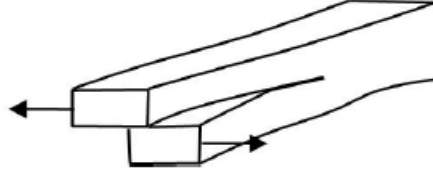


Figure 12: Mode-III Type failure [?]

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 ???. Williams (1957) used the Airy stress function to capture the singularity at the crack tip area using a polar coordinate system [?] (r, θ) as given in see. ??

$$\Phi = r^{\lambda+1} (c_1 \sin(\lambda + 1)\hat{\theta} + c_2 \cos(\lambda + 1)\hat{\theta} + c_3 \sin(\lambda - 1)\hat{\theta} + c_4 \cos(\lambda - 1)\hat{\theta}) \quad (69)$$

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

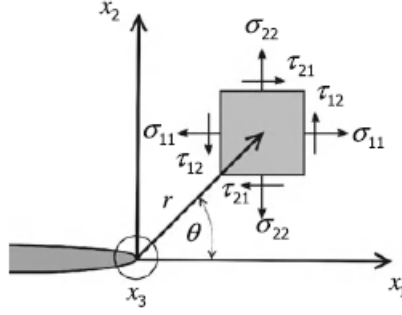


Figure 13: Stresses at the crack tip in Cartesian coordinates [?]

7.1 Auxiliary displacements for mode-I

Substituting the Airy stress function EQUATION in the equilibrium equation of the system, that is, $\nabla^2 \nabla^2 \Phi = 0$ [?] 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 [?]. This will be evaluated as see. ?? to see. ??

$$u_x = \frac{K_I(1+\nu)}{E} \sqrt{\frac{r}{2\pi}} \cos \frac{\theta}{2} \left[k - 1 + 2 \sin^2 \frac{\theta}{2} \right] \quad (70)$$

$$u_y = \frac{K_I(1+\nu)}{E} \sqrt{\frac{r}{2\pi}} \sin \frac{\theta}{2} \left[k + 1 - 2 \cos^2 \frac{\theta}{2} \right] \quad (71)$$

$$u_z = 0 \quad (72)$$

Derivatives of the displacement u_x and u_y w.r.t to x and y [?] is given in see. ?? to eq (76)

$$\frac{du_x}{dX} = \frac{du_x}{dr} * \frac{dr}{dX} + \frac{du_x}{d\theta} * \frac{d\theta}{dX} \quad (73)$$

$$\frac{du_x}{dY} = \frac{du_x}{dr} * \frac{dr}{dY} + \frac{du_x}{d\theta} * \frac{d\theta}{dY} \quad (74)$$

$$\frac{du_y}{dX} = \frac{du_y}{dr} * \frac{dr}{dX} + \frac{du_y}{d\theta} * \frac{d\theta}{dX} \quad (75)$$

$$\frac{du_y}{dY} = \frac{du_y}{dr} * \frac{dr}{dY} + \frac{du_y}{d\theta} * \frac{d\theta}{dY} \quad (76)$$

Derivatives of the displacement U_x and U_y w.r.t to r and θ [?] is given by see. ?? to see. ??

$$\frac{dU_x}{dr} = \frac{K_I(1+\nu)}{2D} \frac{1}{\sqrt{2\pi r}} \cos \frac{\theta}{2} (k - \cos \theta) \quad (77)$$

$$\frac{dU_x}{d\theta} = \frac{K_I(1+\nu)}{D} \sqrt{\frac{r}{2\pi}} \left[-0.5 \sin \frac{\theta}{2} (k - \cos \theta) + \sin \theta \cos \frac{\theta}{2} \right] \quad (78)$$

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

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

Derivatives r and θ w.r.t to x and y [?] is given by see. ?? and eq(82)

$$\frac{dr}{dX} = \cos \theta, \quad \frac{dr}{dY} = \sin \theta \quad (81)$$

$$\frac{d\theta}{dX} = -\sin \frac{\theta}{r}, \quad \frac{d\theta}{dY} = \cos \frac{\theta}{r} \quad (82)$$

7.2 Auxiliary stresses for mode-I

The Auxiliary stresses in case of mode I is given by ?? to ?? [?]

$$\sigma_x = \frac{K_I}{\sqrt{2\pi r}} \cos \frac{\theta}{2} \left[1 - \sin \frac{\theta}{2} \sin \frac{3\theta}{2} \right] \quad (83)$$

$$\sigma_y = \frac{K_I}{\sqrt{2\pi r}} \cos \frac{\theta}{2} \left[1 + \sin \frac{\theta}{2} \sin \frac{3\theta}{2} \right] \quad (84)$$

$$\tau_{xy} = \frac{K_I}{\sqrt{2\pi r}} \sin \frac{\theta}{2} \cos \frac{\theta}{2} \cos \frac{3\theta}{2} \quad (85)$$

$$\sigma_z = \begin{cases} \nu(\sigma_x + \sigma_y) & \text{Plane strain} \\ 0 & \text{Plane stress} \end{cases} \quad (86)$$

7.3 Auxiliary displacements for mode-II

The displacements for mode-II is given by ?? to ?? [?]

$$U_x = \frac{K_{II}(1+\nu)}{E} \sqrt{\frac{r}{2\pi}} \sin \frac{\theta}{2} \left[k + 1 + 2 \cos^2 \frac{\theta}{2} \right] \quad (87)$$

$$U_y = \frac{K_{II}(1+\nu)}{E} \sqrt{\frac{r}{2\pi}} \cos \frac{\theta}{2} \left[k - 1 - 2 \sin^2 \frac{\theta}{2} \right] \quad (88)$$

$$U_z = 0 \quad (89)$$

Derivatives of the displacement U_x and U_y w.r.t to r and θ [?] is given in ?? to ??

$$\frac{dU_x}{dr} = \frac{K_{II}(1+\nu)}{2D} \frac{1}{\sqrt{2\pi r}} \cos \frac{\theta}{2} (k + 2 + \cos \theta) \quad (90)$$

$$\frac{dU_x}{d\theta} = \frac{K_{II}(1+\nu)}{D} \sqrt{\frac{r}{2\pi}} \left[0.5 \cos \frac{\theta}{2} (k + 2 + \cos \theta) - \sin \theta \sin \frac{\theta}{2} \right] \quad (91)$$

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

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

7.4 Auxiliary stresses for mode-II

The Auxiliary stresses in case of mode II is given by ?? to ?? [?]

$$\sigma_x[?] = -\frac{K_{II}}{\sqrt{2\pi r}} \sin \frac{\theta}{2} \left[2 + \cos \frac{\theta}{2} \cos \frac{3\theta}{2} \right] \quad (94)$$

$$\sigma_y[?] = \frac{K_{II}}{\sqrt{2\pi r}} \sin \frac{\theta}{2} \cos \frac{\theta}{2} \cos \frac{3\theta}{2} \quad (95)$$

$$\tau_{xy}[?] = \frac{K_{II}}{\sqrt{2\pi r}} \cos \frac{\theta}{2} \left[1 - \sin \frac{\theta}{2} \sin \frac{3\theta}{2} \right] \quad (96)$$

$$\sigma_z[?] = \begin{cases} \nu(\sigma_x + \sigma_y) & \text{Plane strain} \\ 0 & \text{Plane stress} \end{cases} \quad (97)$$

Wherein "k" in see. ?? 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 see. ?? and see. ??, K_I , and K_{II} are the SIFs for mode-I and mode-II respectively and they are given by ?? and ??.

$$K_I = \sigma_y \sqrt{2 * \pi r} \text{ MPa}\sqrt{\text{m}} \quad (98)$$

$$K_{II} = \sigma_{xy} \sqrt{2 * \pi r} \text{ MPa}\sqrt{\text{m}} \quad (99)$$

wherein

σ_x = Tensile Stress in MPa

σ_{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[?]. 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 (??) and see. ??, which can be computed over an area of the FE mesh refer

to the figure below ???. 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 [?], as shown in Figure 7.16.

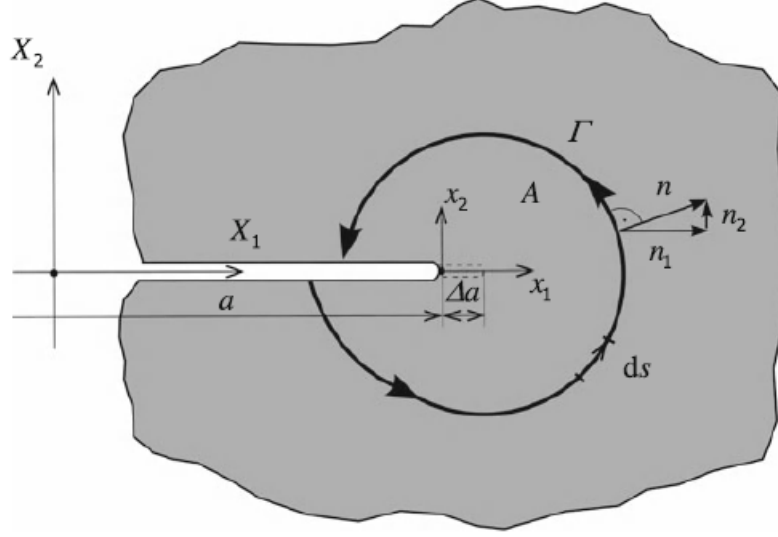


Figure 14: Elements inside the area are integrated[?]

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 the displacement, strain, and stress fields of state (1) that is, $\mathbf{u}_i^{(1)}$, $\epsilon_{ij}^{(1)}$, 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.

$$J = \int_A \left(\sigma_{ij} \frac{\partial u_i}{\partial x_1} - \mathcal{W} \delta_{ij} \right) \frac{\partial q}{\partial x_j} dA \quad (100)$$

$$I^{(1,2)} = \int_A \left(-\mathcal{W}^{(1,2)} \delta_{ij} + \sigma_{ij}^{(1)} \frac{\partial u_i^{(2)}}{\partial x_1} + \sigma_{ij}^{(2)} \frac{\partial u_i^{(1)}}{\partial x_1} \right) \frac{\partial q}{\partial x_j} dA \quad (101)$$

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

$$\mathcal{W}^{(1,2)} = \sigma_{ij}^{(1)} \epsilon_{ij}^{(2)} = \sigma_{ij}^{(2)} \epsilon_{ij}^{(1)} \quad (102)$$

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 [?] as see. ??

$$q = \sum_{I=1}^{N^{elem}} N_I(x) q_I \quad (103)$$

where 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. ??

$$\frac{\partial q}{\partial x} = \sum_{I=1}^{N^{elem}} \frac{\partial N_I}{\partial x} q_I \quad (104)$$

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

$$\left[\frac{\partial q}{\partial X} \right] = [R] * \left[\frac{\partial q}{\partial x} \right] \quad (105)$$

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

$$\begin{bmatrix} \frac{\partial q}{\partial X} \\ \frac{\partial q}{\partial Y} \end{bmatrix} = \begin{bmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{bmatrix} * \begin{bmatrix} \frac{\partial q}{\partial x} \\ \frac{\partial q}{\partial y} \end{bmatrix} \quad (106)$$

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

8.2 The Maximum Circumferential Tensile Stress Criterion

The maximum circumferential tensile stress theory was first presented by Erdogan and Sih (1963) based on the state of stress near the crack tip. Based on this theory, the crack propagates at the crack tip in a radial direction on the plane perpendicular to the direction of maximum tension [?].

In the current case, the hoop stress reaches its maximum value on the plane of zero shear stress. It is assumed that the size of plastic zone at the crack tip is negligible, the singular term solutions of stress at the crack tip can be used to determine the crack propagation angle, where the shear stress becomes zero [?].

The critical angle of crack propagation θ_c can be determined using ??

$$\frac{1}{\sqrt{2\pi r}} \cos \frac{\theta}{2} \left[\frac{1}{2} K_I \sin \theta + \frac{1}{2} K_{II} (3 \cos \theta - 1) \right] = 0 \quad (107)$$

The solution of this equation results in the crack propagation angle θ_c that can be expressed using the angle between the line of crack and the crack growth direction, with the positive value defined in an anti-clockwise direction, as [?] ??

$$\theta_c = 2 \arctan \frac{1}{4} \left(\frac{K_I}{K_{II}} \pm \sqrt{\left(\frac{K_I}{K_{II}} \right)^2 + 8} \right) \quad (108)$$

in which $K_{II} = 0$ results in a pure mode I condition, that is, $\theta_c = 0$. If $K_{II} > 0$, the crack growth direction $\theta_c < 0$, and if $K_{II} < 0$, the crack growth direction $\theta_c > 0$. An efficient expression of the critical angle of crack propagation can also be given as ??.

$$\theta_c = 2 \arctan \left[\frac{-2K_{II}/K_I}{1 + \sqrt{1 + 8 (K_{II}/K_I)^2}} \right] \quad (109)$$

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

dNdx = 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 is 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 element

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 theory

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 I crack

K1, K2 = SIFs for mode I and mode II

9.2 Necessary Files

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

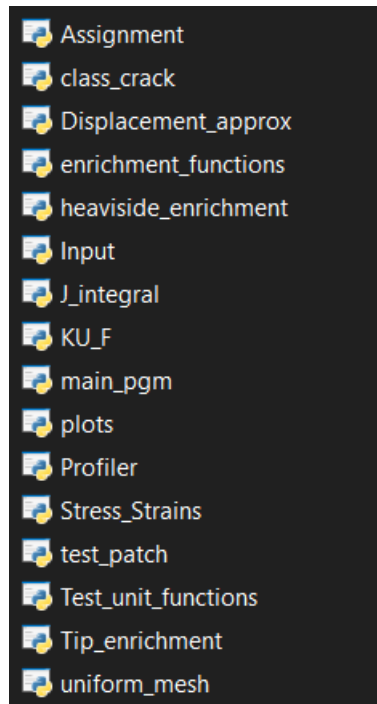


Figure 15: Files necessary to run the program

9.3 Libraries

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

numpy: 1.18.5

scipy: 1.5.0

matplotlib: 3.2.2

shapely: 1.8.0

9.4 Flow charts

The below illustrations ?? and ?? give a basic overview on the program flow.

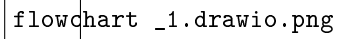
A small rectangular box containing the text "flowchart _1.drawio.png".

Figure 16: Flow chart for the selection of elements

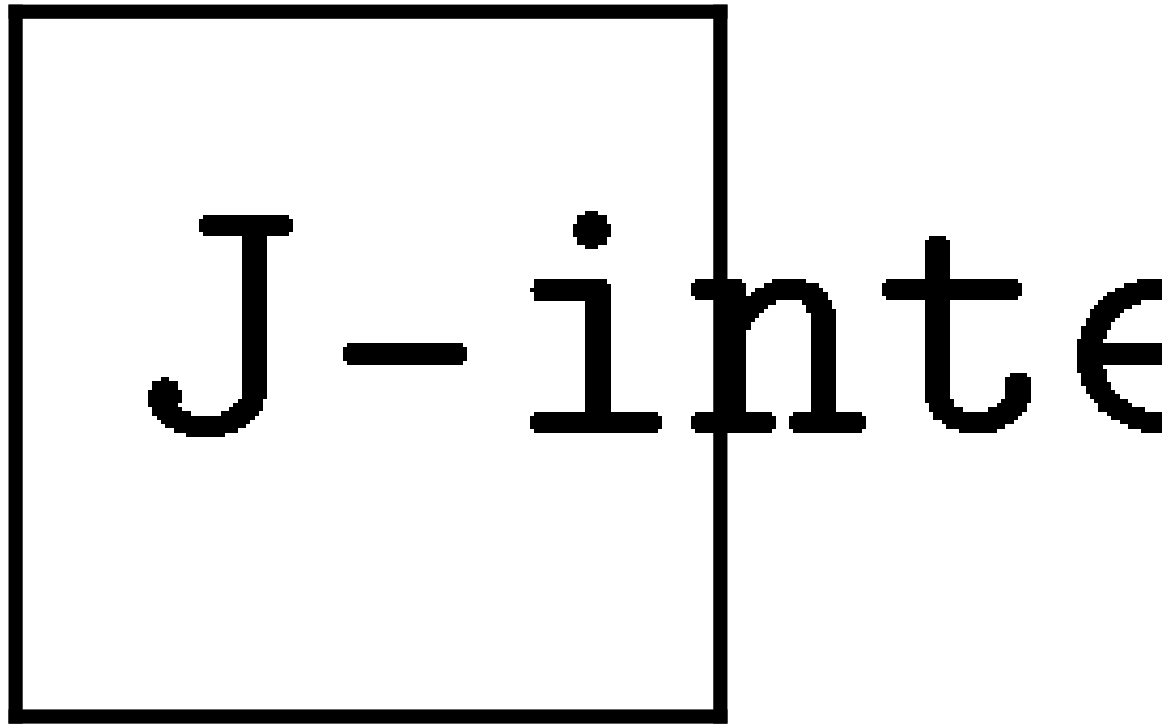


Figure 17: Flow chart calculate SIFs

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. ???. 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

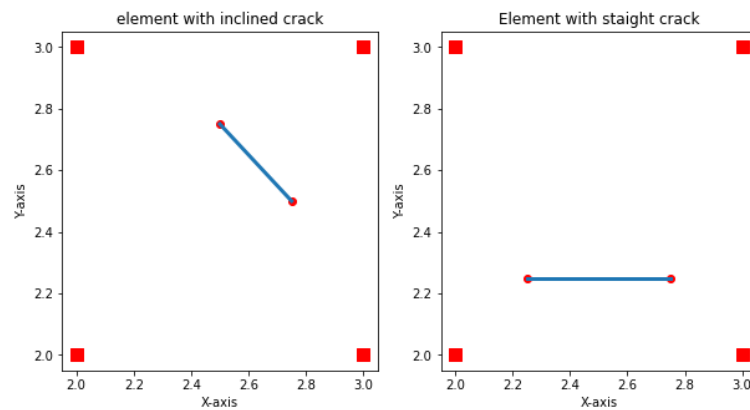


Figure 18: Crack segments inside an element
Left: inclined crack, Right : straight crack

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

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

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. ???. 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

```
epsilon_ij = 0.5*(U_i,j + U_j,i)
```

Parameters

grad_disps : Auxiliary displacements

Returns: Auxiliary strains

$$\epsilon_{ij} = 0.5 * \begin{bmatrix} u_{i,i} + u_{i,i} & u_{i,j} + u_{j,i} \\ u_{j,i} + u_{i,j} & u_{j,j} + u_{j,j} \end{bmatrix} \quad (110)$$

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. ??

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, $\alpha = 90^\circ$

random matrix = [1,2,3]

Rotation matrix =

$$\begin{bmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{bmatrix} \quad (111)$$

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.

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

Example: A geometry consisting of 2x2 elements with 9 nodes see. ??, 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.

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

Result: Test case passed

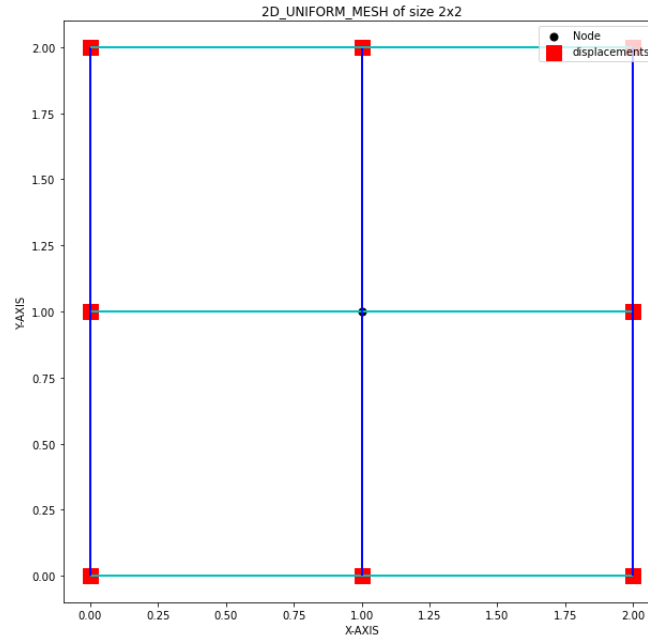


Figure 19: Sample under translation patch test
Red squares: Prescribed displacements

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 checks if the selected element is a Linear elastic or not. The easiest method is to implement Newton's method [?].

Aim: To check if the material response is 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 ??, 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.

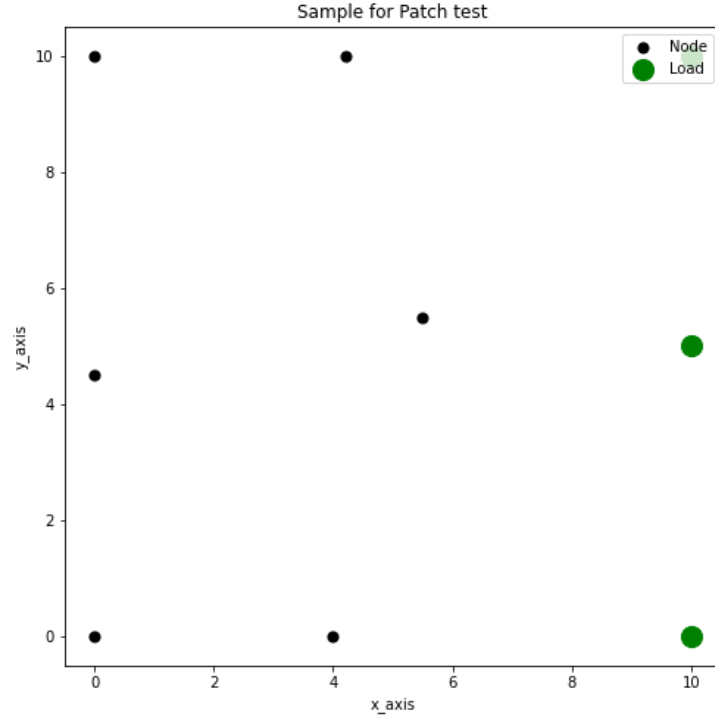


Figure 20: Sample under patch test
Green circles: Prescribed force

Consider the equilibrium equation see. ??

$$\mathbf{R}(\mathbf{u}) = \mathbf{F} - \mathbf{P}(\mathbf{u}) \quad (112)$$

wherein \mathbf{F} is the vector of applied nodal forces, \mathbf{u} is the vector of nodal displacements, and for a static linear elastic problem \mathbf{P} is defined as see. ??

$$\mathbf{P} = \mathbf{K}\mathbf{u} \quad (113)$$

in which \mathbf{K} is the global stiffness matrix and \mathbf{u} is the initial displacement vector. A solution for the see. ?? is defined by requiring the residuals \mathbf{R} to be zero.

For the first iteration prescribe displacements as zero and calculate vector \mathbf{P} and using the \mathbf{P} vector, calculate residuals \mathbf{R} using the see. ??. With the new residuals compute a new displacement vector and add the new displacement vector to the older vector see. ?? then evaluate the above equation but now the residuals will be a zero vector.

$$\mathbf{u} \leftarrow \mathbf{u} + \Delta \mathbf{u} \quad (114)$$

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[?].

Procedure: Here, a single element is considered for the test see. ??. 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.

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()'`

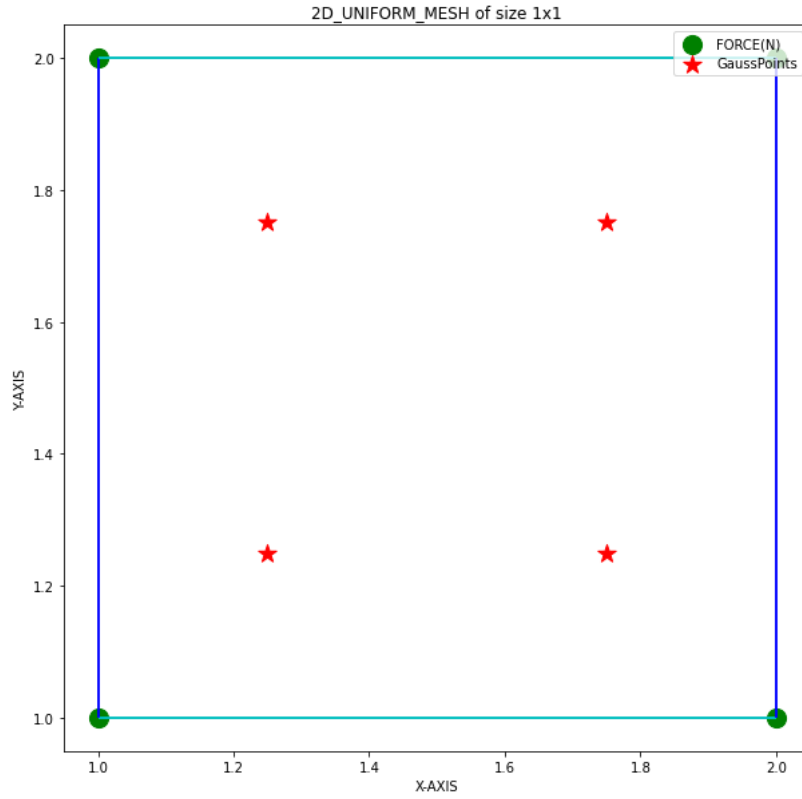


Figure 21: Stresses and strains for 1 element at 4 Gauss points

10.2.4 Sanity check for Shape functions

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

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:

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

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[?].

Procedure: For a given element coordinate system and Gauss points, a Jacobian matrix and its determinant is 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. [?].

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 [?].

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`

11 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 ?? gives you an idea of the time elapsed by each function.

Total time elapsed by all the functions is 54.875 CPU seconds

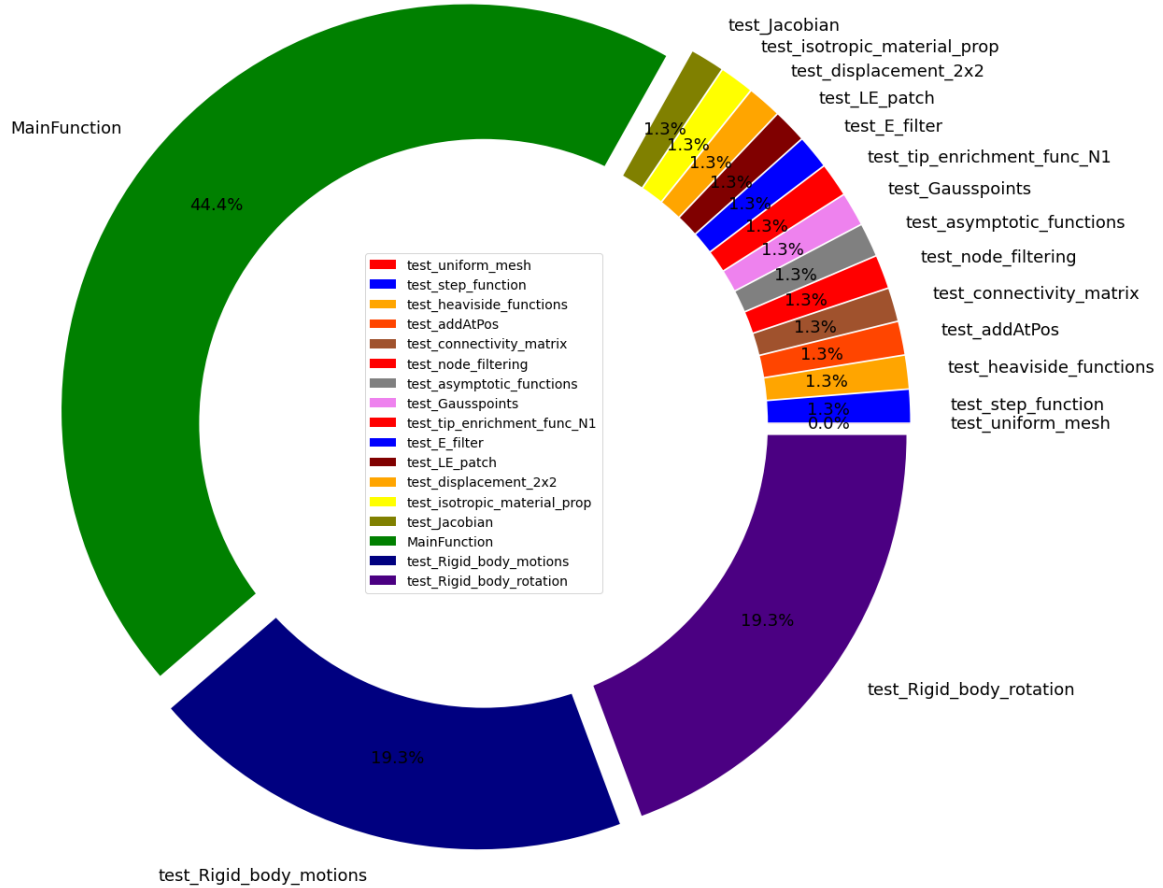


Figure 22: Time Analysis Pie Chart