August 2008

Crack simulation with eXtended Finite Element Methods

Presented by David NOËL

Supervised by Dr. Stéphane BORDAS

Student Research Internship

Contents

List of Figures

Introduction

The particularity of fracture simulation is that this involved discontinuities and interfaces. Moreover the stress and displacement fields have high gradients near the crack tip. In a classical Finite Element Method approach, the interface is modelised by different meshes and also the mesh is refine near the crack tip. This fine mesh evolves with the propagation of the crack (adaptive mesh) and has a huge impact on computational costs. The X-FEM idea (proposed by T. Belytschko in 1999 [5]) is to add special appropriate functions into the standard approximation space to take into account discontinuities and high gradiants. This is one of the most famous method for simulating cracks. Other methods : mesh-free method, adaptative remeshing, etc.

The main support of this internship was an experimental XFEM code for 2D elastic problems in MATLAB. After studying the fracture theory, several computations revealed that some steps of the computation should be improved. Thus, one main objective was to improve the accuracy of the computation by modifying the code. After that the code has been applied on several test cases (center crack under tension, offset cracks,etc.) and also extended on a new method called FleXFEM. In reality the main aim of this work was the obtaining of an accurate code in order to be able to compare in details XFEM and FleXFEM. This work will lead to a publication in the International Journal For Numerical Methods in Ingineering.

The reader will notice that this code is not to be used in real simulation for products conception. Two reason : first MATLAB is not really efficient for problems with a lot of degrees of freedom (dofs) and secondary because this code is "only" adapted for linear elastic fracture (LEFM) in 2D. Nevertheless, this code is and will be very useful for:

▶ beginner in fracture simulation ;

▶ having a good basis to treat more complex simulations (inhomogeneity, 3D, multiple cracks,etc) ;

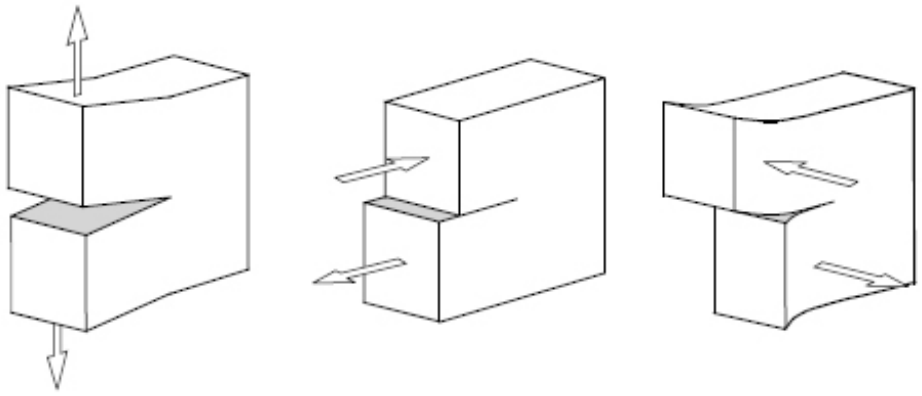
▶ studying basic ideas based on XFEM (like FlexFEM for instance as you will see below) because easier than c++ for example.

Crack simulation:theory

In this report it's considered that the reader has some basis in the Finite Element Method (FEM). Some reminder have been added in the appendix (7.1.1).

2.1 Linear elastic fracture

Deformation near the tip is separate between 3 independent load ways;



(a) mode I: opening mode (b) mode II: sliding or sharing mode (c) mode III: tearing mode

FIG. 2.1: Modes of crack tip deformation.

In Linear Elastic Fracture Mechanics (LEFM) the fields are linear combinations of every individual mode. Thus the stress σ, the strain ε and displacements $\underline{u}$ are given by analytical expressions for each mode (demonstration in [6] [7]). In this document only plan problems will be considered so only the mode I and II are important. See in Table 2.1 the analytical stress and displacement field corresponding to the polar parameters FIG. 2.2 in the global coordinate system.

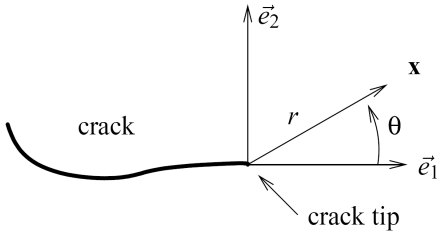


FIG. 2.2: Polar coordinate system associated to the crack tip [1].

Mode I	Mode II
$u_1(r,\theta) = \frac{K_I}{\mu}\sqrt{\frac{r}{2\pi}}\cos\frac{\theta}{2}\left(1-2\nu+\sin^2\frac{\theta}{2}\right)$	$u_1(r,\theta) = \frac{K_{II}}{\mu}\sqrt{\frac{r}{3\pi}}\sin\frac{\theta}{2}\left(2-2\nu+\cos^2\frac{\theta}{2}\right)$
$u_2(r,\theta) = \frac{K_I}{\mu}\sqrt{\frac{r}{2\pi}}\sin\frac{\theta}{2}\left(1-2\nu-\cos^2\frac{\theta}{2}\right)$	$u_2(r,\theta) = \frac{K_{II}}{\mu}\sqrt{\frac{r}{3\pi}}\cos\frac{\theta}{2}\left(-1+2\nu+\sin^2\frac{\theta}{2}\right)$
$\sigma_{11}(r,\theta) = \frac{K_I}{\sqrt{2\pi r}}\cos\frac{\theta}{2}\left(1-\sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right)$	$\sigma_{11}(r,\theta) = -\frac{K_{II}}{\sqrt{2\pi r}}\sin\frac{\theta}{2}\left(2-\cos\frac{\theta}{2}\cos\frac{3\theta}{2}\right)$
$\sigma_{22}(r,\theta) = \frac{K_I}{\sqrt{2\pi r}}\cos\frac{\theta}{2}\left(1+\sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right)$	$\sigma_{22}(r,\theta) = \frac{K_{II}}{\sqrt{2\pi r}}\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\frac{3\theta}{2}$
$\sigma_{33}(r,\theta) = \nu(\sigma_{11}+\sigma_{22})$	$\sigma_{33}(r,\theta) = \nu(\sigma_{11}+\sigma_{22})$
$\sigma_{12}(r,\theta) = \frac{K_I}{\sqrt{2\pi r}}\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\frac{3\theta}{2}$	$\sigma_{12}(r,\theta) = \frac{K_I}{\sqrt{2\pi r}}\cos\frac{\theta}{2}\left(1-\sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right)$

Table 2.1: Analtytical expression of the stress and diplacement fields for modes I and II

As one can see, these field are defined by some magnitude K_I and K_{II}. They are named Stress Intensity Factor (SIF) and there definition is :

$$K_I = \sqrt{2\pi r}\lim_{r\to 0}\sigma_{22}(r,0) \qquad K_{II} = \sqrt{2\pi r}\lim_{r\to 0}\sigma_{11}(r,0) \tag{2.1}$$

2.2 Enrichment of the approximation field

As discussed in the introduction, for increasing the rate of convergence in displacement and simulate the discontinuity, the approximation field is enriched. As the theoretical expressions are known for each mode the functions are choosen to help to reproduce the real displacement field. (The advantage on enriched methods will be discuss in 5.2.3) That's why in XFEM for fracture mechanics the approximate displacment field is given by the following expression [8]:

$$\underline{u}_h(\underline{x}) = \sum_{i\in N}\varphi_i(\underline{x})\underline{u}_i + \sum_{i\in N_d}\varphi_i(\underline{x})H(\underline{x})\underline{a}_i + \sum_{i\in N_p}\varphi_i(\underline{x})\left(\sum_{j=1}^{4}F_j(\underline{x})\underline{b}_i^j\right) \tag{2.2}$$

With

- N is the set of nodes of the mesh ;
- N_d is the set of nodes enriched by the discontinuity (in blue);
- N_p is the set of nodes selected to model the field near the tip (in red) ;
- $\underline{u_i}$ the classical degree of freedom for the node i ;
- $\underline{a_i}$ the extra degree of freedom for the discontinuity function ;
- $\underline{b_i^j}$ the extra degree of freedom for the branch function F_j ;
- φ_i the standard FE shape function associated to the node i ;
- H the Heaviside function ;
- F_j the branch function such as:

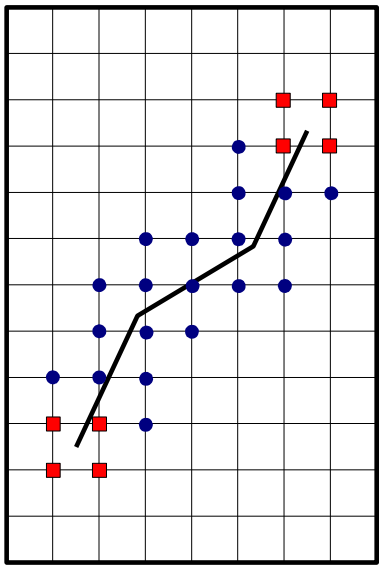


FIG. 2.3: Node selection for the enrichment [2].

$$F_j = \left\{ \sqrt{r}\sin\frac{\theta}{2}, \sqrt{r}\cos\frac{\theta}{2}, \sqrt{r}\sin\frac{\theta}{2}\sin\theta, \sqrt{r}\cos\frac{\theta}{2}\sin\theta \right\} \tag{2.3}$$

As you can verify each algebraical displacement expressions in the Table 2.1 are linear combinations of the functions F_i

Example:

$$u_1(r,\theta) = \frac{K_I}{\mu\sqrt{2\pi}}\Big[(1-2\nu)F_2 + F_3\Big] \tag{2.4}$$

Number of Degrees Of Freedom (dofs):

An enriched node won't have one dof on the contrary to a non-enriched . For instance if the node I belong to N_p it will have 5 degree of freedom. In fact each enrichment function has an additionnal dof that the resolution will determine. That's why in XFEM:

$$dofs = \text{size}(N) + \text{size}(N_d) + \text{size}(N_p)$$

2.3 Method to calculate the Stress Intensity Factors

2.3.1 J integral

Let's suppose the presence of a crack in the domain as described FIG. 2.4. The concept of the J integral has been invented by Cherepanov (1967) It represent the strain energy release rate. In 2D case, it is defined by EQN. (2.5):

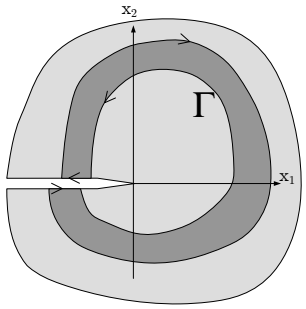


FIG. 2.4: Definition of the Jdomain.

$$J = \int_\Gamma \left[W dx_2 - T_i \frac{\partial u_i}{\partial x_1} d\Gamma \right] \qquad \text{with} \begin{cases} W = \sigma : \varepsilon / 2 \\ T_i = \sigma_{ij} n_j \end{cases} \tag{2.5}$$

J is independent of the path as demonstrated by Rice (1968)[9]. Rice also demonstrate the link between J and the SIF : in two dimensions :

$$J = \frac{1}{E'}(K_I^2 + K_{II}^2) \qquad \text{with } E' = \begin{cases} \dfrac{E}{1+\nu^2} & \text{for plan strain} \\ E & \text{for plan stress} \end{cases} \tag{2.6}$$

2.3.2 Auxiliary fields for extracting K_I and K_{II}

Although there is a link between (σ, ε) and (K_I, K_{II}) each SIF doesn't have a seperate expression. The following method allows the extraction of these SIFs.

This consists in adding some auxiliary fields and superimposing them to the actual field previously computed. If these field are suitably selected the SIF can be extracted.

▶ **Let's transform first the Jintegral into an indiciel form:**

$$J = \frac{1}{2}\int_\Gamma \left[\sigma_{ij}\varepsilon_{ij} dx_2 - 2\sigma_{ij} n_j \frac{\partial u_i}{\partial x_1} \right] = \frac{1}{2}\int_\Gamma \left[\sigma_{ij}\varepsilon_{ij}\delta_{2j} - 2\sigma_{ij}\frac{\partial u_i}{\partial x_1} \right] n_j d\Gamma \tag{2.7}$$

▶ **Let's consider two states of the cracked body:**

$$\begin{cases} (\sigma_{ij}^{(1)}, \varepsilon_{ij}^{(1)}, u_i^{(1)}) & \text{is the present state} \\ (\sigma_{ij}^{(2)}, \varepsilon_{ij}^{(2)}, u_i^{(2)}) & \text{is the auxiliary state} \end{cases}$$

▶ **Let $J^{(1+2)}$ be the J-integral for the sum of the two states:, from Eqn. (2.7):**

$$\begin{aligned} J^{(1+2)} &= \frac{1}{2}\int_\Gamma \left[(\sigma_{ij}^{(1)} + \sigma_{ij}^{(2)})(\varepsilon_{ij}^{(1)} + \varepsilon_{ij}^{(2)})\delta_{1j} - (\sigma_{ij}^{(1)} + \sigma_{ij}^{(1)})\frac{\partial (u_i^{(1)} + u_i^{(2)})}{\partial x_1} \right] \\ &= J^{(1)} + J^{(2)} + I^{(1+2)} \end{aligned} \tag{2.8}$$

with $I^{(1+2)}$ the interaction integral:

$$I^{(1+2)} = \frac{1}{2}\int_{\Gamma}\left[\sigma_{ij}^{(2)}\varepsilon_{ij}^{(1)}\delta_{1j} - 2\sigma_{ij}^{(1)}\frac{\partial u_i^{(2)}}{\partial x_1} - 2\sigma_{ij}^{(2)}\frac{\partial u_i^{(1)}}{\partial x_1}\right] n_j d\Gamma \qquad \text{since} \quad \sigma_{ij}^{(2)}\varepsilon_{ij}^{(1)} = \sigma_{ij}^{(1)}\varepsilon_{ij}^{(2)}$$

▶ **From Eqn. (2.6) :**

$$\begin{aligned} J^{(1+2)} &= \frac{1}{E'}\left[(K_I^{(1)} + K_I^{(2)})^2 + (K_{II}^{(1)} + K_{II}^{(2)})^2\right] \\ &= J^{(1)} + J^{(2)} + \frac{2}{E'}(K_I^{(1)}K_I^{(2)} + K_{II}^{(1)}K_{II}^{(2)}) \end{aligned} \tag{2.9}$$

▶ **Combining Eqn. (2.9) and Eqn. (2.8) we get:**

$$I^{(1+2)} = \frac{2}{E'}(K_I^{(1)}K_I^{(2)} + K_{II}^{(1)}K_{II}^{(2)}) \tag{2.10}$$

▶ **Solution**

Now let's choose particulate state 2 to apply the equation 2.10. See below:

$$\left\{\begin{array}{lcl} \text{state 2} & = & \text{pure Mode II asymptotic field} \\ \text{state 1} & = & \text{pure Mode I asymptotic field} \end{array}\right. \Longrightarrow \left\{\begin{array}{l} (K_I^{(2)}, K_{II}^{(2)}) = (1,0) \\ (K_I^{(2)}, K_{II}^{(2)}) = (0,1) \end{array}\right.$$

▶ **Finally:**

$$\boxed{\left\{\begin{array}{l} K_I^{(1)} = \dfrac{2}{E'}I^{(1,ModeI)} \\ \\ K_{II}^{(1)} = \dfrac{2}{E'}I^{(1,ModeII)} \end{array}\right.} \quad \text{with} \quad I^{(1+2)} = \frac{1}{2}\int_{\Gamma}\left[\sigma_{ij}^{(2)}\varepsilon_{ij}^{(1)}\delta_{1j} - 2\sigma_{ij}^{(1)}\frac{\partial u_i^{(2)}}{\partial x_1} - 2\sigma_{ij}^{(2)}\frac{\partial u_i^{(1)}}{\partial x_1}\right] n_j d\Gamma \tag{2.11}$$

2.4 Crack growth

As soon as there is a crack in a structure, the propagation of this one has to be simulate. In all our method the problem is quasi-static. The propagation is proceeded step by step. So at each step, the direction of the propagation has to be determined. Here are major criterias which are used to determine the direction:

- the maximum energy release rate criterion ;
- the maximum circumferential stress (hoop stress) criterion or the maximum principal stress criterion ;
- the minimum strain energy density criterion ;
- the zero K_{II} criterion.

The hoop stress criterion will be detailed below. In fact this the most frequently used because of its simplicity and accuracy. The propagation direction is computed such that in the direction $\theta = \theta_c$ the circumferential stress $\sigma_{\theta\theta}$ is maximum. Remark: the complete demonstration is in Appendix 7.1.2.

By using (e_r, e_θ) as reference, the expression of $\sigma_{\theta\theta}$ given in TABLE 2.1 become:

$$\sigma_{\theta\theta} = \frac{1}{4}\frac{K_I}{\sqrt{2\pi r}}\left[3\cos\frac{\theta}{2} + \cos\frac{3\theta}{2}\right] + \frac{1}{4}\frac{K_{II}}{\sqrt{2\pi r}}\left[-3\sin\frac{\theta}{2} - 3\sin\frac{3\theta}{2}\right] \tag{2.12}$$

Abstract of the calculation steps detailed in Appendix 7.1.2. :

- development of $\dfrac{\partial\sigma_{\theta\theta}}{\partial\theta} = 0$;
- trigonometrical manipulations ;
- transformation using $t = \tan\dfrac{\theta}{2}$;
- resolution of the quadratic equation.

And finally:

$$\theta_c = 2\arctan\left[\frac{-2K_{II}/K_I}{1+\sqrt{1+8(K_{II}/K_I)^2}}\right] \tag{2.13}$$

Numerical implementation

3.1 Structure of our XFEM code

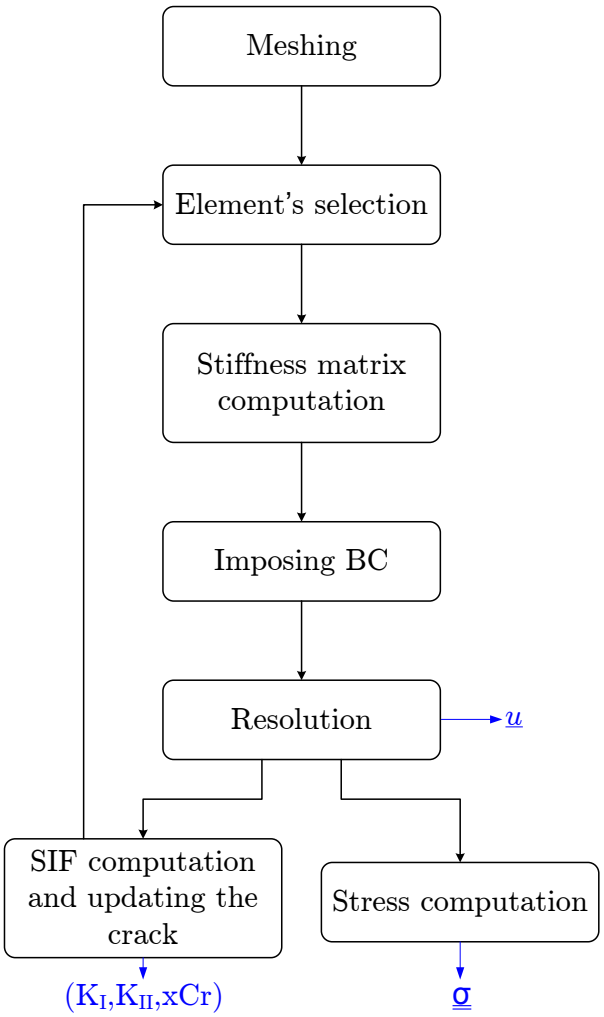


FIG. 3.1: Structure of the XFEM code.

with BC the Boundary Conditions i.e. the imposed displacements and/or imposed stress, $\underline{u}$ the strain field, K_I and K_{II} the Stress Intensity Factors (SIF), xCr the crack path (stored in an *structure* in MATLAB) and σ the stress field .

3.2 Elements and nodes selection

This part is based on the work of Sukumar and Prévost [10] who have presented a efficient way to select nodes for enrichment. In the code, the selection is a bit different because the function node-detect can take into account several cracks and gives also the coordinates of the intersection between the crack and the mesh, and the type of element.

3.2.1 Definition

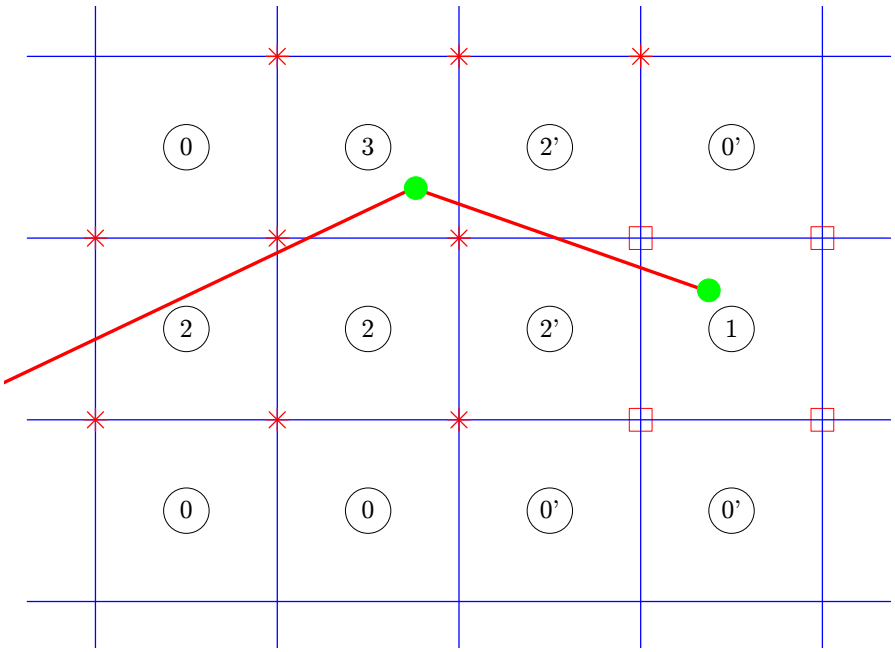


FIG. 3.2: Node and element's classification.

Nodes: The type of enrichment is stored in a array called `enrich_node`. For a node n,

- $enrich\_node(n) = 0$ for no enriched nodes ;
- $enrich\_node(n) = 1$ for nodes enriched by H (red cross) ;
- $enrich\_node(n) = 2$ for nodes enriched by branch functions (red square).

Elements: The type of element is stored in an array called `type_elem`. For the crack number k and the element e:

- $type\_elem(e, k) = 0$ for a normal element ;
- $type\_elem(e, k) = 1$ for a tip-element ;
- $type\_elem(e, k) = 2$ for a split-element ;
- $type\_elem(e, k) = 3$ for a vertex-element.

A *blending-element* will be an element which don't contain the tip but have some of his node enriched by branch functions (n' in the FIG. 3.2).

3.2.2 Element's selection

The type of element is determined by the opposite routine. In this routine the place of the intersection between element's boundaries is also stored.

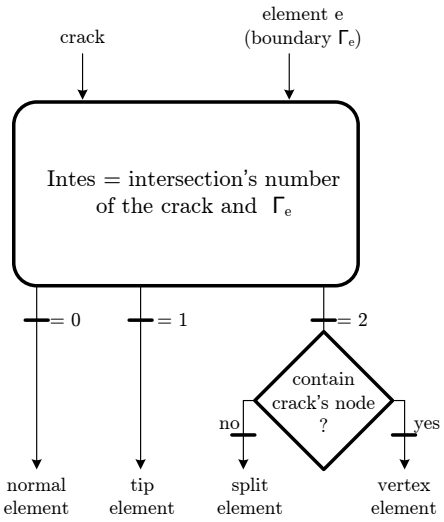


FIG. 3.3: Routine of the element classe's selection.

3.2.3 Node's type - support area

The selection of the enrichment is simple. Tip nodes are enriched with F_i and vertex and split nodes are enriched with the jump H IF the proportion of the considered support area of the node cut by the crack is bigger than the tolerance (usually 10^{-4}).
The support area's tolerance is really important. It prevents for bugs (due to bad condition number of the matrix) appearing when a node is too close to the crack.

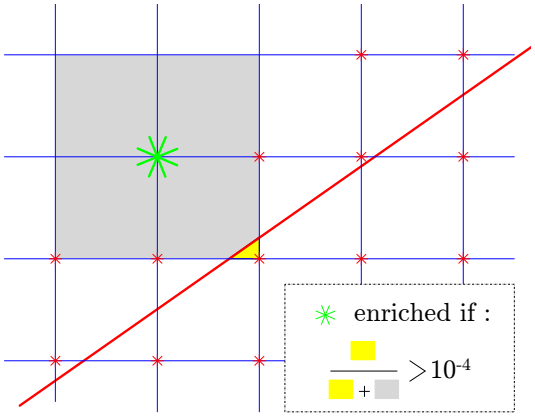


FIG. 3.4: Tolerance definition for selecting a node for the enrichment.

3.3 Enrichment of the approximation field - shifted function

At each enriched node, H or F_j is applied. If no precaution is taken, the code will need special post-processing steps to extract displacments at the nodes. Nevertheless if shifted function are used, the implementation of the rest of the code will be simpler. A shifted function is a enrichment function which take a zero value at the considered node (FIG. 3.5).

$$\forall S \in \{H, F_{1,\dots,4}\}, \forall \underline{x} \in \Omega_{support}, \qquad S_{node}(\underline{x}) = S(\underline{x}) - S(\underline{x}_{node}) \tag{3.1}$$

Remark : for simplify the notation, S_i is the shifted function of the node i ($S \in \{H, F_{1,...,4}\}$).

One-dimension example for a shifted heaviside enrichment:

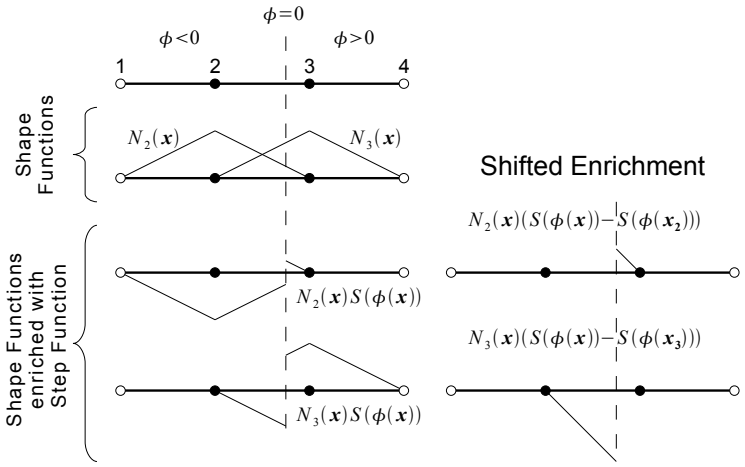


FIG. 3.5: Example of a 1D shifted function. The enrichment discribing the dicontinuity in the middle element is shifted and thus have no influence on the adjacent elements solution [3].

3.4 Construction of $Ku = f$

Let's consider the following figure to explain the computation of the stiffness matrix [11]:

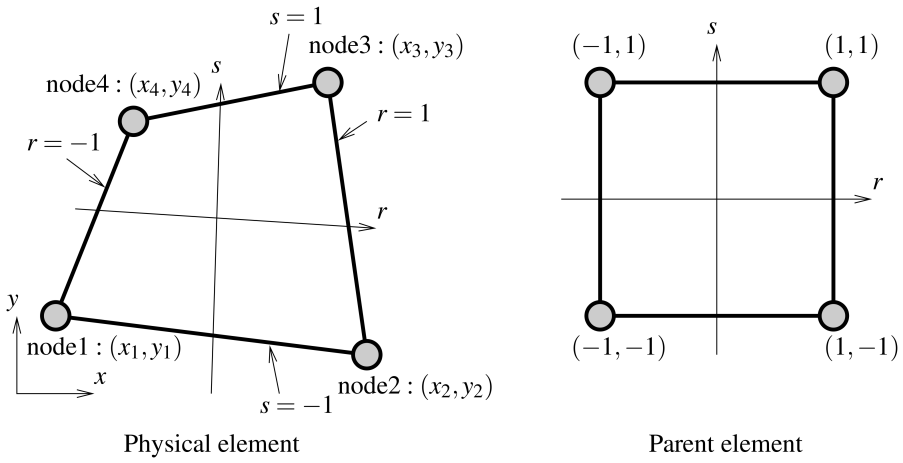


FIG. 3.6: Physical and parent 4-nodes elements [1].

3.4.1 Displacement implementation

The displacement field is given by: $\underline{u}^h = \underline{u}^{std} + \underline{u}^{enr}$ For an element e the discretized displacements are:

$$\underline{u}^e(\underline{x}) = \mathbf{N}^e(\underline{x})\underline{q}^e$$

Where $\underline{N} = [n_1\ N_2\ N_3\ N_4]^T$ is the vector of the shape function:

$$\mathbf{N}^e = \frac{1}{4}\begin{bmatrix} (1+r)(1+s) \\ (1-r)(1+s) \\ (1-r)(1-s) \\ (1+r)(1-s) \end{bmatrix} \tag{3.2}$$

See below an example of $\mathbf{N}^e\ \underline{q}^e$: considering that the field is enriched by only one function F (with $F_i = F(\underline{x}) - F(\underline{x_i})$ the function associated to the node i as discuss in 3.3).

$$\begin{aligned} \underline{q}^e &= [\ \underline{q}^e_{std}\ \ \underline{q}^e_{enr}\ ] \\ &= [u_{x_1}\ u_{x_2}\ u_{x_3}\ u_{x_4}\ u_{y_1}\ u_{y_2}\ u_{y_3}\ u_{y_4}\ a_{x_1}\ a_{x_2}\ a_{x_3}\ a_{x_4}\ a_{y_1}\ a_{y_2}\ a_{y_3}\ a_{y_4}] \end{aligned}$$

$$\begin{aligned} \mathbf{N}^e(\underline{x}) &= [\ \mathbf{N}^e_{std}(\underline{x})\ \ \mathbf{N}^e_{enr}(\underline{x})\ ] \\ &= \begin{bmatrix} N_1 & N_2 & N_3 & N_4 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & N_1 & N_2 & N_3 & N_4 & \end{bmatrix} \\ &\quad \begin{bmatrix} \cdots & N_1F_1 & N_2F_2 & N_3F_3 & N_4F_4 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & N_1F_1 & N_2F_2 & N_3F_3 & N_4F_4 \end{bmatrix} \end{aligned}$$

3.4.2 Stiffness Matrix

By definition an element stiffness matrix is given by:

$$\mathbf{K}^e = \int_{\Omega^e} \underline{\varepsilon}^T\ \mathcal{A}\ \underline{\varepsilon}\ d\Omega \qquad \text{where} \qquad \underline{\varepsilon}^e = \mathbf{D}.\underline{u}^e(\underline{x}) = \mathbf{D}\mathbf{N}^e(\underline{x})\underline{q}^e \tag{3.3}$$

Let $\mathbf{B}^e$ be the discetized gradient operator such as $\mathbf{B}^e = \mathbf{D}\mathbf{N}^e(\underline{x})$ and $\mathbf{B}^e = [\ \mathbf{B}^e_{std}\ \mathbf{B}^e_{enr}\ ]$

Thus:

$$\underline{\varepsilon}^e = \mathbf{B}^e(\underline{x})\underline{q}^e \tag{3.4}$$

With (in case of a 1-function enrichment F such as $F_i = F(x_i, y_i)$):

$$B = \begin{bmatrix} N_{1,x} & N_{2,x} & N_{3,x} & N_{4,x} & 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & N_{1,y} & N_{2,y} & N_{3,y} & N_{4,y} & \cdots \\ N_{1,y} & N_{2,y} & N_{3,y} & N_{4,y} & N_{1,x} & N_{2,x} & N_{3,x} & N_{4,x} & \end{bmatrix}$$
$$\begin{bmatrix} (N_1F_1)_{,x} & (N_2F_2)_{,x} & (N_3F_3)_{,x} & (N_4F_4)_{,x} & 0 & 0 & 0 & 0 \\ \cdots\ 0 & 0 & 0 & 0 & (N_1F_1)_{,y} & (N_2F_2)_{,y} & (N_3F_3)_{,y} & (N_4F_4)_{,y} \\ (N_1F_1)_{,y} & (N_2F_2)_{,y} & (N_3F_3)_{,y} & (N_4F_4)_{,y} & (N_1F_1)_{,x} & (N_2F_2)_{,x} & (N_3F_3)_{,x} & (N_4F_4)_{,x} \end{bmatrix}$$

So EQN. (3.3) become:

$$\mathbf{K}^e = \int_{\Omega_e} {}^t\mathbf{B}^e(\underline{x})\mathcal{A}\mathbf{B}^e(\underline{x})d\Omega \tag{3.5}$$

And also:

$$\mathbf{K}^e = \begin{bmatrix} \int_{\Omega^e} {}^t\mathbf{B}^e_{std}(\underline{x})\mathcal{A}\mathbf{B}^e_{std}(\underline{x})d\Omega & \int_{\Omega^e} {}^t\mathbf{B}^e_{std}(\underline{x})\mathcal{A}\mathbf{B}^e_{enr}(\underline{x})d\Omega \\ \int_{\Omega^e} {}^t\mathbf{B}^e_{enr}(\underline{x})\mathcal{A}\mathbf{B}^e_{std}(\underline{x})d\Omega & \int_{\Omega^e} {}^t\mathbf{B}^e_{enr}(\underline{x})\mathcal{A}\mathbf{B}^e_{enr}(\underline{x})d\Omega \end{bmatrix} \tag{3.6}$$

3.4.3 From the parent element to the physical element

The derivate of the shape function in the physical element $(N_{,x}, N_{,y})$ are needed for computing the stiffness matrix. But for now they are only given in the parent element $(N_{,r}, N_{,s})$ (See FIG. 3.6 for the notations.) :

$$\text{EQN.(3.2)} \Longrightarrow \qquad N_{,r} = \begin{bmatrix} 1+s \\ -1-s \\ -1+s \\ 1-s \end{bmatrix} \qquad \& \qquad N_{,s} = \begin{bmatrix} 1+r \\ 1-r \\ -1+r \\ -1-r \end{bmatrix}$$

Let's use the following fundamental relations:

$$N_{,x} = N_{,r}\frac{\partial r}{\partial x} + N_{,s}\frac{\partial s}{\partial x} \qquad\qquad N_{,y} = N_{,r}\frac{\partial r}{\partial y} + N_{,s}\frac{\partial s}{\partial y}$$

Oftenly the gradient of the transformation $\mathbf{F}$ is used:

$$\mathbf{F} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial s} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial s} \end{bmatrix} \qquad \text{so} \qquad [N_{,x}\ N_{,y}] = [N_{,r}\ N_{,s}].\mathbf{F^{-1}}$$

And for computing $\mathbf{F}$ (example with $\dfrac{\partial x}{\partial r}$)

$$x = \sum_{I=1}^{4} N_I x_I \qquad \Longrightarrow \qquad \frac{\partial x}{\partial r} = \sum_{I=1}^{4} \frac{\partial N_I}{\partial r} x_I$$

Remark: $\mathbf{F}$ is also very useful to change the limits of integrals:

$$\int_{\Omega^e} \_\_ \, d\Omega = \iint_{-1}^{1} \_\_ \, \det(\mathbf{F}) dr\ ds \tag{3.7}$$

Thus:

$$\mathbf{K}^e = \iint_{-1}^{1} {}^t\mathbf{B}^e(r,s).\mathcal{A}.\mathbf{B}^e(r,s) det(\mathbf{F}) dr\ ds \tag{3.8}$$

3.4.4 Integration in XFEM

The integration of EQN. (3.8) has to be transformed in order to be implemented. Like in classical FEM, we use Gauss Points (GP) and let EQN. (3.8) be a sum. The GP are

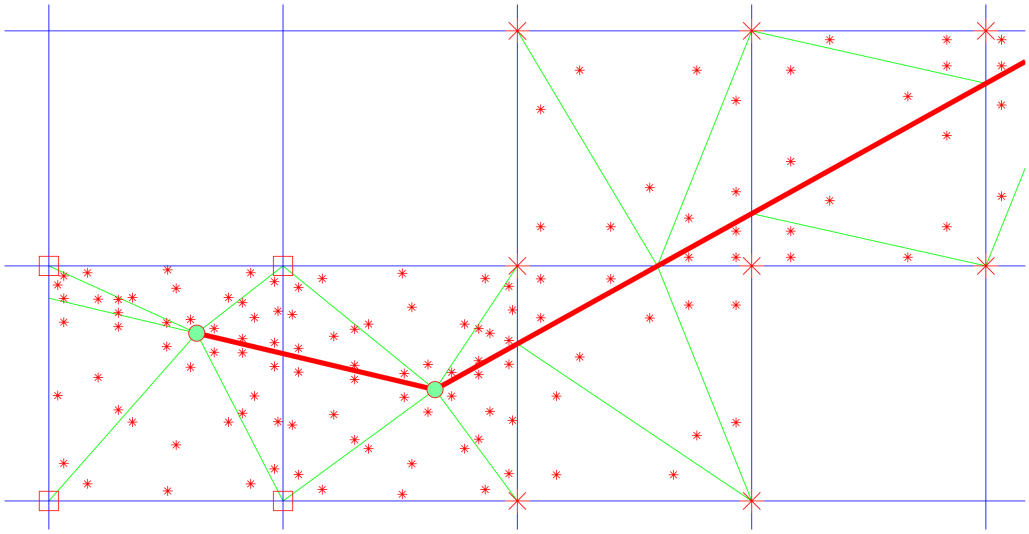


FIG. 3.7: Subtriangles and location of Gauss Points for the integration in XFEM (example with 3GP in each triangle and 4per quad).

caracterized by their position **in the parent element** x_g and their weight ω_g. Thus, the approximate integral is:

$$K^e = \sum_{g=1}^{G} \omega_g {}^t B^e(r,s) \mathcal{A} B^e(r,s) \det(\mathbf{F}) \tag{3.9}$$

Since the integration is performed on the parent element, the mapping described in FIG. 3.4.3 will transform it to the physical element thanks to EQN. (3.7). A subdivision is also perfomed in the elements containing the crack (FIG. 3.7). Indeed it is important to have no GP too close to the crack to be able to compute values of functions at these points.

The influence of the number of GP for the integration will be discuss further (in 4.1) since this represent a important part of this work.

3.5 The SIF computation

Caution: All the implementation of the SIF computation won't be detailed in the part because it was not the aim of this work. Only the main lignes will be explained.

3.5.1 Adaptation of 2.11

The EQN. (2.11) is a 1D inteagrale. For an easier implementation this equation is transformed to an surface integrale using the divergence theorem.

EQN. (2.11) become:

$$I^{(1+2)} = \frac{1}{2} \int_{Jdomain} \left[-\sigma_{ij}^{(2)} \varepsilon_{ij}^{(1)} \delta_{1j} + 2\sigma_{ij}^{(1)} \frac{\partial u_i^{(2)}}{\partial x_1} + 2\sigma_{ij}^{(2)} \frac{\partial u_i^{(1)}}{\partial x_1} \right] \frac{\partial q}{\partial x_j} dS \tag{3.10}$$

with $q(\underline{x})$ a smooth weighting function which takes the value of unity on an open set containing the crack tip (called the J domain)and vanished on an outer contour Γ as described in FIG. 2.4

3.5.2 Integration - J domain

The integral in EQN. (3.10) have to be implemented too. As for the integration performed to obtain the stiffness matrix, we use GP on the Jdomain. The elements selected for the Jdomain are at a distance of 3 element size as prescribed in [12].

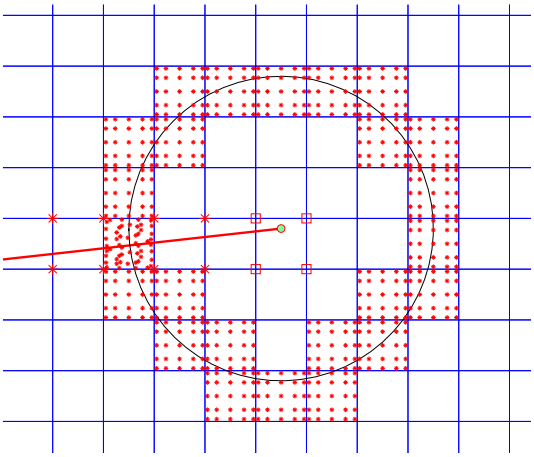


FIG. 3.8: Jdomain and location of the GP for the integration of EQN. (3.10).

3.6 The Flexible eXtended Finite Element Method (FleXFEM)

FleXFEM is a new method born in the university of Glasgow in 2008 thanks to the work of Stéphane BORDAS and his team. They published the first paper on the subject in march 2008 [13].

This method is the union of XFEM and Smooth Finite Element Method (SFEM) [14] [15], method using a technique based on the strain smoothing .

3.6.1 From surface integrals to boundary integrals

In FleXFEM we conside several subcells per element (nc). On each subcell the stress is constant over each smoothing cell, but discontinuous across cells. On the contrary, the displacement field is continuous within the element. The smoothing strain field $\widetilde{\varepsilon}$ at an arbitrary point $\underline{x}_C$ is defined exactly as for the standard SFEM:

$$\widetilde{\varepsilon}^h_{ij}(\underline{x}_C) = \int_\Omega \varepsilon^h_{ij}(\underline{x})\Phi(\underline{x} - \underline{x}_C)d\underline{x} \tag{3.11}$$

where Φ is a smoothing function defined exactly as in the SFEM

$$\Phi \geq 0 \quad \text{and} \quad \int_{\Omega} \Phi(\underline{\mathrm{x}})\mathrm{d}\underline{\mathrm{x}} = 1 \tag{3.12a}$$

$$\Phi\left(\underline{x} - \underline{x}_C\right) = \begin{cases} 1/A_C, \underline{x} \in \Omega_C \\ 0, \underline{x} \notin \Omega_C \end{cases} \tag{3.12b}$$

Similary as discribed in 3.4.2, the strain is written as

$$\widetilde{\varepsilon}^h(x_C) = \int_{\Omega} \mathbf{B}_{\text{xfem}} \underline{q} \Phi(\underline{x} - \underline{x}_C) d\Omega = \widetilde{\mathbf{B}}\ \underline{q} \quad \text{with} \quad \widetilde{\mathbf{B}} = \frac{1}{\mathrm{A_C}} \int_{\Omega_\mathrm{C}} \widetilde{\mathbf{B}}_{\text{xfem}}(\mathbf{x})\mathrm{dx} \tag{3.13}$$

Now, the integration have to be performed to get $\widetilde{\mathbf{B}}$. For a cell C, a node I and its shape function N_I and only one enrichment function F , $\widetilde{\mathbf{B}}_{CI}$ is given by :

$$\widetilde{\mathbf{B}}_{CI} = \left[\widetilde{\mathbf{B}}_{CI_{std}} \| \widetilde{\mathbf{B}}_{CI_{enr}}\right] = \frac{1}{A_C} \int_{\Omega_C} \left[\begin{array}{cc|cc} N_{I,x} & 0 & (N_I F_I)_{,x} & 0 \\ 0 & N_{I,y} & 0 & (N_I F_I)_{,y} \\ N_{I,y} & N_{I,x} & (N_I F_I)_{,y} & (N_I F_I)_{,x} \end{array}\right] d\Omega \tag{3.14}$$

And now by using the divergence theorem, the surface integral become a 1D integral: $\widetilde{\mathbf{B}}_{CI}$ become :

$$\widetilde{\mathbf{B}}_{CI} = \frac{1}{A_C} \int_{\Gamma_C} \left[\begin{array}{cc|cc} n_x N_I & 0 & n_x(N_I F_I) & 0 \\ 0 & n_y N_I & 0 & n_y(N_I F_I) \\ n_y N_I & n_x N_I & n_y(N_I F_I) & n_x(N_I F_I) \end{array}\right] d\Gamma \tag{3.15}$$

With $F_I = F_I(\underline{x}) = F(\underline{x}) - F(\underline{x}_I)$ (see section 3.3).

The assembly is similar to XFEM but considering several subcells n_C in the element, the elementary matrix become:

$$\mathbf{K}_e = \sum_{C=0}^{n_C} \int_{\Omega_C} \widetilde{\mathbf{B}}^T \mathcal{A} \widetilde{\mathbf{B}} d\Omega \tag{3.16}$$

3.6.2 Integration in FleXFEM

The integration is very simple in FleXFEM because its a 1D one. For one GP per edge (which in sufficient for most cases for the exact integration), EQN. (3.15) become : $\forall I \in [[1,4]], \forall C \in [[1,nc]]$,

$$\widetilde{\mathbf{B}}_{CI} = \frac{1}{A_C} \sum_{b=1}^{nb} \left[\begin{array}{cc|cc} n_x N_I(x_b^G) & 0 & n_x N_I(x_b^G) F_I(x_b^G) & 0 \\ 0 & n_y N_I(x_b^G) & 0 & n_y N_I(x_b^G) F_I(x_b^G) \\ n_y N_I(x_b^G) & n_x N_I(x_b^G) & n_y N_I(x_b^G) F_I(x_b^G) & n_x N_I(x_b^G) F_I(x_b^G) \end{array}\right] l_b^C \tag{3.17}$$

with x_b^G the center of the edge (GP) and and l_b^C the length of Γ_b^C. See FIG. 3.6.2, the

quadrature used for the integration in case of 1GP per edge.

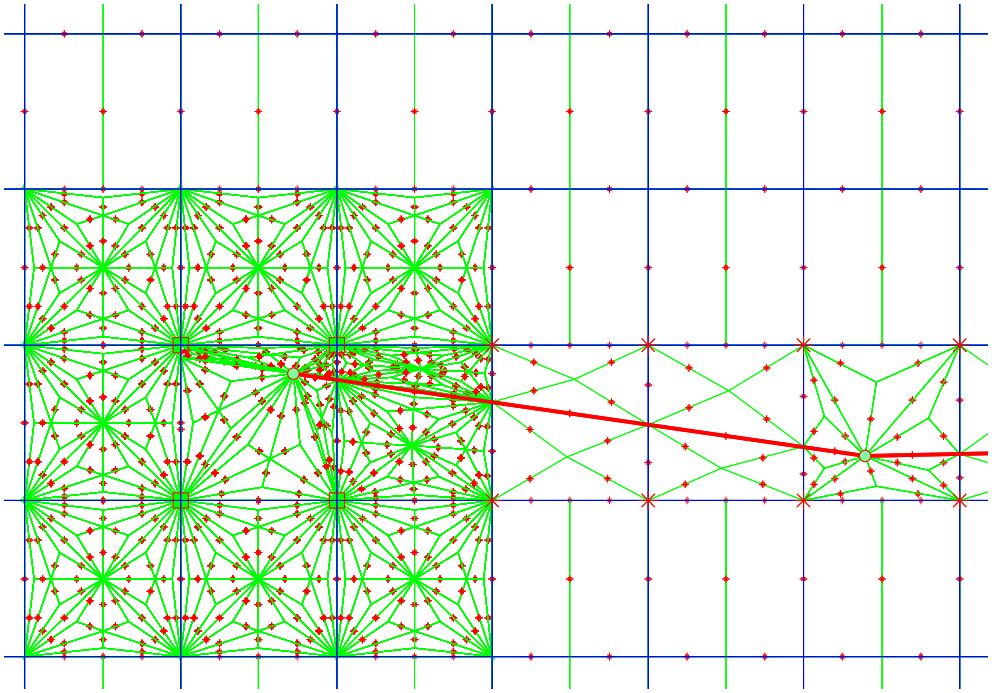


FIG. 3.9: Subcells and location of Gauss Points for the integration in FleXFEM.

EQN. (3.16) is also integrated (very simply because evrything is constant). Finally $\mathbf{K}_e$ is given by:

$$\mathbf{K}_e = \sum_{C=1}^{nc} \widetilde{\mathbf{B}}_C \mathcal{A} \widetilde{\mathbf{B}}_C A_C \quad \text{with } A_C \text{ the area of the subcell } \Omega_C. \tag{3.18}$$

3.6.3 Advantages

This method have some very interesting advantages :

► It decrease the complexity of integration because no more quadrature rules are needed and no mapping is done. This improve the speed of the computation.

► there is no more singular function to integrate because the $1/r$ term does not appear since the derivative of the shape function are not used.

► and other avantage details in [13] concerning the accuracy of the stress field, the accuracy of the SIF, the locking-free properties and also the non-sensibility to distorded meshes.

Improvement in the code

The main modifications made during this internship are explain below. They have been preceded by numerous computations which highlihted parts which needed to be improved.

4.1 Numeric integration for computing K

4.1.1 Order in normal elements

Why using the order 2? Because the functions in these elements are order one polynomial, only one point is needed. In fact, if G is the number of GP and n the order of the polynomial function to integrate, we need $G > \frac{n+1}{2}$

But unfortunately with only one GP in the quad, zero energy mode can appear (Fig. 4.1).

$$\int_{Q4} \varepsilon dx_1 dx_2 = \sum_{i=1}^{G} \omega_i \varepsilon_i(\underline{u}) = 0$$

Although the quad is deformed the integral is zero.
That why the order 2 is used for the integration in the normal elements.

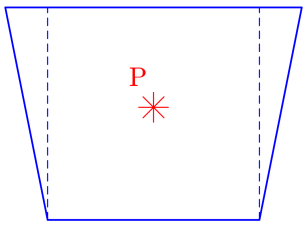


Fig. 4.1: Zero-energy mode.

Inconvenience: Theoretically, with higher order of integration the locking can appear but this is not a problem in our case.

We call locking when some displacements are blocked altought they should not. See Fig. 4.2, a simple exemple for the volumetric locking:
1- considering the triangle 1 for the point P: because of the incompressibility of 1, P can only move vertically
2- considering the triangle 2 for the point P: for the same reason, P can only move horizontally
⇒ P is blocked

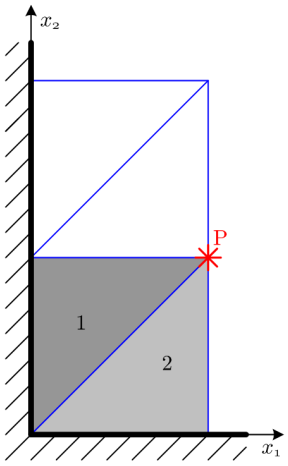


Fig. 4.2: Simple case of locking.

4.1.2 Integration in branch enriched elements

In tip and blending elements, the derivative of the shape function say $1/\sqrt{r}$ functions are integrated. This function is equal to an infinite polynomial sum (Taylor formula). Because $2m-1$ GP are needed to integrate a m-degree polynomial an infinite number of GP is required for he exact integration.

Nevertheless the more GP is used the more the Time Of Computation (TOC) is high. And in computing, having an infinity of GP will not give an exact result. In fact, a computation error is added at each elementary operation. Some results also showed that the accuracy was different on a 32-bits and 64-bits computers. This also proves that the precision of the machine have a real impact on the accuracy of the results.

First computation

One of the first computation made was the propagation for the center crack under tension. FIG. 4.3 show the path obtained.

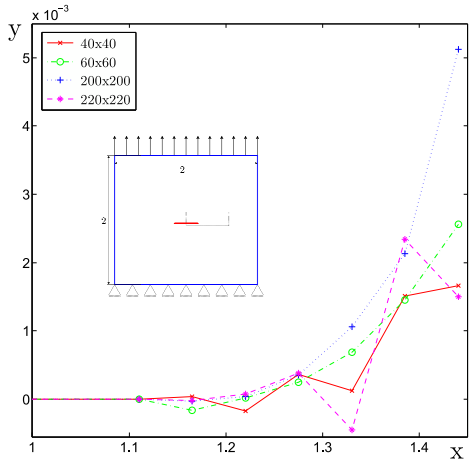


FIG. 4.3: First propagation result for the center crack under tension.

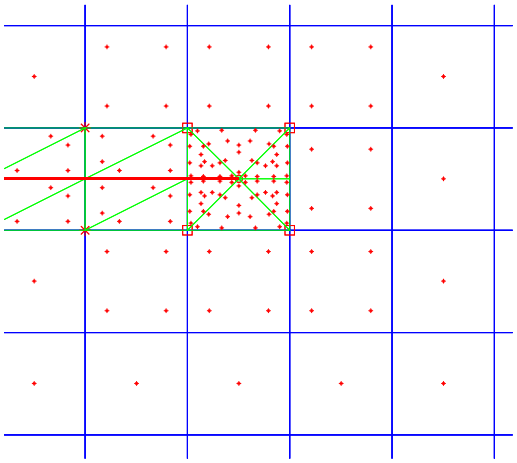


FIG. 4.4: Old GP mapping for the integration.

Interpretation:

One can notice on FIG. 4.4 that, the number near the tip element is not high. Furthermore the branch functions need to be integrated in blending elements too because these elements have some of their nodes enriched with branch functions. The number of GP have to be high in these elements too. Effect of this non sufficient integration accuracy: their is no convergence of the path whis refinement.

4.1.3 Tactless increase of GP

The increase a lot the number of GP was the first modification in the existing code. The old quadrature rule for triangle had a maximum of 13GP. A new rule called Dunavant (same as the author) have been include in the code. It allowed 121GP per triangle. The number of GP has been increased too in blending elements (subtriangulation + new quadrature rule).

See below the effect on the path for the exact same parameters as for FIG. 4.3 and the new GP mapping.

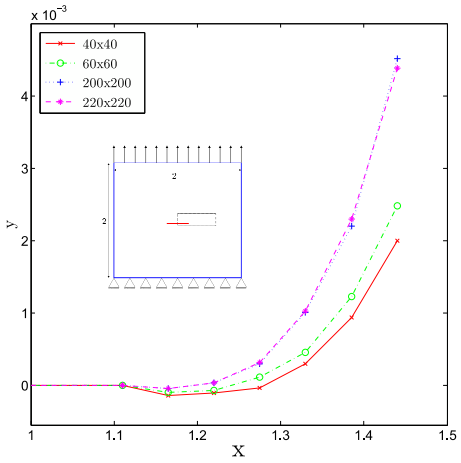


FIG. 4.5: Propagation for the center crack under tension after increasing the number of GP.

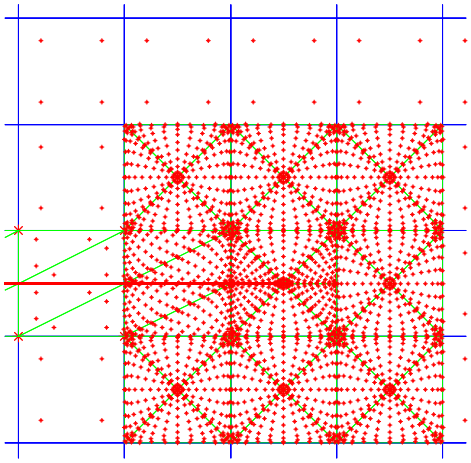


FIG. 4.6: Increase of GP in branch enriched elements with sub-triangulation in blending elements.

Conclusion:

In FIG. 4.5, the path seems better and most of all it convergs with mesh refinement. This result illustrates that increasing the number of GP improves the accuracy of the code.

Remark: After the implementation of this new quadrature rule other computations showed that this particular quadrature rule was less efficient than the previous one (relative error with the exact SIF). In fact there was other more efficient ways to increase the number of GP. That's why this rule has been abandonned. Nonetheless the conclusion of the influence of increasing the number of GP is still valide. The objective was to increase arbitrary the number of GP but still with using our rule which allow a maximum of 13 GP per triangle ; see below.

4.1.4 Increase of GP : the sub-triangulation solution

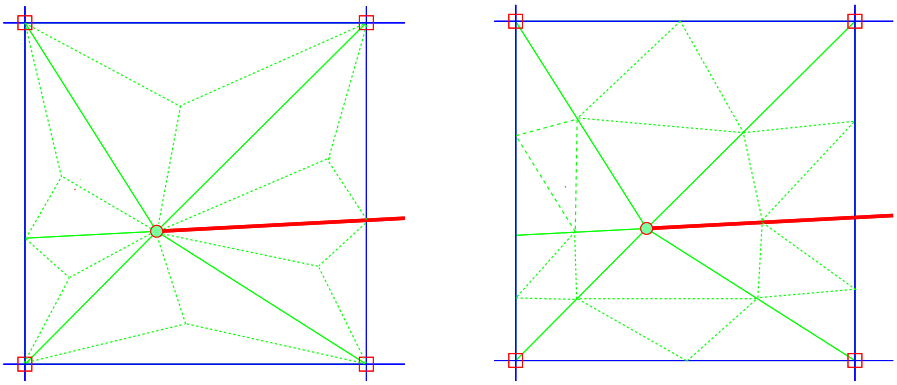
Making subdivisions is a good solution for increasing the number of GP in an element because it is a recursive routine.

Two methods of sub-triangulation have been created : by adding a new point in the middle of the triangle or by adding a new point in middle of each edge of the triangle.

The sub-division could be made with one or several rules. With a fixed number of subdivision's iteration and or when :

- the considered triangle have one of his node which is the tip (refinement of GP near the tip);
- the area of the considered triangle is bigger than a fixed fraction of the element.

Nevertheless these influence of 2 last conditions on the accuracy have not really been tested.



(a) with new point in the traingle (b) with new point on each edge

FIG. 4.7: Sub-triangulation methods.

The article [4] suggests also a efficient method to put GP near the tip with quad quadrature and a mapping to transform it into a triangle. (see FIG. 4.8).

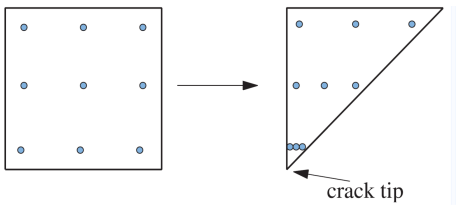


FIG. 4.8: Quadrature proposed by [4] to refine in GP near the tip.

4.1.5 Location test for improving the integration quality.

To try to test the efficiency of the integration, a test called *location test* has been made. With the Griffith problem (discribed in 5.2.1) the influence of the location of the tip element on the relative error to the analytique solution has been studied(see FIG. 4.9).

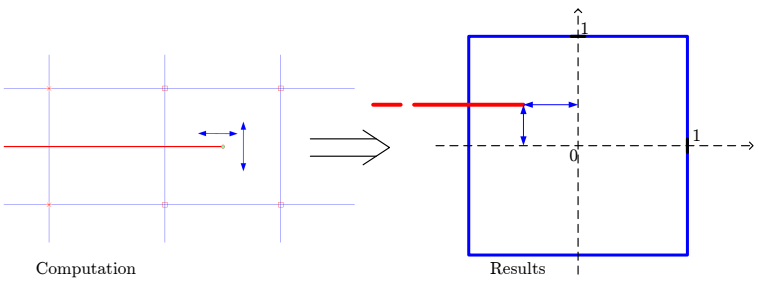


FIG. 4.9: Test the influence of the location of the tip in the element.

The first computation of this test has been made with the quadrature discribed in 4.1.3. This time of computation was high because the number of point on which you have to make a calculation of SIF is really important to have significant results. The relative error for the SIF K_I is plotted in FIG. 4.10. A 3D plot is also available in the appendix (FIG. 7.1) but is more adapted for visulising the relative error when you can rotate the 3D plot.

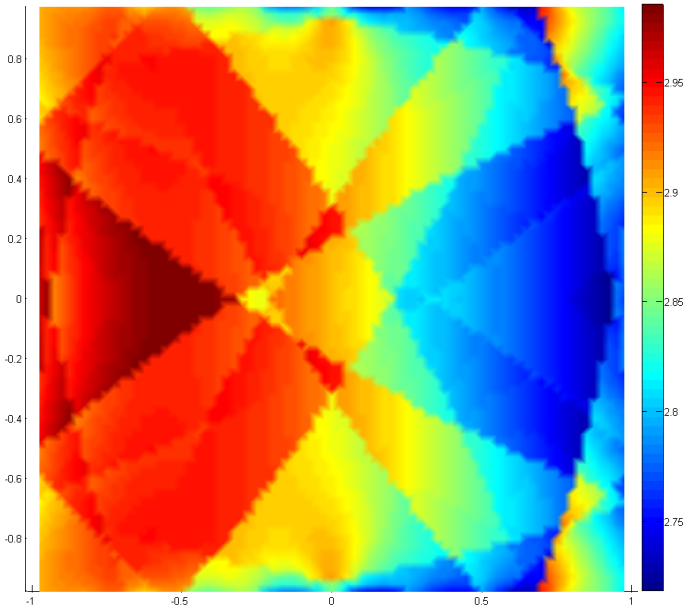


FIG. 4.10: Influence of the tip's location in the element on the relative error ER_{K_I}.

The first interpretation on this result was that the sub-division had a direct impact on the error by creating gaps in the mapping (clearly visible with the limit between different colors in FIG. 4.10). After a lot of computations with several integration rules (trying different sub-divisions types and number and also increasing the GP in the blending elements) have been made but unfortunalty no real influence were visible.

After some reflexion the conclusion was that this influence came from the integration of the auxilary terms in the SIF computation. In fact the location of the tip influence which element is in the Jdomain. As discussed before, the Jdomain have a huge influence on the computation of the SIF. Selecting or not an element can give different values of the SIF and that what explain the gaps in FIG. 4.10.

4.1.6 Displacement energy for choosing the number of GP

Principle:

We need to know precisely how much GP we should use to both have a fixed integration error and a reasonnable time of computation (concerning the integration in the branch-enriched elements). For that, the energy convergence with the Griffith case will be used. This case discribed in 5.2.1 will allow to study the exact link between the quadrature and the accuracy of the integration of the because the exact boundary condition are applied. For mesuring the accuracy of the integration, the following displacment error E_D is computed:

$$E_D = \sum_{I=1}^{N} \frac{||\underline{u}^2_{I,exact} - \underline{u}^2_{I,comp}||}{\underline{u}^2_{I,exact}} \tag{4.1}$$

An iteration has been made for each number of sub-triangulation $n_{sub} \in \{0, ..., 4\}$ for each order of quadrature $order \in \{1, ..., 7\}$ and for the 2 differents sub-triangulation types ($edge - sub$ and $tri - sub$ discribed in 4.7). The displacement error E_D is defined below by a sum on the nodes:

Results:

Unfortunatly nothing can be observe from E_D because the results are not significally. Same test have been made for the global energy and the strain energy error but the conclusion is similar. Nevertheless these computations showed the impact of the subtriangulations on the time of computation :

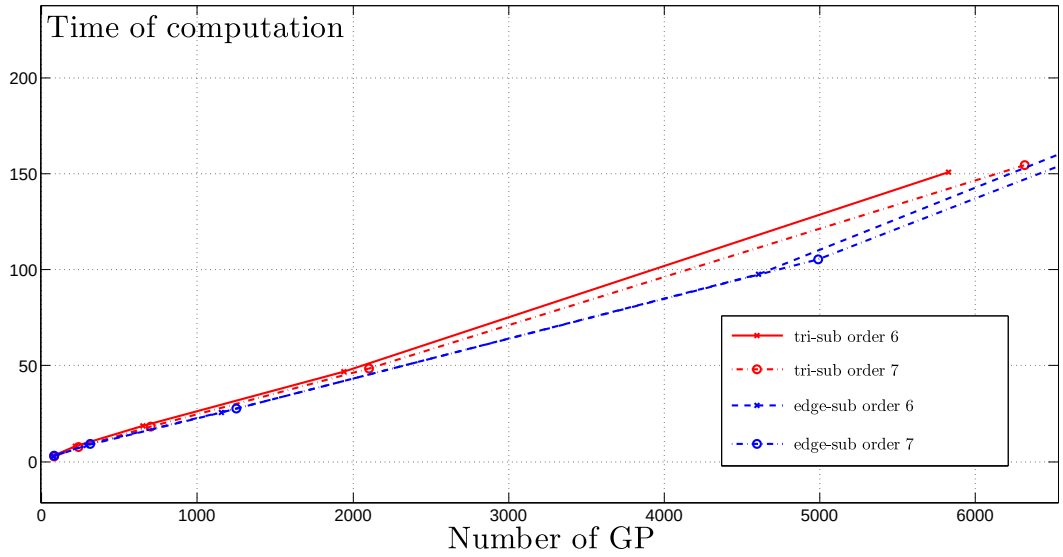


FIG. 4.11: Influence of the number of GP on the time of computation for several quadrature rules

Conclusion:

The sub-triangulation named edge-sub (FIG. 4.7(a)) will be picked up because

- for the same number of GP the TOC is shorter ;
- also because the condition number is always better with this sub-triangulation .

Because there is other computationnal errors introduced, in the SIF for instance, it's not worth it to use a huge number of GP for this integration. The increase of GP in blending elements will be sufficient. In conclusion, for having a limited integration error for the displacements we should use order 7 in the sub-triangles with one sub-triangulation type edge-sub done on the branch enriched elements.

4.2 Variable optimisation - modification of principal function

4.2.1 Variable optimisation

In the old version: the element class was stored in 2 different way: in an array (with numbers 1,2,3) and in 3 lists ([split_elem] [tip_elem] [vertex_elem]). This was not necessary since only the array is sufficient even with multiples cracks.

Confusion: A simplification was necessary to erase all confusion. In fact, a vertex and a split element had the same number in the type_elem array and some of the vertex element were select too to be in [split_elem] too. Caution: the number in type_elem and in enrich_nodes have no similarity. In fact a vertex element has the number 3 in type_elem and its nodes have the number 2 in node_detect (because it is enriched with the heaviside function).

New method: The 3 lists have been suppressed. To handle this change have been made in some function (SIF, gauss_rule, xfemBmatrix, and of course node_detect). These changments still allow multiple cracks since type_elem have a column for each crack.

Effect: These modifications will ease the understanding of the code by suppressing possible confusion and simplify the input and output off some functions.

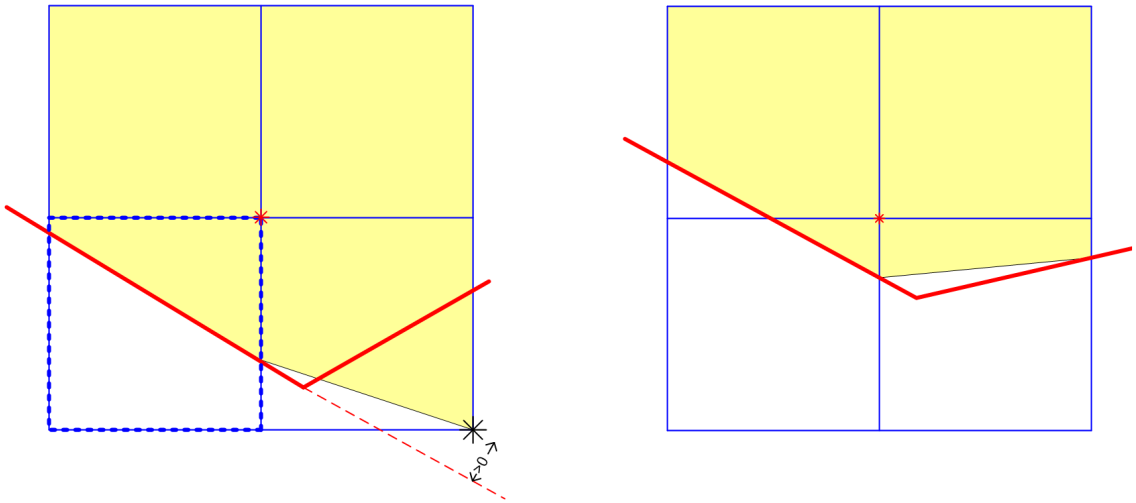
4.2.2 Vertex element in node_detect to compute the support area

Identification of the origine of the problem: In the previous version of the code, vertex elements were not properly taken into account. One could noticed it with non-symetrical results for symetric Boundary Conditions. That's why the function `node_detect.m` have been reorganized. But the main origin of imprecisions was the computation of the support area for vertex elements.

Computation of the support area for vertex elements All the calculation of the support area is based on computing the algebraic distance of the nodes from the nodes in function of the side of the crack. Unfortunately when there is vertex elements in the support the computation of this surface was wrong. Let's just see these following case:

Fig. 4.12(a): in red, the tested point, in yellow the area computed and in dots the reference element (for the description of the path). One can see that there was a problem with the selection of the side of the nodes (see the black one). This have been solved by adding a local definition of the path in each cut element.

Fig. 4.12(b): the computation of the surface of vertex is not that simple because it could be a non-convex area. To obtain the vertex area the surface is computed as if the element were a split element (in yellow) and depending on which side the vertex point, the surface of



(a) Problem of definition of the path for choosing the side of nodes.

(b) Computation of the non-convexe area.

FIG. 4.12: Computation of the support area for vertex elements.

the triangle is added or substracted.

Remark: level sets would also solve this kind of problems [16].

4.3 Locking problem

Description of the problem Some bug appeared when the crack was too close to a node. The node whose support area is really little was not enriched but we still had stress concentration near these points (FIG. 4.13). See below:

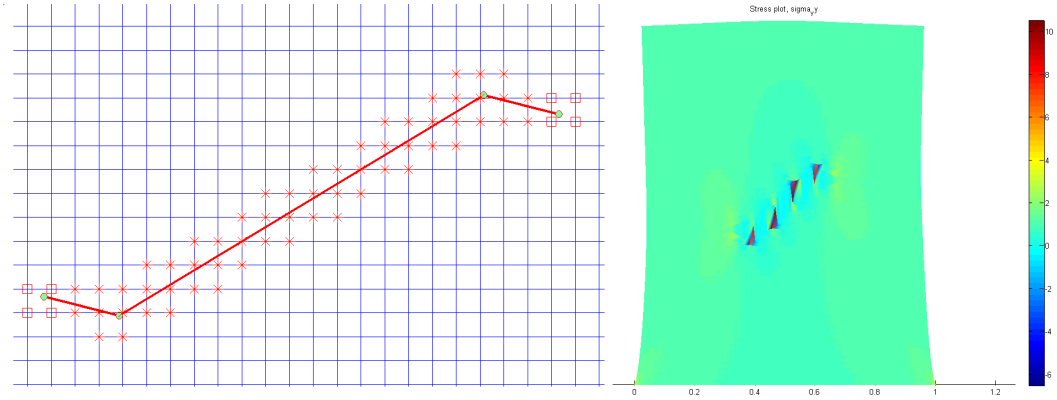


FIG. 4.13: Problem of stress concentration with a crack to close to a node.

Comprehension of the problem and resolution The bug came from the affectation of the heaviside function to these GP. To assign the value of the function at a GP, the code use the distance. Unfortunatly the distance was 0 when then GP were too close (because of the presence of a truncation). So the value associated to this node in **B** was 0 and that's why this locking appeared. Suppressing the truncation while computing the distance solve this problem

and erase the spurious stress concentration.

Remark: One could wonder what append when the crack is on the node? In fact, if precautions are taken for the initial definition of crack, the crack will never be *exactly* on a node and that's why this problem will not appear anymore.

4.4 Post-processing

During this internship a lot of computations were to be made. Mainly it was to study the influence of some parameters on the crack simulation. Results and parameters needed to be saved to be proceed afterwards.

Instead of doing repetitive tasks at each computation, these two following functions (Appendix 7.4)have been created. This permit to earn a lot of time while creating test routines and while proceeding the results.

• **data_save.m** which stores the data in a .txt file. First it write all the fixed parameters (ν, E, the domain size, date of computation, etc.). Then it writes the results between delimiters (" ___ "and " − − −− ") to be able to read the datas afterwards. In the main code call

- ▶ `data_saving(fname,labels,parameters,[],'0');` before the loop to save parameters of the test and give lebels of the data (`; size_mesh ; KI ; KII ; toc ;` for instance);
- ▶ `data_saving(fname,[],data,2,'1');` at the end of the loop to write these 4 values contained in the vector `data`;
- ▶ `data_saving(fname,[],[],[],'3');` after the loop to finalize the data.txt file (ending delimiter and closing).

Remark:

To save a entire array you can use `data_saving(fname,additional_label,xCr,ipas,'2');` at the end of the loop (like for saving the crack path named `xCr`). `Additional_label` is useful to print label to data sets in order to name these sets in a nexus plot.

• **data_post.m** which import the data from the file data.txt and plot these data. It is important to use a file created by `data_saving.m` because a special format is needed (delimiters, etc.).

This function have been designed to not need any coding in using it. The user first choose the data file with a windows interface. Then he choose which data to plot anmong a list of available data. He can also choose to the log mode (useful to make the most of convergence results for instance). And finally the figure will be displayed and the user would be able to change some style parameters directly in the figure windows. some parameters have already been selected (Latex font, font size, colors, etc.)

Remark:

This also allow to plot data comming from a test about the influence of the tip location in the element. The data.txt file just need to contain the flag `location` at the beginning of the data.

Results

5.1 Caution

Crack coincident with a node

The code is not able yet to handle cracks which coincide exactly with a node. This is not really a problem. In fact one has to be careful for the initial definition of the path. After that during the propagation the path will never be *exactly* on a node. That is why sometimes it is good to add an $\epsilon \ll 1$ to the initial coordinates of the crack and/or to the initial length of the crack.

Double split element: Some isolated computation have revealed double split elements (Fig. 5.1). The code can't handle it and give spurious results in that event. This problem disappear with mesh refinement. An other solution to suppress it is to use T3 because with the same number of dofs the number of elements will be double.

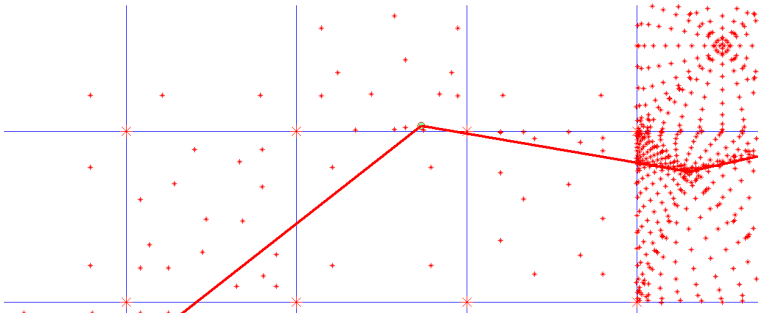


Fig. 5.1: Problem of double spli element.

Center In the following section you will see several case description. Have in mind that when the position of the crack is not specified that is because the crack (or group of cracks) is centered in the domain.

5.2 Exact boundary conditions : Griffith case

5.2.1 Description

Previous simulations have underlined that the domain size has a huge influence on the results. To suppress this influence, exact boundary conditions are applied on the choosen domain (ABCD). This allows to test the ability of the code to approximate the solution

without having to worry about the finite domain of the problem.

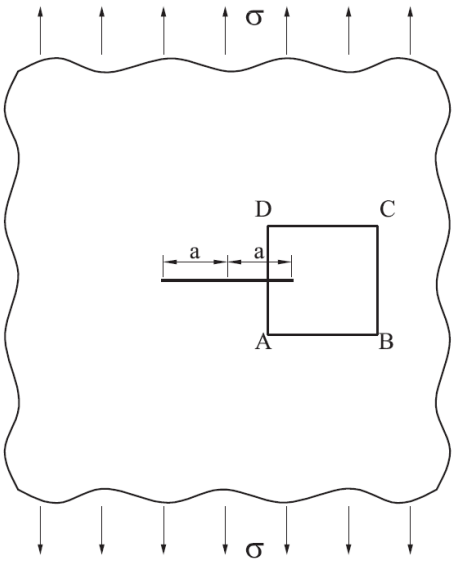


FIG. 5.2: Infinite cracked plate under tension and discretization around the crack tip.

Only the rectangle ABCD is discretized. Exact displacements are applied on all the boundary nodes of this rectangle with the following formula :

$$u_x = \frac{2(1+\nu)}{\sqrt{2\pi}} \frac{K_I}{E} \sqrt{r} \cos\frac{\theta}{2} \left(2 - 2\nu - \cos^2\frac{\theta}{2}\right)$$
$$u_y = \frac{2(1+\nu)}{\sqrt{2\pi}} \frac{K_I}{E} \sqrt{r} \sin\frac{\theta}{2} \left(2 - 2\nu - \cos^2\frac{\theta}{2}\right) \tag{5.1}$$

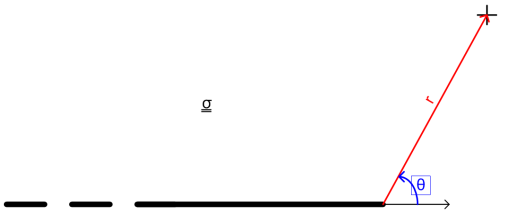


FIG. 5.3: Parameters for computing the exact displacements.

5.2.2 Convergence in energy

This computation will verify that we have a convergence in energy for each method. In this section the energy is computed with the EQN. (5.2):

$$E = \frac{1}{2} \underline{u}^t \mathbf{K} \underline{u} \tag{5.2}$$

This implementation of EQN. (5.2) is simple. This will give equivalent results to the strain energy error computed by EQN. (5.3)

$$E_g = \sum_{GP=1}^{GPnumber} (\underline{\varepsilon}_{exact}^T - \underline{\varepsilon}_{comp}^T) \mathcal{A} (\underline{\varepsilon}_{exact} - \underline{\varepsilon}_{comp}) det(\mathbf{J})\ w_{GP} \tag{5.3}$$

In theory the plot of the energy in function of the number of dofs in scale log.log is straight line if there is convergence and the slope a of it (< 1) give the rate of convergence (i.e. speed).

	FEM	XFEM	FleXFEM
slope	0.5184	0.5229	0.5352
Norm of residuals	0.015747	0.021637	0.031866

Table 5.1: Global energy convergence rates for FEMs.

The slopes obtained in TABLE 5.1 prooved that the code for the 3 methods do converge in strain energy. That also underlignes that the enrichment of the SFEM is not absurd. Also to have a better convergence rate for enriched methods the fixed enrichment area should be implemented. In fact the effect of the enrichment in XFEM vanish with mesh refinement

because the enriched area goes to zero. With an adapted fixed enrichment area, convergence rates would be improoved [4].

5.2.3 Convergence for K_I : comparison of classical FEM, XFEM and FleXFEM

With a domain 1*1 and a crack length of 100:

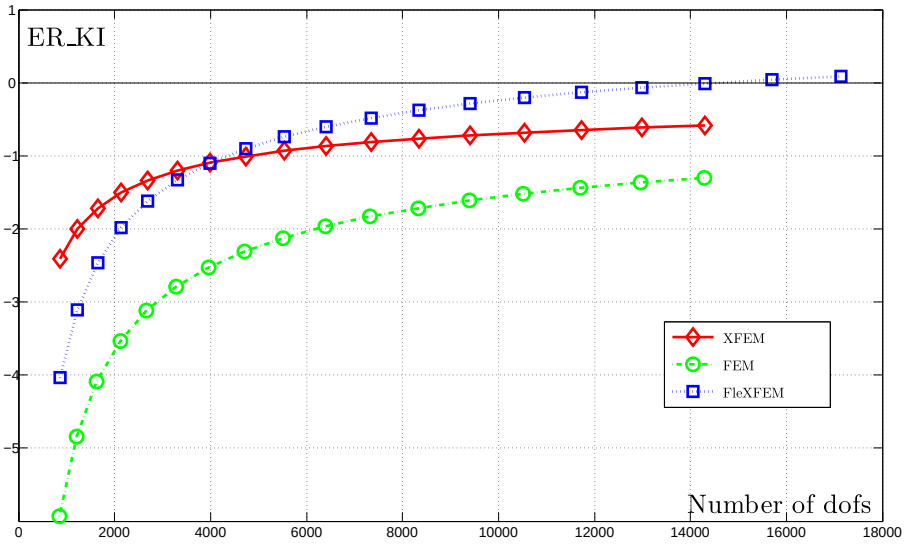


FIG. 5.4: Convergence of the relative error of the SIF K_I with mesh refinement for different FE methods.

Observations The XFEM permit to have a better accuracy than the classical FEM with an equivalent time of computation. Concerniong FleXFEM several computation have shown that the SIF don't converge to zero and that the accuracy of the SIF are really sensible to the quadrature chosen.

Remark: With an other implementation in which the number of nodes per meter is fixed and in which the domain area is increasing we noticed this: for the same number of elements in the domain the SIF values are the same. An other implementation where the total crack length was a parameter showed that it has no influence on the SIF's accuracy. That why in conclusion, only the number of dofs have an influence in the Griffith case.

5.3 Inclined crack under tension

5.3.1 Description

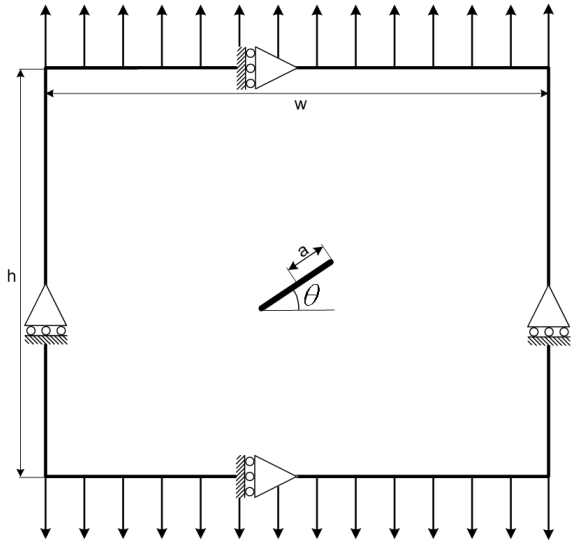


FIG. 5.5: Description of the inclined crack under tension.

These Boundary Conditions (BC) have been choosen in order to minimize the effect on finite domain and to have symmetries.

	units	value
E	$N.mm^{-2}$	2.10^7
ν	-	$0,3$
σ	$N.mm^{-2}$	10^3

Table 5.2: Parameters for inclined crack under tension

5.3.2 Angle influence

This computation has been made with $(L, D) = (4, 4)$ and $node\_density = 50$

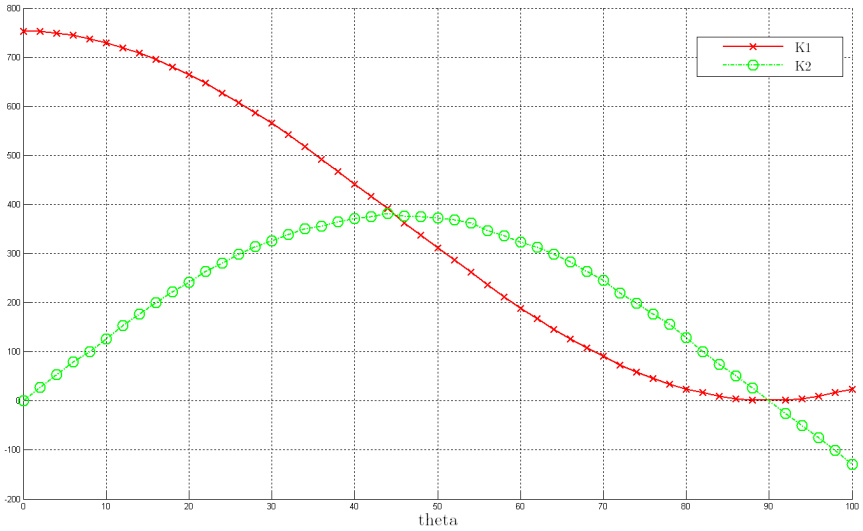


FIG. 5.6: Inclined crack: SIF in function of θ.

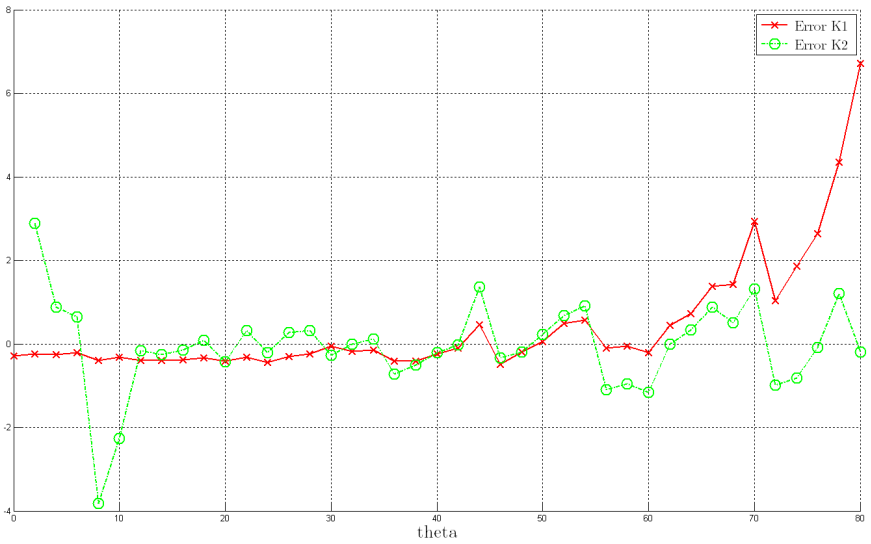


FIG. 5.7: Inclined crack: relative errors for the SIF in function of θ

The SIF plotted in FIG. 5.6 are regular and seems correct. Unfortunatly, the relative error is not tiny for all angles. The relative error for K_I and K_{II} when θ approches 90ř is normal. It is due to the little values of this SIF and the biggest impact of the integration error and computer imprecisions.

5.3.3 Propagation

All these results came from a computation with: $\beta = 40°$, $\Delta_{inc} = 0.055$, and $(L, D) = (1, 1)$

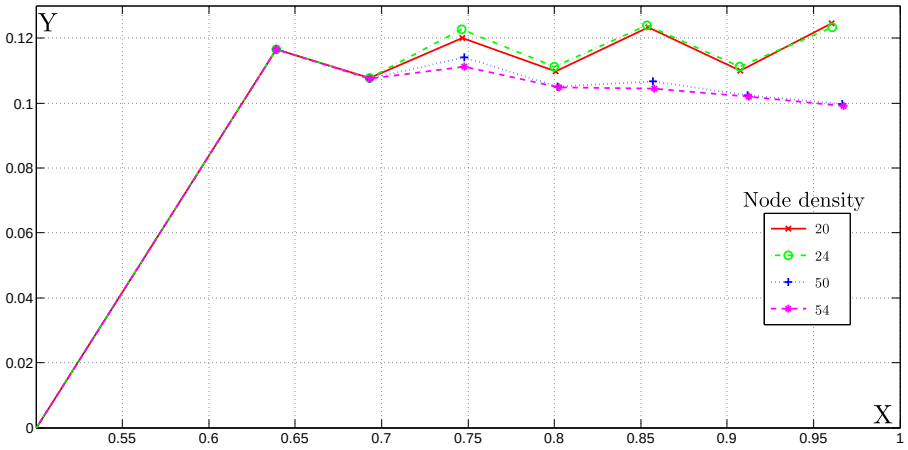


FIG. 5.8: Inclined crack: Path of the propagation for several node density.

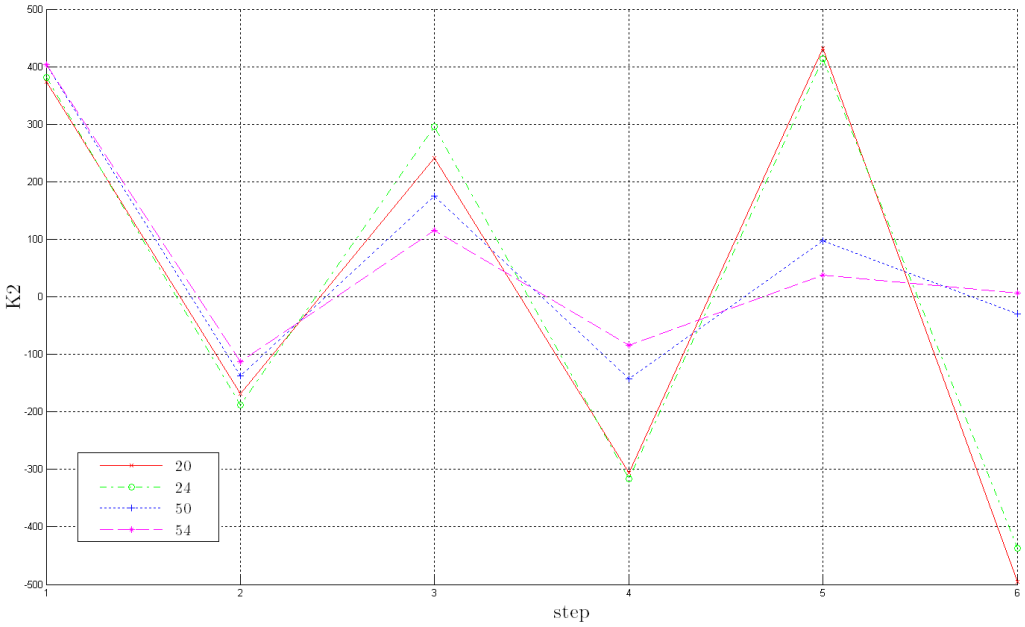


FIG. 5.9: Inclined crack: K2 for several node densityin function of the step of propagation.

FIG. 5.8 shows a convergence of the path with mesh refinement. Also the path has smaller oscillations with mesh refinement. This is due to the convergence for fine mesh of the SIF K_{II} visible on FIG. 5.9

5.3.4 Influence of the increment of the propagation

We wanted to know what is the influence of Δ_{inc} on the path. So with $(L, D) = (4, 4)$ and $node\_density = 50$ (i.e. 50000dofs) several propagations with different Δ_{inc} have been computed. Remark: the following equation should be verified to avoid computational problems:

$$\Delta_{inc} \geq \frac{\sqrt{2}}{node\_density - 1}$$

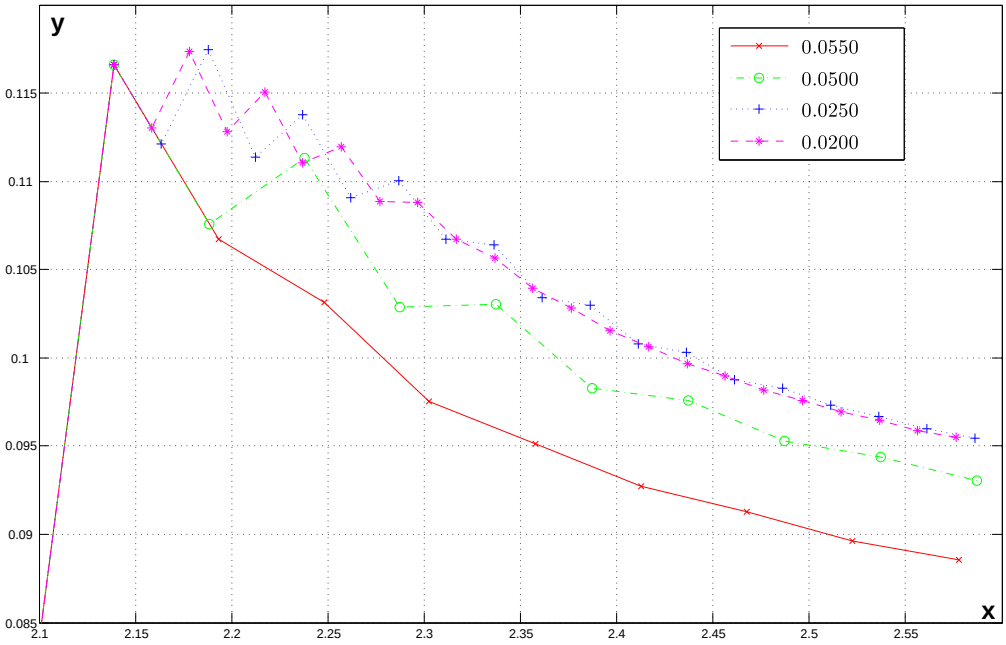


FIG. 5.10: Inclined crack: path for several increment of propagation Δ_{inc}.

Remark: note that the increment of propagation is really important. The computed path is significantly different for these values of Δ_{inc} . Fortunatly the path converge with reffinement of the path.

5.4 Offset cracks

5.4.1 Description:

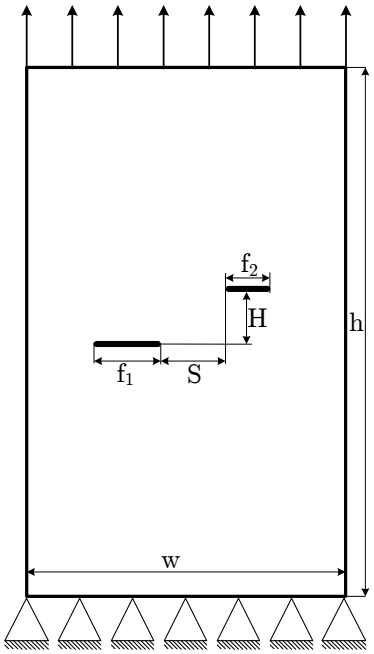


FIG. 5.11: Description of the offset cracks case.

For this case all computations have been made to compare the results given by our code with the results presented in [17]. This article also include experimental results for the propagation which have a special interest to test the accuracy of our code.

5.4.2 Influence of the distance between the 2 cracks

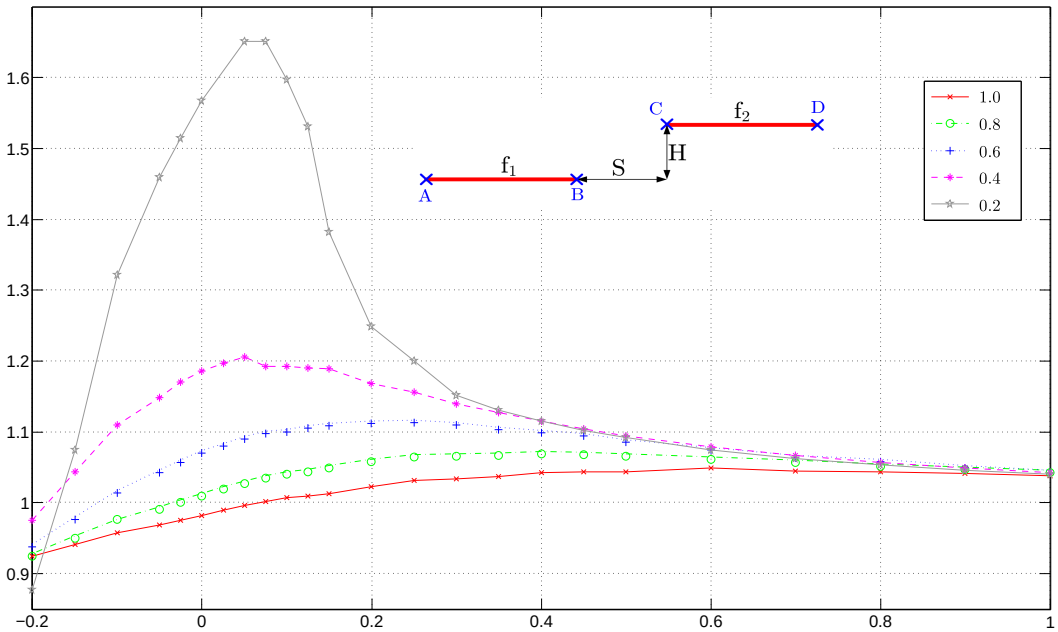


FIG. 5.12: Influence of the offset parameters (S in abscisse and H in legend) on the ratio $K_{Ib}/\sigma\sqrt{\pi f_1/2}$.

This test is realised with a "large" domain size (800x160). Because the cracks are close, the mesh needs to be very fine. See below the kink of results you obtain whan the Jdomain contains the other crack:

That is why sometimes the Jdomain contained enriched elements FIG. 5.13 and so lead to erroneous results. For instance in FIG. 5.14 the mesh is not refined enough because for $H = 10$ the J domain contains enriched elements and hence the SIF computation for $H = 10$ is not accurate.

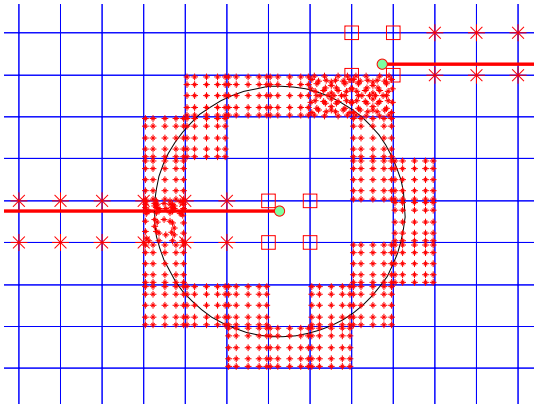


FIG. 5.13: Integration in SIF computation performed in enriched elements.

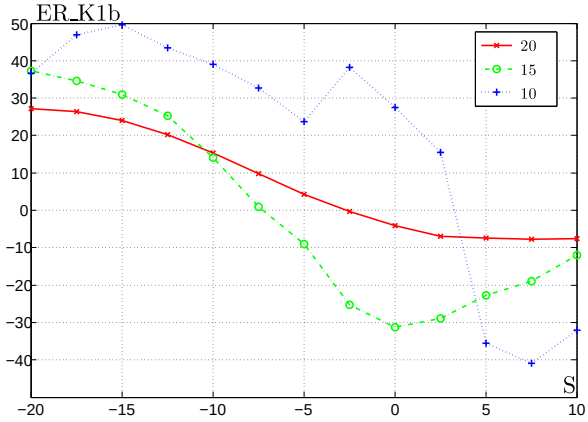


FIG. 5.14: Mistaken results obtained with a non valide J domain due to the presence of enriched elements in it.

That is a problem because the number of dofs is "high". The capability of the computer was a limit for this computation. It was not possible to use a mesh as fine as we wanted to. The solution for this problem was to use a finer mesh in x than in y.

5.4.3 Interaction of the cracks for different H and S

These computation have been made to study the interaction of a crack on the other. In absisce Er_K1b is the relative error of the SIF at the tip B compared to the exact value considering only one tip $K_{I,exact} = \sigma\sqrt{\pi * f_1/2}$.

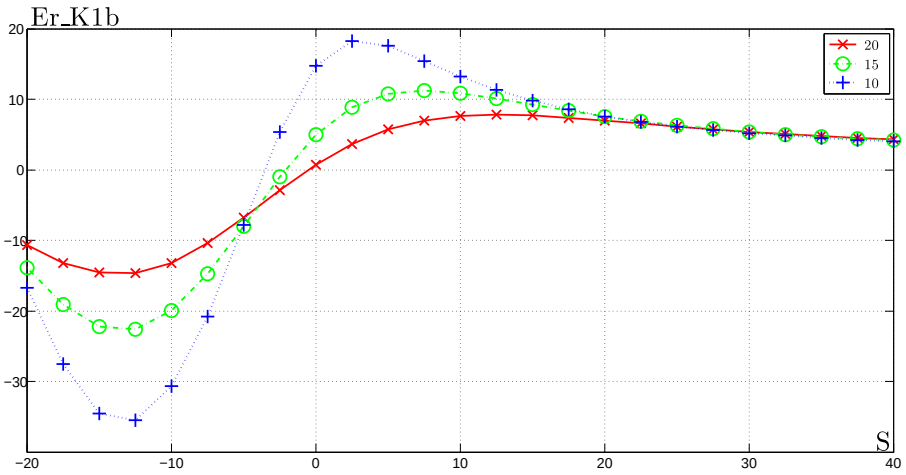


FIG. 5.15: Interaction of the cracks on the relative error in K_I for different H and S values.

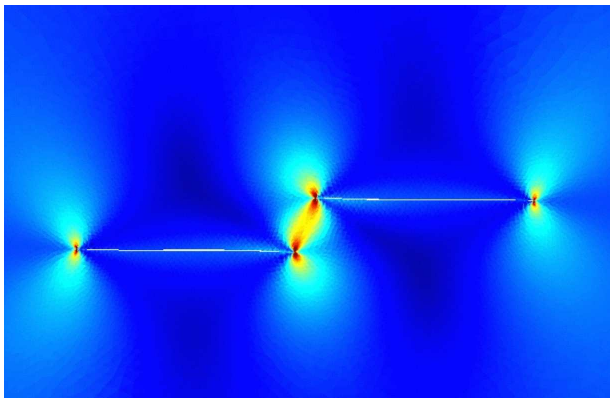


FIG. 5.16: Visible stress interaction of the 2 cracks.

Remarks: one can see on FIG. 5.15 that the closer the tips the larger the SIF. Also, with negative values of S, K_I is smaller than in the case of one center crack under tension.

5.4.4 Influence of Δ_{inc} on the path

On the contrary to the inclined crack, the increment of propagation does not have a huge influence on the path.

In conclusion, the increment of propagation length is important when the angle variation is important (as in FIG. 5.8).

So, in this case, the only effect of refining Δ_{inc} is to have a more smoothed path description.

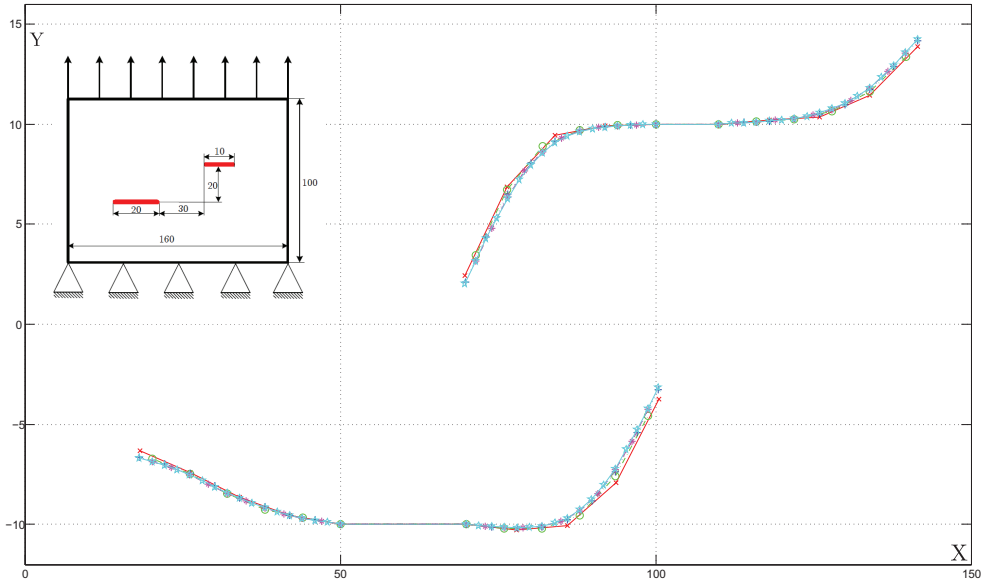


FIG. 5.17: Propagation path for the offset crack with different increment size of propagation.

5.4.5 Propagation

Depending on the values of H, S, f_1 and f_2 it is found experimentaly that the cracks either run into each other or do not [17]. We tried to simulate this experiement in order to obtain the same results:

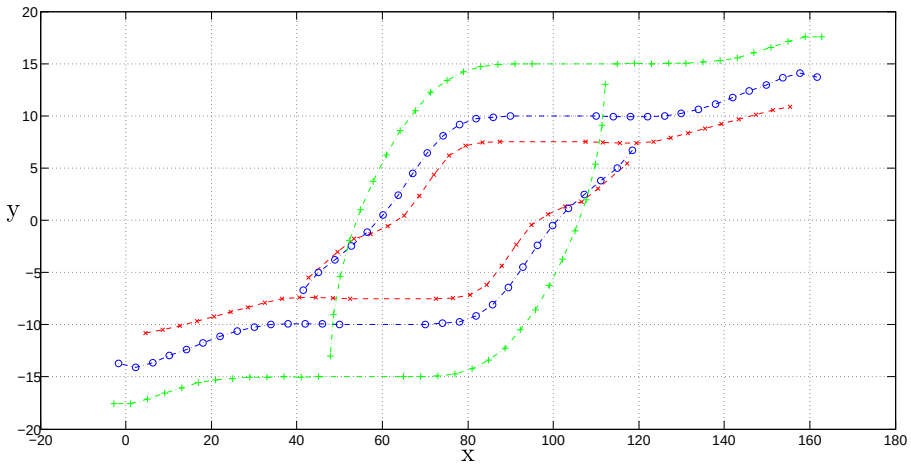


FIG. 5.18: Propagation for the offset carck for differnt values of H and S.

Unfortunatly in all cases the cracks are running into each other. The problem with the current code is that no propagation criterion is included i.e. at each step the increment of propagation is fixed a priori. But in reality in some cases the left tip will propagate more slowly than the right one and bacause both cracks have an interaction on each other the path will not be the same. Even worse : for some values of the parameters only one of the two cracks will propagate and or code is not able to simulate that for now. That's why we didn't manage to verify our results with the experimental ones.

Conclusion and future work

This work have permit to improve the accuracy. Mainly, the coding, the integration of the stiffness matrix, the tolerances (for singular cases)have been improved. The results obtained for the test case seems logical. Nevertheless these results still need to be compared with FleXFEM ones to finally published the article based on the comparison of XFEM and FleXFEM.

Nonetheless similar test cases still need to be computed with the FleXFEM code. Because this method is new, other detailed studies will be made to know the impact of the number of subcells on the convergence of the SIF for instance. The impact of the new smoothed domain integral (SmJ) on the computation of the Stress Intensity Factor is to be done too. Finally, it will be important to determine precisely in which situation the FleXFEM method is more suited to simulate cracks.

Appendix

7.1 Calculation for the theory

7.1.1 Mechanical basis - aide-memoire

Hooke's law:

In linear elasticity: In linear elasticity for isotropic materials:

$$\begin{cases} \sigma = \underline{\underline{\underline{\underline{\mathcal{A}}}}}.\varepsilon \\ \varepsilon = \underline{\underline{\underline{\underline{\mathcal{A}}}}}.\sigma \end{cases} \quad \Rightarrow \quad \begin{cases} \sigma = 2\nu\varepsilon + \lambda Tr(\varepsilon)\mathbf{I}_d \\ \varepsilon = \dfrac{1+\nu}{E}\sigma - \dfrac{\nu}{E}Tr(\sigma)\mathbf{I}_d \end{cases}$$

With:

$$E = \frac{\mu(3\lambda+2\mu)}{\lambda+\mu} \qquad \nu = \frac{\lambda}{2(\lambda+\mu)}$$
$$\mu = \frac{E}{2(1+\nu)} \qquad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}$$

The Voigt's notation

It's obvious that the previous form of the Hooke's law is not simple to implement in programming because of all the difficulty of indices. That's why often we use the Voigt's notation:

$$\{\sigma\} = \{\sigma_{11}\ \sigma_{22}\ \sigma_{33}\ \sigma_{33}\ \sigma_{12}\ \sigma_{13}\ \sigma_{23}\}^T \qquad \{\varepsilon\} = \{\varepsilon_{11}\ \varepsilon_{22}\ \varepsilon_{33}\ \varepsilon_{33}\ 2\varepsilon_{12}\ 2\varepsilon_{13}\ 2\varepsilon_{23}\}^T$$

Plane stress

$$\sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} & 0 \\ \sigma_{12} & \sigma_{22} & 0 \\ 0 & 0 & 0 \end{pmatrix} \text{ and } \varepsilon = \begin{pmatrix} \varepsilon_{11} & \varepsilon_{12} & 0 \\ \varepsilon_{12} & \varepsilon_{22} & 0 \\ 0 & 0 & \varepsilon_{33} \end{pmatrix} \text{ avec } \varepsilon_{33} = \frac{\nu}{E}(\sigma_{11}+\sigma_{22})$$

Plane strain

$$\varepsilon = \begin{bmatrix} \varepsilon_{11} & \varepsilon_{12} & 0 \\ \varepsilon_{12} & \varepsilon_{22} & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ and } \sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & 0 \\ \sigma_{12} & \sigma_{22} & 0 \\ 0 & 0 & \sigma_{33} \end{bmatrix} \text{ avec } \sigma_{33} = \lambda(\varepsilon_{11}+\varepsilon_{22})$$

Expression of $\mathcal{A}$ with plane strain and Voigt's notation

$$\begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{pmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & (1-2\nu)/2 \end{bmatrix} \begin{pmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{12} \end{pmatrix}$$

$\mathcal{A}$ is formulate with ν and E instead of μ and λ because these material coefficients are more common.

Expression of $\mathcal{A}$ with plane stress and Voigt's notation

$$\begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{pmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix} \begin{pmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{12} \end{pmatrix}$$

7.1.2 Demonstration of the criterion for θ_c

$$\forall (r,\theta) \in \mathbb{R}^{+*} \times]-\pi,\pi[, \quad \sigma_{\theta\theta} = \frac{1}{4}\frac{K_I}{\sqrt{2\pi r}}\left[3\cos\frac{\theta}{2} + \cos\frac{3\theta}{2}\right] + \frac{1}{4}\frac{K_{II}}{\sqrt{2\pi r}}\left[-3\sin\frac{\theta}{2} - 3\sin\frac{3\theta}{2}\right]$$

Let's find $\theta \in]-\pi,\pi[$ such as $\quad \forall r \in R^{+*}, \dfrac{\partial \sigma_{\theta\theta}}{\partial \theta} = 0$

$$\Rightarrow K_I\left(\sin\frac{\theta}{2} + sin\frac{3\theta}{2}\right) + K_{II}\left(\cos\frac{\theta}{2} + 3\cos\frac{3\theta}{2}\right) = 0$$

Let's consider the following trigonometrical formulas :

$$\begin{cases} \sin p + \sin q = 2\sin\dfrac{p+q}{2}\cos\dfrac{p-q}{2} \\ \\ \cos p + \cos q = 2\cos\dfrac{p+q}{2}\cos\dfrac{p-q}{2} \end{cases}$$

$$\Rightarrow K_I \cos\frac{\theta}{2}\sin\theta + K_{II}\left(\cos\theta\cos\frac{\theta}{2} + 2\cos\frac{3\theta}{2}\right) = 0$$

But :

$$\begin{aligned} \cos\frac{3\theta}{2} &= \cos\frac{\theta}{2}\cos\theta - \sin\frac{\theta}{2}\sin\theta \\ &= \cos\frac{\theta}{2}\cos\theta - 2\sin^2\frac{\theta}{2}\cos\frac{\theta}{2} \\ &= \cos\frac{\theta}{2}\cos\theta + (\cos\theta - 1)\cos\frac{\theta}{2} \\ &= 2\cos\theta\cos\frac{\theta}{2} - \cos\frac{\theta}{2} \end{aligned}$$

$\theta \in]-\pi,\pi[$ so $\cos\frac{\theta}{2} \neq 0$ Finally the equation to solve is :

$$K_I \sin\theta + K_{II}(3\cos\theta - 1) = 0 \tag{7.1}$$

Now let's resolve the previous equation by using $t = \tan\frac{\theta}{2}$

$$\cos\theta = \frac{2t}{1+t^2} \qquad \sin\theta = \frac{1-t^2}{1+t^2}$$

$$(7.1) \Rightarrow \quad K_I t + K_{II}(1+2t^2) = 0$$

$$\Rightarrow t = \frac{1}{4}\left(\frac{K_I}{K_{II}} \pm \sqrt{\left(\frac{K_I}{K_{II}}\right)^2 + 8}\right)$$

So finally:

$$\theta = 2\arctan\left[\frac{1}{4}\left(\frac{K_I}{K_{II}} \pm \sqrt{\left(\frac{K_I}{K_{II}}\right)^2 + 8}\right)\right] \tag{7.2}$$

7.2 Additionnal results

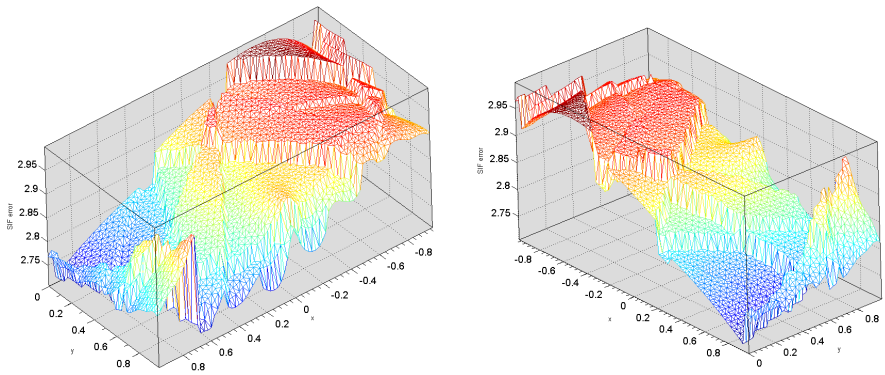


FIG. 7.1: Influence of the tip's location in the element on the relative error ER_{K_I} in 3D

7.3 Modified functions

In the following key functions, some change have been made on top of variable optimisation changments. These changment both simplify and improve the qualty of the code.

7.3.1 node_detect.m

```
function [type_elem,elem_crk,xTip,xVertex,enrich_node] = node_detect(xCr,elems)

% select the type of element then the type of element
global node element

type_elem = zeros(size(element,1),size(xCr,2));
elem_crk = zeros(size(element,1),4);

xCr_element = zeros(size(element,1),2);
xTip = zeros(size(element,1),2);
xVertex = zeros(size(element,1),2);

enrich_node = zeros(size(node,1),size(xCr,2));

% select the special elements(tip, vertex, split)
for kk = 1:size(xCr,2)
    for iel=1:size(elems,1) %loop on elems (=elements selected to be enricged)
        e = elems(iel) ;
```

```
        sctr=element(e,:);
        vv = node(sctr,:);
        crk_int = [];
        intes = 0;
        flag1 = 0;
        flag2 = 0;
        for kj = 1:size(xCr(kk).coor,1)-1 %loop over the elements of the fracture
            q1 = xCr(kk).coor(kj,:);
            q2 = xCr(kk).coor(kj+1,:);
            sctrl = [sctr sctr(1,1)];
            for iedge=1:size(sctr,2) %loop over the edges of elements
                nnode1=sctrl(iedge);
                nnode2=sctrl(iedge+1);
                p1 = [node(nnode1,:)];
                p2 = [node(nnode2,:)];
                intersect=segments_int_2d(p1,p2,q1,q2) ;
                intes = intes + intersect(1);
                if intersect(1) > 0
                    crk_int = [crk_int intersect(2) intersect(3)];
                    flag1 = inhull(xCr(kk).coor(kj,:),vv,[],-1e-8);
                    flag2 = inhull(xCr(kk).coor(kj+1,:),vv,[],-1e-8);
                    xCr_element(e,:)  = xCr(kk).coor(kj,:)  * flag1 + xCr(kk).coor(kj+1,:)  * flag2;
                end
            end %for iedge
        end

        %---- let's choose the categorie ----%
        if ((intes == 2) & (flag1 == 0) & (flag2 == 0)) % SPLIT
            type_elem(elems(iel),kk) = 2;
            elem_crk(e,:)  = crk_int;
        end
        if (((flag1 == 1) | flag2==1) & (intes==2)) % VERTEX
            type_elem(e,kk) = 3;
            elem_crk(e,:)  = crk_int;
            xVertex(e,:)  = xCr_element(e,:);
        end
        if (intes == 1) % TIP
            type_elem(e,kk) = 1;
            xTip(e,:)  = xCr_element(e,:);
            elem_crk(e,:)  = [crk_int xTip(e,1) xTip(e,2)];
        end
    end % iel
end % kk

% select the enriched nodes
for kk = 1:size(xCr,2)
    for iel=1:size(elems,1) %loop on elems (=elements selected to be enriched)
        sctr = element(elems(iel),:);
        if type_elem(elems(iel),kk) == 1 % tip
            enrich_node(sctr,kk) = 1;
        elseif type_elem(elems(iel),kk) == 2 % split
            for in=1:length(sctr) % loop on the nodes of the element
                if enrich_node (sctr(in),kk) == 0 % already enriched
                    [Aw, Awp] = support_area(sctr(in),elems(iel),type_elem,elem_crk,xVertex,kk);
                    if (abs(Awp / Aw) > 1e-4) & (abs((Aw-Awp) / Aw) > 1e-4)
                        enrich_node(sctr(in),kk) = 2;
                    end
                end
            end
        elseif type_elem(elems(iel),kk) == 3 %vertex
            for in=1:length(sctr)
                if enrich_node (sctr(in),kk) == 0 % already enriched
                    [Aw, Awp] = support_area(sctr(in),elems(iel),type_elem,elem_crk,xVertex,kk);
                    if ((abs(Awp / Aw) > 1e-4) & (abs((Aw-Awp) / Aw) > 1e-4))
                        enrich_node(sctr(in),kk) = 2;
                    end
                end
            end % loop on the nodes
        end %if
    end %loop on the elements
end %loop on cracks
```

7.3.2 gauss_rule.m

Structure of the function:

The structure of this function have been changed with a *switch* statement instead of *if*.

This erase problems due to the order of the statement of the *if* conditions.

Order of integration:

The orders of integration have been deplaced in the input of this function which allow two things. First it is possible to change these orders easily in the main program (for studying their influence for instance) and most of all to be able to use this function in the main.m and in the SIF.m too. In fact the order are different in both functions.

Blending elements:

It is more simple to deal with blending element because we also increase once and for all the order in blending/vertex and blending/split at the beginning.

```
function [W,Q] = gauss_rule(iel,enrich_node,elem_crk,type_elem,xTip,xVertex,elemType,normal_order,...
tip_order,split_order,vertex_order,sub_div_tip)

% Choose Gauss quadrature rules adapted at each element

global node element

sctr = element(iel,:); % element connectivity
sub_div = 0; % dfault number of sub-triangulation = 0

% --- blending elements ---- %
tip_enr = find(enrich_node(sctr,:)  == 1); % tip enriched element?
if size(tip_enr,1) > 0 % maybe to much but that's a security
    normal_order = tip_order;
    split_order = tip_order;
    vertex_order = tip_order;
    sub_div = sub_div_tip; % let's sub-divise as much as in the tip element
end

type_iel = max(type_elem(iel,:)); %to deal with multiples cracks
% ->just select the number which is not 0 (intersection is not possible)

switch num2str(type_iel)
case {'0'}
    if ((elemType == 'Q4') & (normal_order < tip_order))
        [W,Q] = quadrature(normal_order,'GAUSS',2);
    elseif (elemType == 'Q4')
        [W,Q] = disBlendingQ4quad(normal_order,node(sctr,:),sub_div); % higher order with triangulation...

    elseif elemType == 'T3'
        [W,Q] = quadrature(normal_order,'TRIANGULAR',2);
    end
case {'1'} % tip element -----------------------
    phi = LS(iel,elem_crk);
    nodes = node(sctr,:);
    if elemType == 'Q4'
        [W,Q] = disTipQ4quad(tip_order,phi,nodes,xTip(iel,:),node(element(iel,:),:),sub_div_tip);
    elseif elemType == 'T3'
        [W,Q] = disTipT3(tip_order,phi,nodes,xTip(iel,:));
    end
case {'2'} % split ---------------------------
    phi = LS(iel,elem_crk);
    if (elemType == 'Q4')
        [W,Q] = discontQ4quad(split_order,phi,node(element(iel,:),:),sub_div);
    elseif (elemType == 'T3')
        [W,Q] = discontT3(split_order,phi);
    end
case {'3'} % vertex --------------------------
    phi = LS(iel,elem_crk);
    nodes = node(sctr,:);
    if elemType == 'Q4'
        [W,Q] = disTipQ4quad(vertex_order,phi,nodes,xVertex(iel,:),node(element(iel,:),:),sub_div);%
same quadrature as tip
    elseif elemType == 'T3'
        [W,Q] = disTipT3(vertex_order,phi,nodes,xVertex(iel,:));
    end
end
```

7.3.3 SIF.m

Quadrature: In this function the vertex element for the J domain were not taken into account. They were consider as split element and this is obvious that the accuracy of the results was not optimum. That's one of the reason why the part concerning the quadrature have been replaced by the function gauss_rule. This allow us to take into account easily the quadrature of vertex and split elements which are in the J domain.

Warning message: a message to warn the user is display when the integration is performed on tip function enriched elements because this should not append for having accurate results.

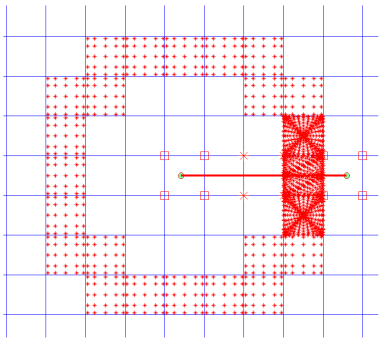


FIG. 7.2: Integration on tip enriched elements.

In the main before calling the function A simplify in the main where the SIF computation in done. Before there was a loop on all elements of the *enrdomain* with an *inhull* test. Moreover the same procedure was written for the right and the left tip although they are the same. That's why we simplified by making a loop on the tip elements.

7.4 New functions

7.4.1 sub_triangulation.m

```
% make the sub-triangulation of triangles (simplified version)
function [node,tri] = sub_triangulation (node,ntip,number_sub)
sub_rule = 2;
% 1 -- making a predefined number of subtriangle divison with 1 new
% point in the center of each triangle
% 2 -- making a predefined number of subtriangle divison with 1 new
% point in the center of each edge of the triangle
switch num2str(sub_rule)
    case{'1'}
    tri = delaunay(node(:,1),node(:,2));
    tri = tricheck(node,tri) ;% to assure a positive Jacobian
    for i=1:number_sub
        size_tri = size(tri,1);
        for ics = 1:size_tri
            node = [node ; sum(node(tri(ics,:),1))/3 sum(node(tri(ics,:),2))/3];
            tri = [tri ; tri(ics,1) tri(ics,2) size(node,1)];
            tri = [tri ; tri(ics,2) tri(ics,3) size(node,1)];
            tri(ics,:)  = [tri(ics,3) tri(ics,1) size(node,1)];
            tri = tricheck(node,tri) ;% to assure a positive Jacobian
        end
    end
    case{'2'}
    tri = delaunay(node(:,1),node(:,2));
    tri = tricheck(node,tri) ;
```

```
    size_tri = size(tri,1);
    for ics = 1:size_tri
        new_node1 = [sum(node(tri(ics,[1,2]),1))/2 sum(node(tri(ics,[1,2]),2))/2 ];
        new_node2 = [sum(node(tri(ics,[2,3]),1))/2 sum(node(tri(ics,[2,3]),2))/2 ];
        new_node3 = [sum(node(tri(ics,[3,1]),1))/2 sum(node(tri(ics,[3,1]),2))/2 ];
        node = [node; new_node1 ; new_node2 ; new_node3];
        tri = [tri ; tri(ics,1) (size(node,1) - 2) (size(node,1)) ];
        tri = [tri ; tri(ics,2) (size(node,1) - 2) (size(node,1) - 1) ];
        tri = [tri ; tri(ics,3) (size(node,1) - 1) (size(node,1)) ];
        tri(ics,:)  = [(size(node,1) - 2) (size(node,1) -1) (size(node,1))];
        tri = tricheck(node,tri) ;
    end

end
```

7.4.2 Pseg_dist.m

This function give the minimal distance of a point to one or several segment(s). It as been usefull to stop the multiple crack computation before the Jdomain contain the crack or branch enriched elements.

```
function dist = Pseg_dist(pt,segx,segy)

% compute the distance of a point pt to several segments
% dicribed by an array of points (segx,segy)
% function created by David NOËL: david.noel@ens-cachan.fr

Csize = size(segx)
Dist = [];
for m = 1 :  Csize-1 % compute only the half
    x = pt(1); y = pt(2);
    x0= segx(m); x1= segx(m+1);
    y0= segy(m); y1= segy(m+1);
    % projection point
    xp=(x0*y1^2-x1*y1*y0-y1*x0*y-x0*y1*y0+y1*x1*y+x1^2*x+x1*y0^2+x0*y0*y...
    -2*x1*x0*x-x1*y0*y+x0^2*x)/(y1^2-2*y1*y0+y0^2+x1^2-2*x1*x0+x0^2);
    yp=(y1*x1*x-y1*x0*x-y0*x1*x+y1^2*y+y1*x0^2-2*y1*y0*y-x0*x1*y0+y0*x0*x...
    +y0^2*y+x1^2*y0-x1*y1*x0)/(y1^2-2*y1*y0+y0^2+x1^2-2*x1*x0+x0^2);
    if (xp<x0) && (xp<x1)
        dist(m) = (x-x0)^2 + (y-y0)^2; % distance to node0
    elseif (xp>x0) && (xp>x1)
        dist(m) = (x-x1)^2 + (y-y1)^2; % distance to node1
    else
        dist(m) = (x-xp)^2 + (y-yp)^2; % projection distance
    end
end
```

7.4.3 data_saving.m

```
function [] = data_saving(fname,title,data,delimiter,type)

switch type

case {'0'} % save the parameters and main informations
    monid = fopen(fname,'w+t');
    tooday = clock;
    fprintf(monid,'Computation date :  %2.0f %2.0f %4.0f ',[tooday(3), tooday(2),tooday(1)]);
    fprintf(monid,' by:  David NOEL \n ');
    fprintf(monid,' version of the code :  v1.4 \n \n');
    fprintf(monid,'Parameters for case 1 \n \n');
    fprintf(monid,'Dimensions LxD: %3.0f %3.0f \n',[data(1),data(2)]);
    fprintf(monid,'Delta increment:  %12.4e \n', data(3));
    fprintf(monid,'Material properties:  %12.4e %12.4e \n',[data(4),data(5)]);
    fprintf(monid,'Load:  %12.4e \n',data(6));
    fprintf(monid,'initial number of nodes per meter:  %3.0f \n \n \n',data(7));
    fprintf(monid,'*data \n');
    fprintf(monid,title );
    fprintf(monid, '\n\n __');
    fclose(monid);
```

```
case {'1'} % save non nexus data line per line
    monid = fopen(fname,'a');
    if delimiter == 1 % put delimiters (multiple data sets)
        fprintf(monid,'-- \n \n \n');
        fprintf(monid,'__ ');
        fprintf(monid,'%12.4f \n',title);
    end
    for i=1:size(data) % writing data
        fprintf(monid, '%12.4e ' , data(i));
    end
    fprintf(monid, '\n');
    fclose(monid);

case {'2'} % save the path of the propagation
    monid = fopen(fname,'a'); % (!!!  data in a structure)
    if delimiter == 1 % put delimiters (multiple data sets)
        fprintf(monid,'-- \n \n \n');
        fprintf(monid,'__\n');
        %fprintf(monid,'%12.4f \n',title);
    end
    for toto=1:size(data.coor,1)
        fprintf(monid, '%12.4e %12.4e \n', [data.coor(toto,1),data.coor(toto,2)]);
    end
    fclose(monid);

case {'3'} % put final delimiters used to stop the imput in data_post.m
    monid = fopen(fname,'a');
    fprintf(monid,'--\n \n'); % final data-set delimiter
    fprintf(monid,'****\n \n'); % final data delimiter
    fprintf(monid,'*ENDFILE'); % final file delimiter
    fclose(monid);
end
```

7.4.4 data_post.m

```
 [fname,pname] = uigetfile('input\*.txt',...  % gets name of input file
'Choose input file');

fmid=fopen([pname fname],'rt'); % open the file

% ----- laoding the datas -------- %
tline = fgets(fmid); % read line
n=0;
while isempty(strfind(tline,'*ENDFILE')),
    tline = fgets(fmid);disp(tline); % display the parameters (info)
    if strfind(tline,'*data') % begin-delimiter of the data
        tline = fgets(fmid);
        title = sscanf(tline,'%c'); % get the names of the columns
        remain= title;
        for k = 1:size(strfind(title,';'),2)
            [token, remain] = strtok(remain,';'); % separate with ;
            col{k} = token; % vector of labels
        end
        while isempty(strfind(tline,'****')), % final-delimiter of the data
            tline = fgets(fmid);
            if strfind(tline,'__') % begin-delimiter of the data set
                n=n+1; % number of the set
                set_title{n} = sscanf(tline(3:length(tline)),'%c');
                m=1;
                tline = fgets(fmid);
                while isempty(strfind(tline,'--')), % final-delimiter of the data-set
                    temp_data = sscanf(tline, '%f %f %f %f %f %f %f %f ');
                    data(m,n,:)=temp_data';
                    tline = fgets(fmid);
                    m=m+1;
                end
            end
        end
    end
end
% ----------------- %

if strfind(set_title{1}, 'location')
    type = 'location'
elseif size(data,2)==1 % choose the plotting mode (simple set or multiple set)
    type = 'simple'
elseif size(data,2)>1
```

```
    type = 'multiple'
end

disp('The data are:')  % choose which data you want to plot
for i=1:size(col,2) % show the data available
    aff = strcat(num2str(i),' = ',col{i}); disp(aff)
end
% --- choosing data to plot --- %

disp('-----------')
if strfind(set_title{1}, 'location')
    toplot = input('Which data do you want to plot?  (ex:  [3,7,8])');
    elem_dens = input('For scaling :  element per unit(ex:39) = ?');
else
    xaxis = input('Column Number on x :  ');
    yaxis = input('Column Number(s) on y (ex:  [1,3,8]) :  ');
    logx = input('Put <<1>> if you want log sacle on x ');
    logy = input('Put <<1>> if you want log sacle on y ');
    disp('-----------')
    % --- modifications of the log scale --- %
    if logx == 1
        data(:,:,xaxis) = log(data(:,:,xaxis));
        col(xaxis) = strcat('log(',col(xaxis),')');
    end
    if logy == 1
        data(:,:,yaxis)=log(data(:,:,yaxis));
        col(yaxis) = strcat('log(',col(yaxis),')');
    end
end
% ---------------------- %

% -- ploting ----%
style = {'r-x','g-.o','b:+','m-*','c-p','r-.h','g-x','b-.o','m:+','c-*','r-p','g-.h'};
v = get(0,'ScreenSize');
h = 0;
switch type
case {'simple'}
    figure('Color',[1 1 1],'Position', ...
    [0 0 0.5*v(1,3) 0.5*v(1,4)]) % set properly the figure window
    hold on
    for ics=1:size(yaxis,2); % number of sets
        plot((data(:,1,xaxis)),(data(:,1,yaxis(ics))),style{ics},'LineWidth',2,'MarkerSize',12)
        xlabel(col{xaxis},'FontSize',20,'Interpreter','latex')
    end
    if size(yaxis,2)<2
        ylabel(col{yaxis},'FontSize',20,'Interpreter','latex')
    else
        h = legend(col{yaxis},1,'Interpreter','latex'); % 1 mean:  right high corner
        set(h,'FontSize',16,'Interpreter','latex')
    end
case {'multiple'}
    figure('Color',[1 1 1],'Position', ...
    [0 0 0.5*v(1,3) 0.5*v(1,4)]) % set properly the figure windows
    hold on
    for ics=1:n
        plot((data(:,ics,xaxis)),(data(:,ics,yaxis)),style{ics},'LineWidth',0.6,'MarkerSize',8)
    end
    h = legend(set_title,1);
    set(h,'FontSize',16,'Interpreter','latex')
    xlabel(col{xaxis},'FontSize',20,'Interpreter','latex')
    ylabel(col{yaxis},'FontSize',20,'Interpreter','latex')
case {'location'}
    for ics = 1:  size(toplot,2)
        dimension = input('Press 2 for 2D or 3 for 3D: dimension=?  ');
        data(:,1,1) = data(:,1,1)/(2*elem_dens); % transforming the element to a 2x2 domain
        data(:,1,2) = data(:,1,2)/(2*elem_dens);
        tri = delaunay(data(:,1,1),data(:,1,2)); %
        figure('Color',[1 1 1],'Position', [0 0 0.5*v(1,3) 0.5*v(1,4)])
        if dimension == 3
            %trisurf(tri,data(:,1),data(:,2),(data(:,1,toplot(ics))));
            trimesh(tri,data(:,1),data(:,2),(data(:,1,toplot(ics))));
        elseif dimension == 2
            plot_field([data(:,1),data(:,2)],tri,'T3',(data(:,1,toplot(ics))));
        else
            disp('unknown choice')
        end
    end
end
grid on;box;
```

Bibliography

[1] Stéphane Bordas, Antoine Legay, and Anthony Gravouil. Xf-fem mini-course. In *EPFL*.

[2] Tino Bog. A locking-free mindlin-reissner plate element for arbitrary cracks with smoothed curvature. Master's thesis, University of Canterbury, New Zeland, 2007.

[3] Timon Rabczuk and Wolfgang A. Wall. *EXtended Finite Element and Meshfree Methods*. Chair of Computational Mechanics, Technical University of Munich, 2006.

[4] Patrick Laborde, Julien Pommier, Yves Renard, and Michel Salaün. High-order extended finite element method for cracked domains. *International Journal for Numerical Methods in Engineering*, 2005.

[5] T. Belytschko and T. Black. Elastic crack growth in finite elements with minimal remeshing. *International Journal for Numerical Methods in Engineering*, 45(5):601–620, 1999.

[6] Alen Zehnder. Lecture notes on fracture mechanics. In *Theoretical an Applied Mechanics Papers and Monographs*.

[7] Soheil Mohammadi. *eXtended Finite Element Method*. 2008.

[8] Pierre-Alain Guidault. *Une statégie de calcul pour les stuctures fissurées : analyse locale-globale et approche multiéchelle pour la fissuration.* PhD thesis, ENS Cachan, 2005.

[9] J. R. RICE. A path independant integral and the approximate analysis of strain concentraion by notches ans cracks. *Journal of Applied Mechanics*, 35:379–386, 1968.

[10] N. Sukumar and J.-H. Prévost. Modeling quasi-static crack growth with the extended finite element method. part ii: Numerical applications. *International Jounal of Solids and Structures*, 40:7539–7552, 2003.

[11] Marc Bonnet and Attilio Frangi. *Analyse des solides déformables par la méthode des éléments finis*. 2006.

[12] Nguyen Vinh Phu. *An object approach to the eXtended Finite Element Method with application to fracure mechanics*. PhD thesis, Hochiminh City University of Technology, 2005.

[13] Stéphane Bordas, Timon Rabczuk, Nguyen-Xuan Hung, Sundarajan Natarajan, Tino Bog, Quan Do Minh, and Hiep Nguyen Vinh. Advances and recent developments on the smoothed finite element method (sfem) and first results in smoothed extended finite element method (smxfem). *Computers and Structures*, in press, 2008.

[14] H. Nguyen-Xuan, S. Bordas, and H. Nguyen-Dang. Smooth finite elements: Convergence, accuracy and properties. *International Journal for Numerical Methods in Engineering*, 2008. in press, doi:10.1002/nme.2146.

[15] G. R. Liu, T. T. Nguyen, K. Y. Dai, and K. Y. Lam. Theoretical aspects of the smoothed finite element method (sfem). *International Journal for Numerical Methods in Engineering*, 2007.

[16] Marc Duflot. A study of the representation of cracks with level sets. *International Journal for Numerical Methods in Engineering*, 2007.

[17] Kunio Hasegawa, Koichi Saito, and Katsumasa Miyazaki. Alignement rules for non-aligned flaws at lefm evaluation procedures. In *Pressure Vessels and Piping Division Conference*.