INTER-UNIVERSITY COOPERATION PROGRAM SPONSORED BY
BELGIAN GOVERNMENT AND REALIZED BY UNIVERSITY OF LIEGE

European Master
in Engineering Science
of Mechanics of Constructions

AN OBJECT ORIENTED APPROACH TO THE EXTENDED FINITE ELEMENT METHOD WITH APPLICATIONS TO FRACTURE MECHANICS

presented at

Hochiminh City University of Technology

by

NGUYEN VINH PHU

November 2005

INTER-UNIVERSITY COOPERATION PROGRAM SPONSORED BY
BELGIAN GOVERNMENT AND REALIZED BY UNIVERSITY OF LIEGE

European Master
in Engineering Science
of Mechanics of Constructions

AN OBJECT ORIENTED APPROACH TO THE EXTENDED FINITE ELEMENT METHOD WITH APPLICATIONS TO FRACTURE MECHANICS

presented at

Hochiminh City University of Technology

by

NGUYEN VINH PHU

November 2005

ABSTRACT

An Object-Oriented approach to the Extended Finite Element Method with Applications to Fracture Mechanics

Nguyen Vinh Phu

The eXtended Finite Element Method (X-FEM) is used to simulate the crack growth without remeshing. A C++ library for X-FEM is implemented which allows an easy extension for further development of the method.

In X-FEM, the standard FE approximation space is enriched with specially tailored functions to help capture the challenging features of a problem. Enrichment functions may be discontinuous (to model discontinuities in the field), their derivatives can be discontinuous (to model kinks in the field), or they can be chosen to incorporate a known characteristic of the solution (such as the square root singularity of linear elastic fracture mechanics).

Object oriented programming languages have been shown as robust tools to develop a variety of applications and the finite element methods are not exceptions. The major advantage of adopting an object oriented approach is that the program expansion is more simple and natural, as new implementations have little impact on the existing code. Thus, the reuse of code is maximized. Moreover, compared to the classical, structured programming, the use of object-oriented programming leads to a closer integration between theory and computer implementation. Object oriented programming is particularly useful in the development of large and complex programs, as the finite element schemes that should handle different element

types, constitutive models, analysis algorithms and, for X-FEM, enrichment items (cracks, material interfaces, holes), enrichment functions etc..

Applications to linear elastic fracture mechanics, single static crack as well as crack growth are presented and demonstrated with several numerical examples.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my advisor, Professor Nguyen Dang Hung, for his constant support, useful directions to us, students of EMMC. I feel very fortunate and proud to have been one of his students.

A few lines are too short to make a complete account of my deep appreciation for my second but major advisor, Stéphane Bordas. I wish to first thank him for accepted me as his first student, for his trust and constant encouragements, which have been essential ingredients in the construction of this thesis. It has been a profound pleasure to work with him during the last four months and I wish that this could last never-ending.

I thanks all the professors here at EMMC program who have dispensed wonderful lessons to me. I am thinking especially about Professors Nguyen Dang Hung for advanced solid mechanics and fracture mechanics, Patrick Dular for the finite element method, Pierre Beckers for computer graphics, Bui Cong Thanh for optimization of structures, Le Dinh Tuan for dynamics of constructions and Chu Quoc Thang for theory of plate and shell.

Although I started interacting with Dr. Raphael Bonnaz late in the course of this work, he has been attainable and responsive since I started working on the simulation of crack growth. His nice idea on crack update helped my work much improved. His code for post processing is really useful. It is a pleasure for me to know and work with him.

I thanks my former colleagues, Tran Duc Han who noticed me the X-FEM which became my thesis later. Phan Hong Quang for his warmth to what I asked

him and Chau Dinh Thanh, who checked the schedule of this thesis.

Without friends, my life would not be the same.

I thanks to Pham My, Truong Quang Tri, Le Bui Hong Phong, Dao Duy Luu for their external friendship to me. Especial thanks to my sister, Hoang Thi Kim Anh, who has been always beside me at the most difficult moments in my life.

I also would like to thank my new friend, Cyrille Dunant, who was so patient with my stupid questions, to share his expertise in the C++ programming language. His nice code on Delaunay triangulation, computational geometry helped me a lots. I am infinitely grateful for what he did and hope to be given a chance to repay him one day.

I special thanks to my family, my Parent, my Mother, Sister for all their emotional support and encouragement all along my studies. Finally, I dedicate this work to my parents.

To my parents

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Finite element methods have been widely used since their appearance in the early 1960s. This popularity is due to their ability to deal with numerous problems and due to their robustness. The method has been used with great success in areas such as solid mechanics, fluid mechanics, heat transfer, electromagnetism, etc. In spite of this success, there are several areas where current finite element methods are less than ideal.

Due to the fact that standard finite element methods are based on piecewise differentiable polynomial approximations, they are not well suited to problems with discontinuous and/or singular solutions. Typically, finite element methods require significant mesh refinement or meshes which conform with these features to get accurate results. These features occur in problems including fracture mechanics, contact, composites, etc. In response to this deficiency of standard finite element methods, extended finite elements have been developed.

With enriched approximation spaces, X-FEM is able to reproduce the prob-

lematic features, i.e., discontinuities and the singularities, and dramatically improved results are obtained. Recently, the partition of unity approach (Melenk and Babuška, 1996; Duarte and Oden, 1996) offered a systematic methodology to incorporate arbitrary functions into the finite element approximation space.

Since its first appearance (Moës, Dolbow, and Belytschko, 1999) the eXtended Finite Element Method (X-FEM) has been successfully applied to numerous solid mechanics problems such as 2-dimensional static and crack growth problems (Sukumar and Prévost, 2003), extension to 3-dimensional was presented in Sukumar et al. (2003). The XFEM has also been used to model computational phenomena in areas such as fluids mechanics, phase transformations (Chessa et al., 2002), material science and biofilm growth (Bordas, 2003; S. Bordas and Chopp, 2005) among others.

The application of object-oriented concepts to finite element programming has been receiving great attention over the last years (Mackerle, 2000). The major advantage of adopting an object oriented approach is that the program expansion is more simple and natural, as new implementations have little impact on the existing code. Thus, the reuse of code is maximized. Moreover, compared to the classical, structured programming, the use of object-oriented programming leads to a closer integration between theory and computer implementation. Object oriented programming is particularly useful in the development of large and complex programs, as finite element codes that are meant to handle different element types, constitutive models, and analysis algorithms.

1.2 Central Idea

The central idea of this thesis is the implementation of a flexible C++ library, named OpenXFEM++, for the X-FEM. This library was built based on the FEMOBJ (Zimmermann, Pelerin, and Bomme, 1992), an object oriented finite element package for static and dynamic nonlinear applications. The current code is used to solve a variety of problems in Linear Elastic Fracture Mechanics including the computation of fracture parameters and the simulation of crack growth. With the object-oriented approach, OpenXFEM++ is an extendable, easy-to-maintain computer code and new problems can be added without major obstacles. The C++ code has been compiled with Microsoft Visual Studio.NET 2003. Work is in progress for the porting of the code to g++.

1.3 Outline

An outline of the remainder of this thesis is as follows. In Chapter 2, after an overview of computational fracture mechanics, the X-FEM is examined in detail. The enriched approximation space is first presented. The definition and selection of enriched nodes are then stated. To exactly integrate the weak form (including discontinuous functions), elements split by the discontinuities need to be partitioned. Mesh geometry interaction, one of the main tasks of the X-FEM, is also presented. The final section of chapter 2 presents methods to increase the rate of convergence of the X-FEM. Chapter 3 introduces the proposed C++ implementation of the X-FEM. The most salient features of object-oriented programming are first quickly reviewed, then the design of the classes is explained. The extension to

new problems is presented in Section 3.4 Numerical examples are given in Chapter 4. Conclusions and future work are stated in Chapter 5. The class hierarchy of the OpenXFEM++ is provided in Appendix A. The data file and some preprocessing Matlab routines, are given in Appendix B. The derivation of the discretized equations, the derivatives of the near tip enrichment functions and the auxiliary fields used in the interaction integral to compute the SIFs are presented in Appendix C.

Chapter 2

Fracture mechanics and the eXtended Finite Element Method

2.1 Computational fracture mechanics

2.1.1 Crack tip fields in LEFM

In this section, a brief description of relevant aspects of linear elastic fracture mechanics is given (almost shifted from (Bordas, 2003). First, cracks are defined mathematically and the concept of loading mode is presented. Then, the crack tip fields are presented and commented upon. Solutions of the equilibrium equations are given and commented upon.

Mathematically, a crack is understood as a traction-free (unless otherwise stated) line of discontinuity in two dimensions and a traction-free surface of discontinuity, which needs not be planar, in three-dimensions.

Cracks may be loaded in three independent ways, as shown in Figure 2.1:

- Opening mode (*mode I*)
- Sliding or shearing mode (*mode II*)

- Tearing mode (*mode III*)

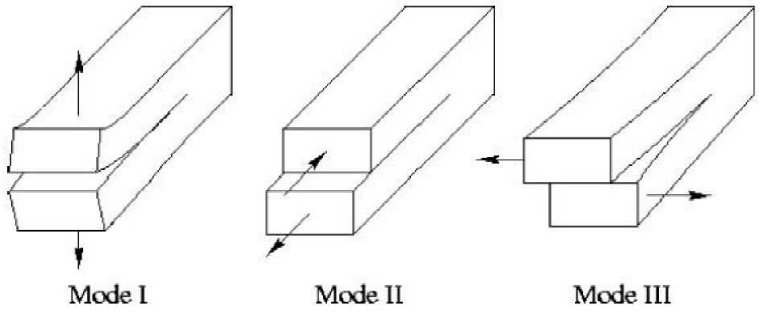


Figure 2.1: Modes of crack tip deformation

Crack tip stress, strain and displacement fields may be represented as a linear combination of those for each individual mode.

Via suitable stress functions (Kanninen and Popelar, 1985), one may develop expressions for each of the three modes of deformation presented above. For each loading mode, the magnitude of the stress field is defined by a scalar coefficient called the *stress intensity factor*. There is one stress intensity factor for each loading mode that will be referred to as K_I, K_{II} and K_{III} for modes I, II and III respectively. A crack is said to be loaded in *mixed-mode* when more than one stress intensity factor is necessary to represent the crack tip fields.

In a Cartesian coordinate system linked to the body Ω, and using a polar coordinate system linked to the crack tip, as shown in Figure 2.2, the full set of stress and displacement fields at any point $P = \mathbf{x} = (r, \theta) \in \Omega$, under plane strain (Timoshenko and Goodier, 1970; Kanninen and Popelar, 1985) are as follows:

- Mode I

$$
\begin{aligned}
u(r,\theta) &= \frac{K_I}{\mu}\sqrt{\frac{r}{2\pi}}\cos\frac{\theta}{2}\left(1-2\upsilon+\sin^2\frac{\theta}{2}\right)\\
v(r,\theta) &= \frac{K_I}{\mu}\sqrt{\frac{r}{2\pi}}\sin\frac{\theta}{2}\left(2-2\upsilon-\cos^2\frac{\theta}{2}\right)\\
w(r,\theta) &= 0
\end{aligned}
\tag{2.1}
$$

$$
\begin{aligned}
\sigma_{xx}(r,\theta) &= \frac{K_I}{\sqrt{2\pi r}}\cos\frac{\theta}{2}\left(1-\sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right)\\
\sigma_{yy}(r,\theta) &= \frac{K_I}{\sqrt{2\pi r}}\cos\frac{\theta}{2}\left(1+\sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right)\\
\sigma_{zz}(r,\theta) &= \nu\left(\sigma_{xx}+\sigma_{yy}\right)\\
\sigma_{xy}(r,\theta) &= \frac{K_I}{\sqrt{2\pi r}}\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\frac{3\theta}{2}\\
\sigma_{xz}(r,\theta) &= \sigma_{yz}(r,\theta) = 0
\end{aligned}
\tag{2.2}
$$

where K_I is the *mode I* stress intensity factor, defined by

$$
K_I = (2\pi r)^{\frac{1}{2}}\lim_{r\to 0}\sigma_{yy}(r,0) \tag{2.3}
$$

- Mode II

$$
\begin{aligned}
u(r,\theta) &= \frac{K_{II}}{\mu}\sqrt{\frac{r}{2\pi}}\sin\frac{\theta}{2}\left(2-2\upsilon+\cos^2\frac{\theta}{2}\right)\\
v(r,\theta) &= \frac{K_{II}}{\mu}\sqrt{\frac{r}{2\pi}}\cos\frac{\theta}{2}\left(-1+2\upsilon+\sin^2\frac{\theta}{2}\right)\\
w(r,\theta) &= 0
\end{aligned}
\tag{2.4}
$$

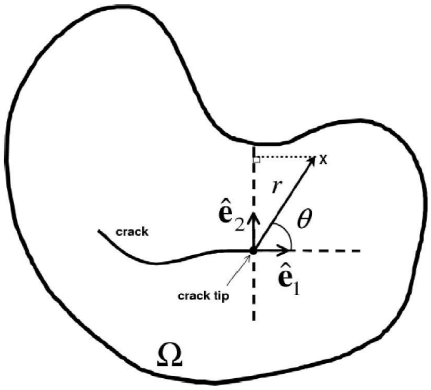


Figure 2.2: Polar coordinate system associated with a crack tip

$$
\begin{aligned}
\sigma_{xx}(r,\theta) &= -\frac{K_{II}}{\sqrt{2\pi r}}\sin\frac{\theta}{2}\left(2+\cos\frac{\theta}{2}\cos\frac{3\theta}{2}\right)\\
\sigma_{yy}(r,\theta) &= \frac{K_{II}}{\sqrt{2\pi r}}\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\frac{3\theta}{2}\\
\sigma_{zz}(r,\theta) &= \nu\left(\sigma_{xx}+\sigma_{yy}\right)\\
\sigma_{xy}(r,\theta) &= \frac{K_{II}}{\sqrt{2\pi r}}\cos\frac{\theta}{2}\left(1-\sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right)\\
\sigma_{xz}(r,\theta) &= \sigma_{yz}(r,\theta) = 0
\end{aligned}
\tag{2.5}
$$

where K_{II} is the *mode II* stress intensity factor, defined by

$$
K_{II} = (2\pi r)^{\frac{1}{2}} \lim_{r\to 0} \sigma_{xy}(r,0) \tag{2.6}
$$

In all the expressions above, E is Young's modulus, ν is Poisson's ratio and μ is the shear modulus of elasticity. To obtain the plane stress (Timoshenko and Goodier, 1970) expressions for the displacement and stress fields from the expressions above, σ_{zz} should be set to zero and Poisson's ratio ν should be everywhere replaced by $\nu/(1+\nu)$.

- Mode III

$$
\begin{aligned}
u(r,\theta) &= 0 \\
v(r,\theta) &= 0 \\
w(r,\theta) &= \frac{K_{III}}{\mu}\sqrt{\frac{2r}{\pi}}\sin\frac{\theta}{2}
\end{aligned}
\tag{2.7}
$$

$$
\begin{aligned}
\sigma_{xx}(r,\theta) &= 0 \\
\sigma_{yy}(r,\theta) &= 0 \\
\sigma_{zz}(r,\theta) &= 0 \\
\sigma_{xy}(r,\theta) &= 0 \\
\sigma_{xz}(r,\theta) &= -\frac{K_{III}}{\sqrt{2\pi r}}\sin\frac{\theta}{2} \\
\sigma_{yz}(r,\theta) &= \frac{K_{III}}{\sqrt{2\pi r}}\cos\frac{\theta}{2}
\end{aligned}
\tag{2.8}
$$

where μ is the shear modulus of elasticity and K_{III} is the *mode III* stress intensity factor, defined by

$$K_{III} = (2\pi r)^{\frac{1}{2}} \lim_{r\to 0} \sigma_{zy}(r,0) \tag{2.9}$$

In the above, the stress fields are square root singular (i.e., the stresses vary as $r^{-1/2}$ in the vicinity of the crack tip) and the stress intensity factors may be regarded as the amplitudes of the singular stress fields. Stress intensity factors carry the unit of *stress times square root of length*. The standard units are $MPa\sqrt{m}$ in the metric system and $ksi\sqrt{in}$ in the english (imperial) system.

2.1.2 Fatigue

By characterizing sub-critical crack growth using linear elastic fracture mechanics parameters, it is possible to predict crack growth rates under cyclic loading, and hence the number of cycles required for a crack to extend from some initial length to a predetermined length of interest to the designer. Paris and Erdogan (1963) proposed a law for fatigue crack growth relating the increment in crack advance da to the increment in number of cycles dN and the stress intensity factor range ΔK

$$\frac{da}{dN} = C\,(\Delta K)^m \tag{2.10}$$

where C and m are material constants, determined experimentally by standard fatigue tests and $\Delta K = K_{\max} - K_{\min}$ is the stress intensity factor range.

There are three regions characterizing fatigue crack growth in typical alloys, as depicted in Figure 2.3. Region A begins with a threshold value of stress intensity, $\Delta K_{threshold}$ below which crack propagation does not occur and continues until the slope of the curve becomes constant. $\Delta K_{threshold}$ may be associated with the attainment of a sufficient level of activity in the crack tip region.

Region B represents the zone in which the relationship between $\ln \frac{da}{dN}$ and $\ln \Delta K$ is linear and in this region, fatigue crack growth is governed by the Paris law. The life of many cracked engineering structures may be considered solely in this range once allowance has been made for the minimum crack length employed, which is normally related to the limitations of a particular inspection technique or design code requirements. For instance, the parameter that is controllable in practical situations is the inspection interval, i.e., the time interval between two

successive non-destructive evaluations of the structure of interest.

Region C exhibits a steep slope, where a small increment in the stress intensity factor range ΔK leads to a large increment in crack advance per cycle da/dN. The material behavior in this region is complicated by the possible attainment of plastic zone dimensions which are large compared with specimen dimensions, ductile tearing and values of $K_{\max}$ that approach fracture toughness (Barsoum, 1977).

To obtain an extension of Paris' law for mixed-mode loading, simply replace the stress intensity factor range ΔK by an *equivalent stress intensity factor* range ΔK_e. One possible expression for the equivalent stress intensity factor is

$$\Delta K_e = \sqrt{\Delta K_I^2 + \Delta K_{II}^2 + \Delta K_{III}^2} \tag{2.11}$$

2.1.3 Crack growth in a linear elastic solid

In this report, when the term *crack growth* or *crack propagation* is employed, it shall be to refer to *quasi-static crack growth*, in which inertia (acceleration) effects are neglected. In this quasi-static approach, the body is assumed in equilibrium at all times.

In addition to a criterion for crack extension, modelling crack growth necessitates the determination of the crack growth direction. Among the criteria for determining this growth direction, the following are commonly used:

1. The *maximum energy release rate* criterion (Nuismer, 1975).

2. The *maximum circumferential stress* (hoop stress) criterion or the maximum

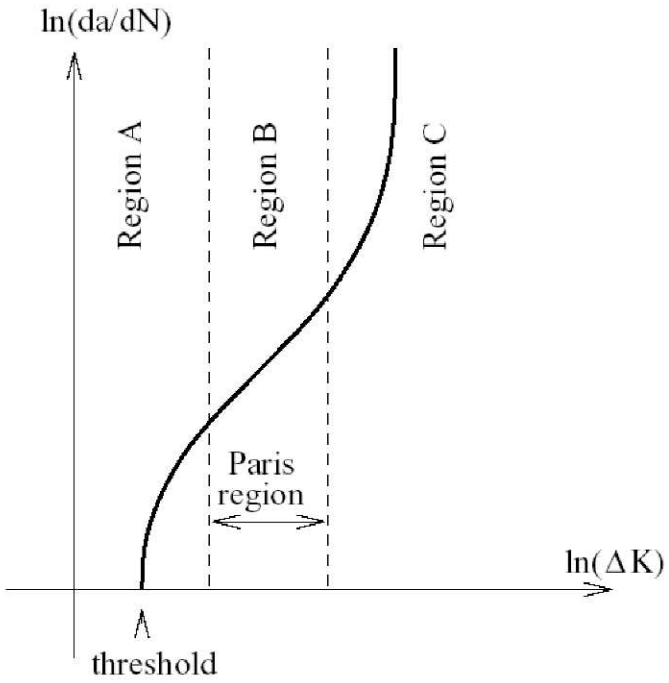


Figure 2.3: Paris fatigue crack growth, schematic representation: the Paris region corresponds to the portion of the curve that is almost linear.

principal stress criterion (Erdogan and Sih, 1963).

3. The *minimum strain energy density* criterion (Sih, 1973).

4. The *zero* K_{II} criterion.

All the crack growth simulations shown in this thesis use the maximum hoop stress criterion, which states that the crack will propagate from its tip in a direction $\theta = \theta_c$ such that the circumferential stress $\sigma_{\theta\theta}$ is maximum. The usual polar coordinate system related to the crack tip is used to describe the crack propagation direction, as shown in Figure 2.2.

Since $\sigma_{\theta\theta}$ is a principal stress in the direction of crack propagation, the crack will propagate in the direction such that the shear stress is zero. Setting the shear stress to zero in the expression for the asymptotic fields of elastic fracture mechanics then allows the determination of the value of the crack propagation angle as

$$\theta_c = 2 \arctan \left[\frac{1}{4} \left(\frac{K_I}{K_{II}} - sign(K_{II}) \sqrt{\left(\frac{K_I}{K_{II}} \right)^2 + 8} \right) \right] \tag{2.12}$$

where the *sign* function takes the value $+1$ when its argument is positive and the value -1 when its argument is negative, zero being assumed a positive number. The derivation of this equation is given in Appendix C.

2.1.4 Contour integrals and their domain representations in two-dimensions

Among the numerical methods for calculating fracture parameters, boundary integral methods (Forth and Keat, 1996; Sladek et al., 2000) and the domain integral method (Shih et al., 1986; Nikishkov and Atluri, 1987; Moran and Shih, 1987) have proved adequate tools. In this work, the domain integral method, in conjunction with interaction energy integrals, is used to determine mixed-mode stress intensity factors. In the interaction energy integral method, auxiliary fields are introduced and superimposed onto the actual fields satisfying the boundary value problem. By suitably selecting these auxiliary fields, a relationship can be found between the mixed-mode stress intensity factors and the interaction energy integrals. These integrals can be represented in so-called domain forms and evaluated in a post-processing step, once the solution to the boundary value problem is known.

The energy release rate for general mixed-mode problems in two dimensions can be written:

$$J = \frac{1}{E'}\left(K_I^2 + K_{II}^2\right) \tag{2.13}$$

where E' is defined as

$$E' = \begin{cases} \dfrac{E}{1-\nu^2} & \text{for plane strain} \\ E & \text{for plane stress} \end{cases} \tag{2.14}$$

Consider a crack in two dimensions. This crack is represented by a line-segment and a local orthogonal, Cartesian crack tip coordinate system $(\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2)$ is constructed.

Let Γ be a contour encompassing the crack tip and $\mathbf{n}$ be the unit normal to the contour Γ, oriented as shown in Figure 2.4. The contour integral J is defined as (Rice, 1968)

$$J = \int_{\Gamma} \left[W dx_2 - T_i \frac{\partial u_i}{\partial x_1} d\Gamma \right] = \int_{\Gamma} \left[W \delta_{1j} - \sigma_{ij} \frac{\partial u_i}{\partial x_1} \right] n_j d\Gamma \tag{2.15}$$

where $T_i = \sigma_{ij} n_j$ is the traction on the contour Γ.

Two states of a cracked body are considered. State 1, $(\sigma_{ij}^{(1)}, \epsilon_{ij}^{(1)}, u_i^{(1)})$, corresponds to the present state and state 2, $(\sigma_{ij}^{(2)}, \epsilon_{ij}^{(2)}, u_i^{(2)})$, is an auxiliary state. The J-integral for the sum of the two states is

$$J^{(1+2)} = \int_{\Gamma} \left[\frac{1}{2} (\sigma_{ij}^{(1)} + \sigma_{ij}^{(2)})(\epsilon_{ij}^{(1)} + \epsilon_{ij}^{(2)}) \delta_{1j} - (\sigma_{ij}^{(1)} + \sigma_{ij}^{(2)}) \frac{\partial (u_i^{(1)} + u_i^{(2)})}{\partial x_1} \right] n_j d\Gamma \tag{2.16}$$

Expanding and rearranging terms gives

$$J^{(1+2)} = J^{(1)} + J^{(2)} + I^{(1+2)} \tag{2.17}$$

where $I^{(1+2)}$ is called the interaction integral for states 1 and 2

$$I^{(1+2)} = \int_{\Gamma} \left[W^{(1,2)} \delta_{1j} - \sigma_{ij}^{(1)} \frac{\partial u_i^{(2)}}{\partial x_1} - \sigma_{ij}^{(2)} \frac{\partial u_i^{(1)}}{\partial x_1} \right] n_j d\Gamma \tag{2.18}$$

where $W^{(1,2)}$ is the interaction strain energy

$$W^{(1,2)} = \sigma_{ij}^{(1)} \epsilon_{ij}^{(2)} = \sigma_{ij}^{(2)} \epsilon_{ij}^{(1)} \tag{2.19}$$

Writing equation (2.13) for the combined states gives

$$J^{(1,2)} = J^{(1)} + J^{(2)} + \frac{2}{E'}(K_I^{(1)}K_I^{(2)} + K_{II}^{(2)}K_{II}^{(1)}) \tag{2.20}$$

Equating (2.17) and (2.20) leads to the following relationship

$$I^{(1+2)} = \frac{2}{E'}(K_I^{(1)}K_I^{(2)} + K_{II}^{(2)}K_{II}^{(1)}) \tag{2.21}$$

Making suitable choice of state 2 as the pure mode I asymptotic fields with $K_I^{(2)} = 1, K_{II}^{(2)} = 0$, gives the mode I SIF in terms of the interaction integral

$$K_I^{(1)} = \frac{E'}{2} I^{(1,ModeI)} \tag{2.22}$$

Similarly, choose state 2 as the pure mode II asymptotic fields with $K_{II}^{(2)} = 1, K_I^{(2)} = 0$, gives the mode II SIF in terms of the interaction integral

$$K_{II}^{(1)} = \frac{E'}{2} I^{(1,ModeII)} \tag{2.23}$$

The contour integral (2.18) is not well suited for its numerical evaluation. It is useful, therefore, to recast this integral into an equivalent domain form by multiplying the integrand with a sufficiently smooth weighting function $q(\mathbf{x})$ which takes a value of unity on an open set containing the crack tip and vanishes on an outer prescribed contour C_0.

The interaction integral for states 1 and 2 can be written (see Figure 2.4):

$$I^{(1,2)} = \lim_{\Gamma \to 0} \left\{ \int_{\Gamma \cup C_0 \cup C^+ \cup C^-} \left[W^{(1,2)} \delta_{1j} - \sigma_{ij}^{(1)} \frac{\partial u_i^{(2)}}{\partial x_1} - \sigma_{ij}^{(2)} \frac{\partial u_i^{(1)}}{\partial x_1} \right] q m_j ds \right\} \tag{2.24}$$

In Figure 2.4, let the contour C be the union of the curves Γ, C_0, C^+ and C^- and let m_j be the outward normal to the domain A.

Note that the boundary of the domain A is $\partial A = C = \Gamma \cup C_0 \cup C^+ \cup C^-$. The outward normal m_j to the domain A is:

$$\mathbf{m} = \begin{cases} -\mathbf{n} & \text{on} \quad \Gamma \\ \\ +\mathbf{n} & \text{on} \quad C_0 \cup C^+ \cup C^- \end{cases} \tag{2.25}$$

From (2.24), and using the result in (2.25), and the divergence theorem the interaction integral can be simplified

$$I^{(1,2)} = \int_A \left[-W^{(1,2)} \delta_{1j} + \sigma_{ij}^{(1)} \frac{\partial u_i^{(2)}}{\partial x_1} + \sigma_{ij}^{(2)} \frac{\partial u_i^{(1)}}{\partial x_1} \right] \frac{\partial q}{\partial x_j} dA \tag{2.26}$$

It is critical to note that, in order to derive equation (2.24), the crack faces are assumed *straight* and *traction-free*. This remark shall be kept in mind when assessing the accuracy of the code in computing stress intensity factors at the tips of a circular-arc crack in Section 4.1. Other domain integrals were developed for curved cracks in 2D. See for example (M. Lorentzon, 2000), in which a path-independent integral is derived for non-straight cracks.

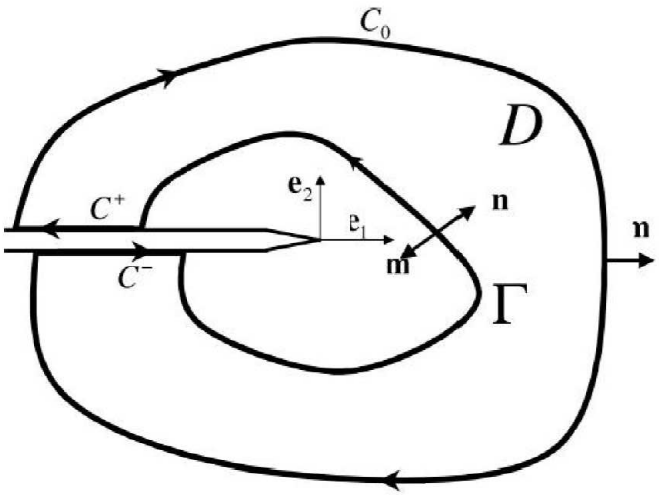


Figure 2.4: Domain used for computation of mixed mode stress intensity factors in two dimensional space

2.2 Modeling cracks with the eXtended Finite Element Method

2.2.1 Governing equations

In this section, the governing equations of elastostatics with internal boundaries are briefly reviewed and an associated weak form is given. Consider a domain Ω, bounded by Γ. The boundary is partitioned into three sets: Γ_u, Γ_t and Γ_c as shown in Figure 2.5. Displacements are prescribed on Γ_u, tractions are prescribed on Γ_t and all the Γ_c are assumed to be a traction free surface.

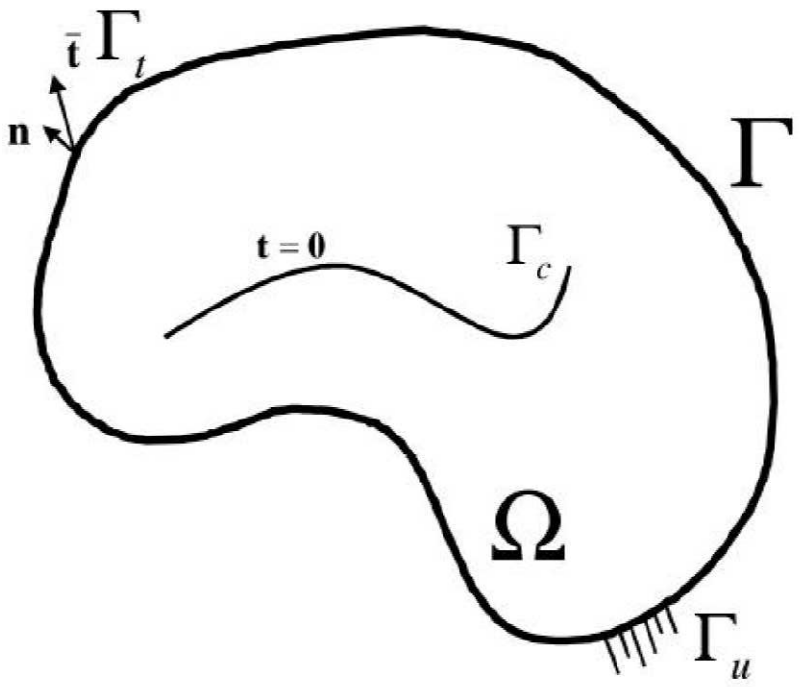


Figure 2.5: Body with a crack

The equilibrium conditions and boundary conditions for this problem are

$$\boldsymbol{\nabla} \cdot \sigma + \mathbf{b} = \mathbf{0} \quad in\ \Omega \tag{2.27}$$

$$\sigma \cdot \mathbf{n} = \bar{\mathbf{t}} \quad on\ \Gamma_t \tag{2.28}$$

$$\sigma \cdot \mathbf{n} = \mathbf{0} \quad on\ \Gamma_c \tag{2.29}$$

$$u = \bar{u} \quad on \quad \Gamma_u \tag{2.30}$$

where σ is the Cauchy stress tensor, $\mathbf{u}$ is the displacement field, $\mathbf{b}$ is the body force per unit volume and $\mathbf{n}$ is the unit outward normal. It is assumed that displacements remain small and the kinematics equations consist of the strain-displacement relation:

$$\varepsilon = \varepsilon(\mathbf{u}) = \boldsymbol{\nabla}_s \mathbf{u} \tag{2.31}$$

where $\boldsymbol{\nabla}_s(\cdot)$ is the symmetric part of the gradient operator. The constitutive relation for the elastic material under consideration is given by Hooke's law

$$\sigma = \mathbf{C} : \varepsilon \tag{2.32}$$

2.2.2 Variational formulation or Weak form

Let the space of admissible displacement fields (trial function space) be defined by

$$\mathcal{U} = \{\mathbf{u} \in \mathcal{S} | \mathbf{u} = \bar{\mathbf{u}} \text{ on } \Gamma_u \text{ and } \mathbf{u} \text{ discontinuous on } \Gamma_c\} \tag{2.33}$$

Babuška and Rosenzweig (1972) and Grisvard (1985) discuss in detail the choice of the space of admissible displacements $\mathcal{S}$ when the body contains internal boundaries or re-entrant corners. Similarly, the test function space may be defined as

$$\mathcal{U}_0 = \{\mathbf{v} \in \mathcal{S} | \mathbf{v} = \mathbf{0} \text{ on } \Gamma_u \text{ and } \mathbf{v} \text{ discontinuous on } \Gamma_c\} \tag{2.34}$$

A weak formulation of the equilibrium equations is given by

$$\text{Find } \mathbf{u} \in \mathcal{U} \quad | \quad \forall \mathbf{v} \in \mathcal{U}_0, \int_\Omega \sigma(\mathbf{u}) : \varepsilon(\mathbf{v}) d\Omega = \int_\Omega \mathbf{b} \cdot \mathbf{v} d\Gamma + \int_{\Gamma_t} \bar{\mathbf{t}} \cdot \mathbf{v} d\Gamma \tag{2.35}$$

or, using the constitutive relation,

$$\text{Find } \mathbf{u} \in \mathcal{U} \quad | \quad \forall \mathbf{v} \in \mathcal{U}_0, \int_\Omega \varepsilon(\mathbf{u}) : \mathbf{C} : \varepsilon(\mathbf{v}) d\Omega = \int_\Omega \mathbf{b} \cdot \mathbf{v} d\Gamma + \int_{\Gamma_t} \bar{\mathbf{t}} \cdot \mathbf{v} d\Gamma \tag{2.36}$$

Define the bilinear form $\mathcal{B}$

$$\forall \mathbf{u} \in \mathcal{U}, \forall \mathbf{v} \in \mathcal{U}_0, \ \mathcal{B}(\mathbf{u}, \mathbf{v}) = \int_\Omega \epsilon(\mathbf{u}) : \mathbf{C} : \varepsilon(\mathbf{v}) d\Omega \tag{2.37}$$

and the linear form $\mathcal{L}$

$$\forall \mathbf{v} \in \mathcal{U}_0, \mathcal{L}(\mathbf{v}) = \int_\Omega \mathbf{b} \cdot \mathbf{v} d\Gamma + \int_{\Gamma_t} \bar{\mathbf{t}} \cdot \mathbf{v} d\Gamma \tag{2.38}$$

With these notations, the above can be rewritten

$$\text{Find } \mathbf{u} \in \mathcal{U} \quad | \quad \forall \mathbf{v} \in \mathcal{U}_0,\ \mathcal{B}(\mathbf{u}, \mathbf{v}) = \mathcal{L}(\mathbf{v}) \tag{2.39}$$

This convenient notation will be used in Appendix C to derive the discretized equations for the eXtended Finite Element Method.

2.2.3 Extended finite element approximation

The basic idea of the X-FEM is to enrich a classical finite element space with some additional functions. These functions are built as the product of global enrichment functions with some finite element shape functions.

Consider a point $\mathbf{x}$ that lies inside a finite element e. Denote the element's nodal set as $\mathcal{N}_e = \{n_1, n_2, ..., n_{m_e}\}$, where m_e is the number of nodes of element e. The enriched displacement approximation for a vector-valued function $\mathbf{u}^h : \mathbb{R}^d \rightarrow \mathbb{R}^d$ assumes the form

$$\mathbf{u}^h(\mathbf{x}) = \sum_{I \in \mathcal{N}_e} N_I(\mathbf{x})\mathbf{u}_I + \sum_{J \in \mathcal{N}^{enr}} \widetilde{N_j}(\mathbf{x})\Psi(\mathbf{x})\mathbf{a}_J \tag{2.40}$$

where the nodal set $\mathcal{N}^{enr}$ is the set of nodes whose support is intersected by the domain Ω_g associated with a geometric entity such as a hole, or crack surface and $\mathcal{N}_e$ is the set of nodes that are not enriched. N_I and $\widetilde{N_J}$ are finite element shape functions. Mathematically,

$$\mathcal{N}^{enr} = \{n_J : n_J \in \mathcal{N}_e | \omega_J \cap \Omega_{enr} \neq \mathrm{O}\} \tag{2.41}$$

In the above equation, $\omega_J = \text{supp}(n_J)$ is the support of the nodal shape function $N_J(\mathbf{x})$, which consists of the union of all elements with n_J as one of its vertices; and Ω_{enr} is the domain associated with a geometric entity such as a hole, or crack surface. The choice of the function $\Psi\colon \mathbf{x} \mapsto \Psi(\mathbf{x})$ depends on the geometric entity under consideration.

To retain the interpolation property of the approximation, i.e., $\mathbf{u}^h(\mathbf{x}_I) = \mathbf{u}_I$, the following modification is often made

$$\mathbf{u}^h(\mathbf{x}) = \sum_{I\in\mathcal{N}_e} N_I(\mathbf{x})\mathbf{u}_I + \sum_{J\in\mathcal{N}^{enr}} \widetilde{N_J}(\mathbf{x})(\Psi(\mathbf{x}) - \Psi(\mathbf{x}_J))\mathbf{a}_J \tag{2.42}$$

Note that from (2.42) the enriched part of the approximation vanishes at the nodes.

In the particular instance of 2D crack modeling, the enriched displacement approximation is written as (Moës, Dolbow, and Belytschko, 1999)

$$\mathbf{u}^h(\mathbf{x}) = \sum_{I\in\mathcal{N}} N_I(\mathbf{x})\mathbf{u}_I + \sum_{J\in\mathcal{N}^{disc}} \widetilde{N}_J(\mathbf{x})H_J(\mathbf{x})\mathbf{a}_J + \sum_{K\in\mathcal{N}^{asympt}} \widetilde{N}_K(\mathbf{x}) \sum_{\alpha=1}^{4} B_{\alpha K}(\mathbf{x})\mathbf{b}_{\alpha K} \tag{2.43}$$

where $\mathcal{N}$ is the set of conventional (not enriched) nodes, $\mathcal{N}^{disc}$ is the set of nodes enriched with discontinuous enrichement and $\mathcal{N}^{asympt}$ the set of nodes enriched with asymptotic enrichment.

The determination of $\mathcal{N}^{disc}$ and $\mathcal{N}^{asympt}$ is done according to the following rules (see Figure 2.6):

- $\mathcal{N}^{disc}$ is the set of nodes whose support is entirely split by the crack,
- $\mathcal{N}^{asympt}$ is the set of nodes which contain the crack tip in the support of their shape functions.

H is the modified Heaviside step function and the functions B span the space of the asymptotic crack tip fields. The $\mathbf{u}_I$'s are the unknown standard displacement degrees of freedom associated with node I, the $\mathbf{a}_J$'s are the unknown enrichment coefficients associated with the discontinuous enrichment function H_J active on node J and defined by

$$H_J(\mathbf{x}) = H(\mathbf{x}) - H(\mathbf{x}_J) \tag{2.44}$$

$H(\mathbf{x})$ is the modified Heaviside step function which takes on the value $+1$ above the crack and -1 below the crack:

$$H(\mathbf{x}) = \begin{cases} +1 & \text{if } (\mathbf{x} - \mathbf{x}^*).\mathbf{n} \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.45}$$

where $\mathbf{x}$ is a sample point, $\mathbf{x}^*$ (lies on the crack) is the closest point projection of $\mathbf{x}$, and $\mathbf{n}$ is the unit outward normal to the crack at $\mathbf{x}^*$.

Finally, $\mathbf{b}_{\alpha K}$ are additional, enrichment degrees of freedom associated with the enrichment function $B_{\alpha K}$ active at node K and defined by

$$B_{\alpha K}(\mathbf{x}) = B_\alpha(\mathbf{x}) - B_\alpha(\mathbf{x}_K) \tag{2.46}$$

The crack tip enrichment functions in isotropic elasticity B_α are –obtained from the asymptotic displacement fields.

$$\mathbf{B} \equiv [B_1, B_2, B_3, B_4] = \left[\sqrt{r}\sin\frac{\theta}{2}, \sqrt{r}\cos\frac{\theta}{2}, \sqrt{r}\sin\frac{\theta}{2}\cos\theta, \sqrt{r}\cos\frac{\theta}{2}\cos\theta\right] \tag{2.47}$$

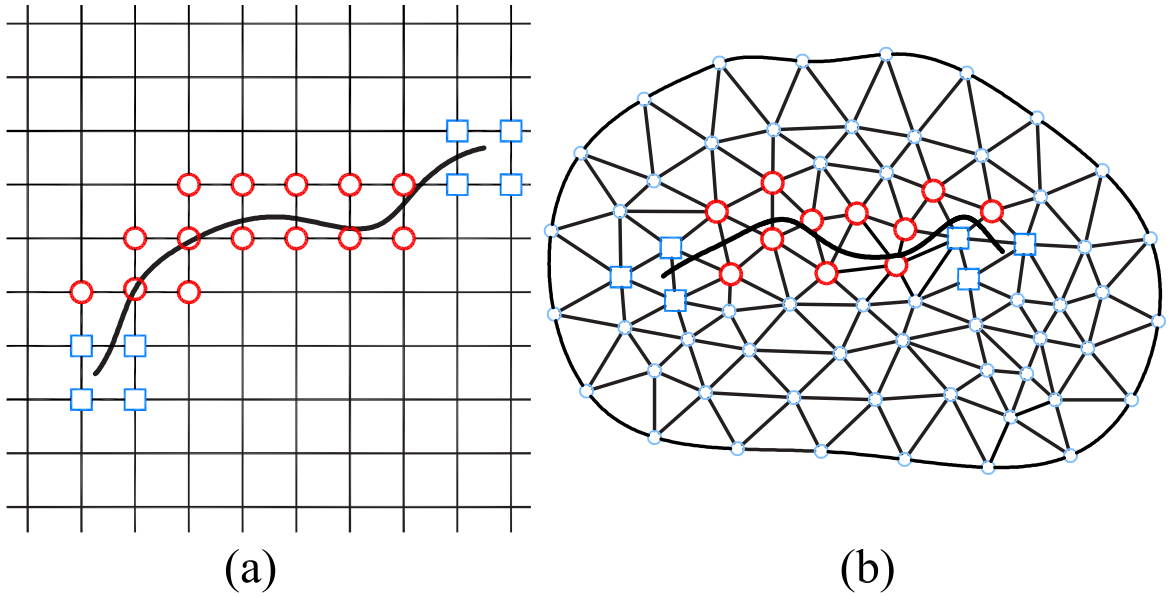


Figure 2.6: Selection of enriched nodes for 2D crack problem. Circled nodes are enriched by the step function whereas the squared nodes are enriched by the crack tip functions. (a) on a structured mesh; (b) on an unstructured mesh.

Here, r and θ are polar coordinates in the local crack tip coordinate system as shown in Figure 2.8. Note that the first function in the above equation is discontinuous across the crack as shown in Figure 2.7. It represents the discontinuity near the tip, while the other three functions are added to get accurate result with relatively coarse meshes. It is not clear from known numerical examples with X-FEM whether adding higher order terms in the asymptotic expansion of the near-tip fields significantly contributes to further enhancement of the solution. Also, as shall be seen in the numerical examples, the choice of the set of nodes to be enriched with the asymptotic near-tip fields has a non-negligeable influence on the results.

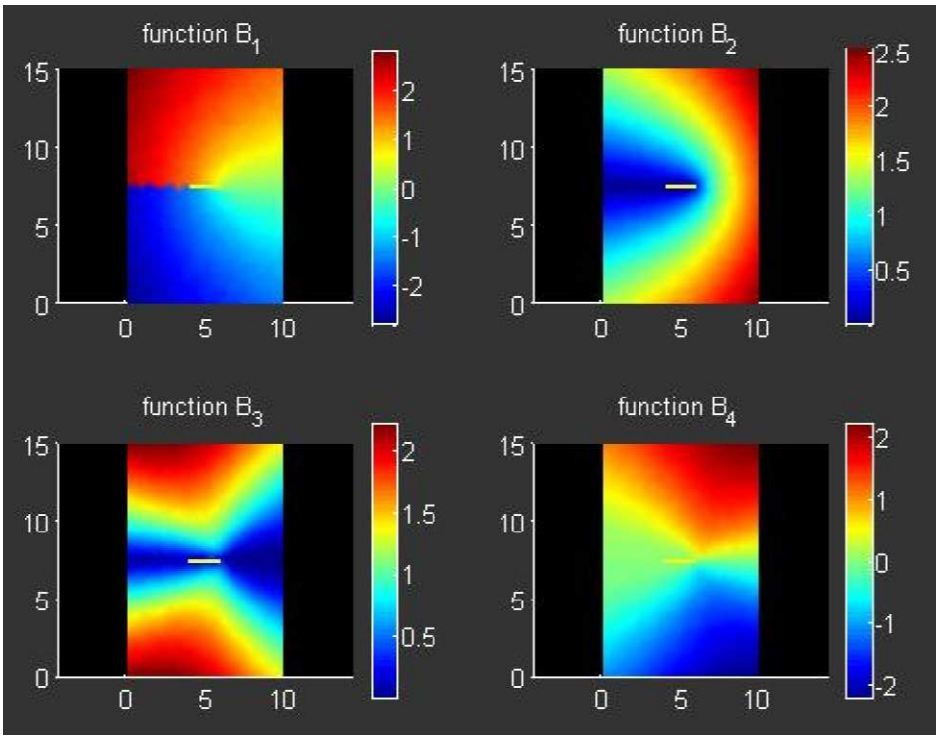


Figure 2.7: 2D view of near tip asymptotic functions

Note that, in equation (2.43), two types of shape functions are used, N_I to interpolate the standard displacement field and $\widetilde{N_J}$ multiplied with the enrichment functions to form the enriched shape functions.

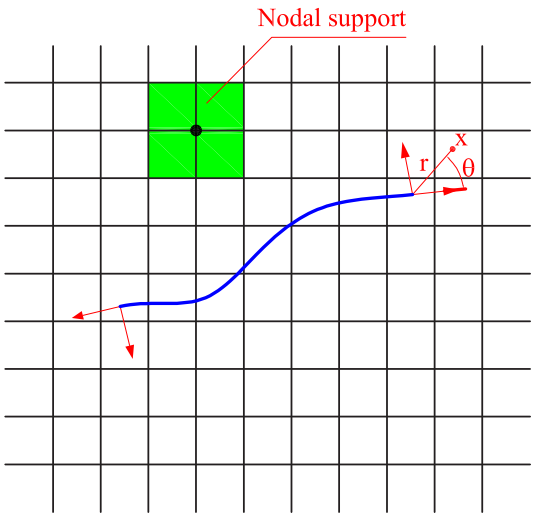
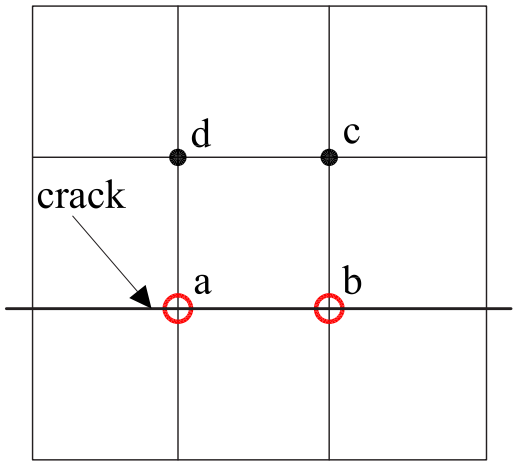
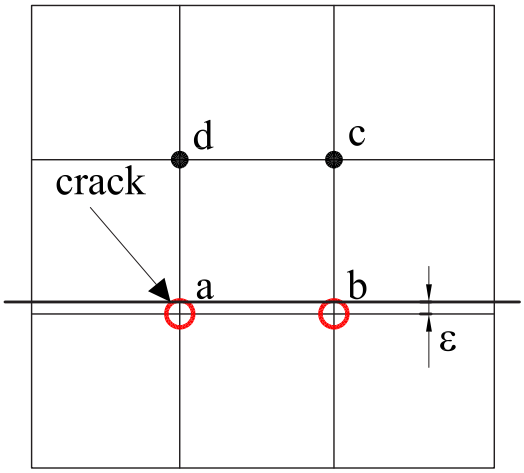


Figure 2.8: Coordinate configuration for crack tip enrichment function

We have defined the set of nodes that are enriched by the Heaviside function as the set of nodes whose support is completely cut by the crack. A direct use of this definition could lead to an ill-conditioned stiffness matrix. Consider Figure 2.9 which illustrates a crack cutting through finite elements. In Figure 2.9(a), nodes a and b are enriched by H function as usual whereas nodes c and d are not enriched since their supports are not cut by the crack. In Figure 2.9(b), a direct application of the support criterion leads to the enrichment of nodes c and d. Hence, the regular and enriched shape functions at these nodes will only differ in the very thin band of width ε, leading to ill-conditioned system matrix because



(a)



(b)

Figure 2.9: Effects of crack near edge: (a) crack aligned with a mesh. The nodes a and b are enriched and the nodes c and d are not enriched; and (b) crack almost aligned with a mesh. Nodes c and d should not be enriched to avoid singular stiffness matrix.

the resulting basis functions are almost identical. Therefore, nodes c and d should not be enriched by H function. The criterion for selection of H enriched nodes, as given in (Dolbow, 1999), relies on the following considerations. For a certain node, the area of its support denoted by A_ω is computed. The part of its support's area above, A_ω^{ab} and below, A_ω^{be} the crack, are computed (see Figure 2.10). Then, the following ratios are calculated

$$r_{ab} = \frac{A_\omega^{ab}}{A_\omega} \tag{2.48a}$$

$$r_{be} = \frac{A_\omega^{be}}{A_\omega} \tag{2.48b}$$

If either of the above ratios are below a specified tolerance, the node is no longer enriched with H. In practice, a tolerance of 10^{-4} is used.

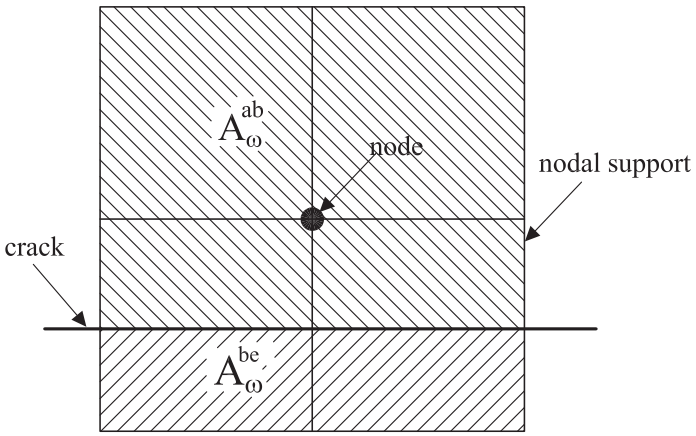


Figure 2.10: Area criterion for selection of $H(\mathbf{x})$ enriched nodes

2.2.4 Discretized equilibrium equations

By substituting the displacement approximation (2.43) into the strain definition (2.31), we arrive at the following expression for the approximated strain:

$$\varepsilon^h = \overline{\mathbf{B}}\overline{\mathbf{u}} = \begin{bmatrix} \mathbf{B}_{\mathrm{I}}^{\mathbf{u}} & \mathbf{B}_{\mathrm{J}}^{\mathbf{a}} & \mathbf{B}_{\mathrm{K}}^{\mathrm{b1}} & \mathbf{B}_{\mathrm{K}}^{\mathrm{b2}} & \mathbf{B}_{\mathrm{K}}^{\mathrm{b3}} & \mathbf{B}_{\mathrm{K}}^{\mathrm{b4}} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\mathrm{I}} \\ \mathbf{a}_{\mathrm{J}} \\ \mathbf{b}_{\mathrm{K}}^{1} \\ \mathbf{b}_{\mathrm{K}}^{2} \\ \mathbf{b}_{\mathrm{K}}^{3} \\ \mathbf{b}_{\mathrm{K}}^{4} \end{bmatrix} \tag{2.49}$$

where $\overline{\mathbf{B}}$ is the discretized symmetric gradient of the extended shape functions. Its components are

$$\mathbf{B}_{\mathrm{I}}^{\mathbf{u}} = \begin{bmatrix} N_{I,x} & 0 \\ 0 & N_{I,y} \\ N_{I,y} & N_{I,x} \end{bmatrix} \tag{2.50}$$

$$\mathbf{B}_{\mathrm{J}}^{\mathbf{a}} = \begin{bmatrix} (\widetilde{N_J}(H - H(\mathbf{x}_{\mathrm{J}})))_{,\mathrm{x}} & 0 \\ 0 & (\widetilde{N_J}(H - H(\mathbf{x}_{\mathrm{J}})))_{,\mathrm{y}} \\ (\widetilde{N_J}(H - H(\mathbf{x}_{\mathrm{J}})))_{,\mathrm{y}} & (\widetilde{N_J}(H - H(\mathbf{x}_{\mathrm{J}})))_{,\mathrm{x}} \end{bmatrix} \tag{2.51}$$

$$\mathbf{B}_{\mathrm{K}}^{\mathrm{bl}}\,|_{\mathrm{l}=1,2,3,4} = \begin{bmatrix} (\widetilde{N_K}(B_K^l - B_K^l(\mathbf{x}_{\mathrm{K}})))_{,\mathrm{x}} & 0 \\ 0 & (\widetilde{N_K}(B_K^l - B_K^l(\mathbf{x}_{\mathrm{K}})))_{,\mathrm{y}} \\ (\widetilde{N_K}(B_K^l - B_K^l(\mathbf{x}_{\mathrm{K}})))_{,\mathrm{y}} & (\widetilde{N_K}(B_K^l - B_K^l(\mathbf{x}_{\mathrm{K}})))_{,\mathrm{x}} \end{bmatrix} \tag{2.52}$$

Substituting the displacement (2.43) and the strain (2.49) into the weak form (2.36), the standard discrete system of equations is obtained:

$$\mathbf{K}\mathbf{u} = \mathbf{f}^{\text{ext}} \tag{2.53}$$

where $\mathbf{f}^{\text{ext}}$ is the vector of external nodal forces and $\mathbf{K}$ is the stiffness matrix:

$$\mathbf{K} = \int_{\Omega^{\text{h}}} \overline{\mathbf{B}}^{\text{T}} \mathbf{C} \overline{\mathbf{B}} \text{d}\Omega \tag{2.54}$$

The expression of the external forces vector $\mathbf{f}^{\text{ext}}$ is as follows

$$\mathbf{f}^{\text{ext}} = \{\mathbf{f}_{\text{I}}^{\mathbf{u}}; \mathbf{f}_{\text{J}}^{\mathbf{a}}; \mathbf{f}_{\text{K}}^{\text{b1}}; \mathbf{f}_{\text{K}}^{\text{b2}}; \mathbf{f}_{\text{K}}^{\text{b3}}; \mathbf{f}_{\text{K}}^{\text{b4}}\} \tag{2.55}$$

with

$$\mathbf{f}_I^{\mathbf{u}} = \int_{\Gamma_t} N_I \overline{\mathbf{t}} \text{d}\Gamma + \int_{\Omega} N_I \mathbf{b} \text{d}\Omega \tag{2.56}$$

$$\mathbf{f}_J^{\mathbf{a}} = \int_{\Gamma_t} \widetilde{N}_J (H - H(\mathbf{x}_{\text{J}})) \overline{\mathbf{t}} \text{d}\Gamma + \int_{\Omega} \widetilde{N}_J (H - H(\mathbf{x}_{\text{J}})) \mathbf{b} \text{d}\Omega \tag{2.57}$$

$$\mathbf{f}_K^{bl} \mid_{l=1,2,3,4} = \int_{\Gamma_t} \widetilde{N}_K (B_K^l - B_K^l(\mathbf{x}_{\text{K}})) \overline{\mathbf{t}} \text{d}\Gamma + \int_{\Omega} \widetilde{N}_K (B_K^l - B_K^l(\mathbf{x}_{\text{K}})) \mathbf{b} \text{d}\Omega \tag{2.58}$$

2.2.5 Element integration

In the derivation of the weak form, the divergence theorem is used to lower the continuity requirements on the trial functions. The divergence theorem is only applicable on a domain on which u_i sufficiently regular, i.e., u_i must not contain discontinuities or singularities, and hence the crack surface must be an internal boundary of the domain of integration. For example, for element ABCD in Figure 2.11, it is necessary to divide this element into two sub-domains ABGFE and EFGCD and do the integration on these two sub-domains. The design and development of quadrature rules over arbitrary polygons has not yet reached a mature stage. Therefore, instead of performing the integration over polygonal domains, it these domains are partitioned into sub-triangles, see Figure 2.11. To get accurate results, in the sub-triangles, high order Gauss quadrature rule must be used. In this thesis, 13 Gauss points are used in each sub-triangle. It is worthwhile to note that researchers have come up with singular numerical integration techniques (Laborde, Pommier, Renard, and Salaun, 2004).

The numerical integration procedure for elements cut by the crack is as follow

1. Build the Delaunay triangulation to get the sub-triangles
2. For each sub-triangle, the coordinates and weights of 13 Gauss points are computed and then converted into the parent coordinate system of the original element.

Details are given in Appendix C.

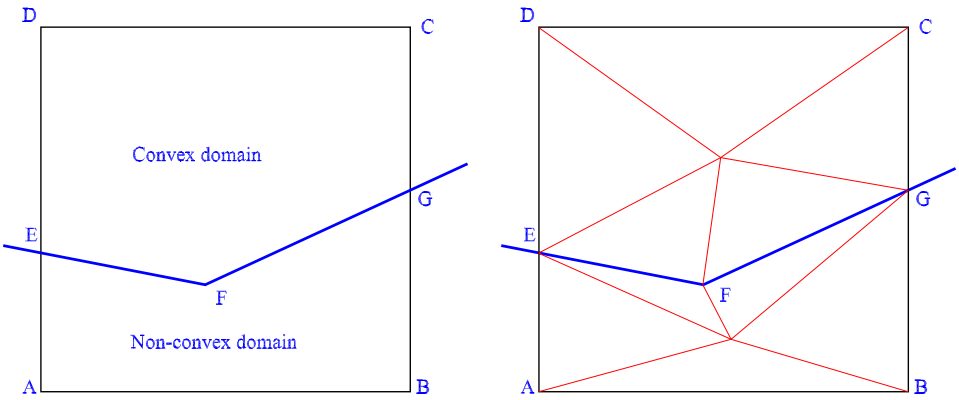


Figure 2.11: Subdomains used in integration on polygon

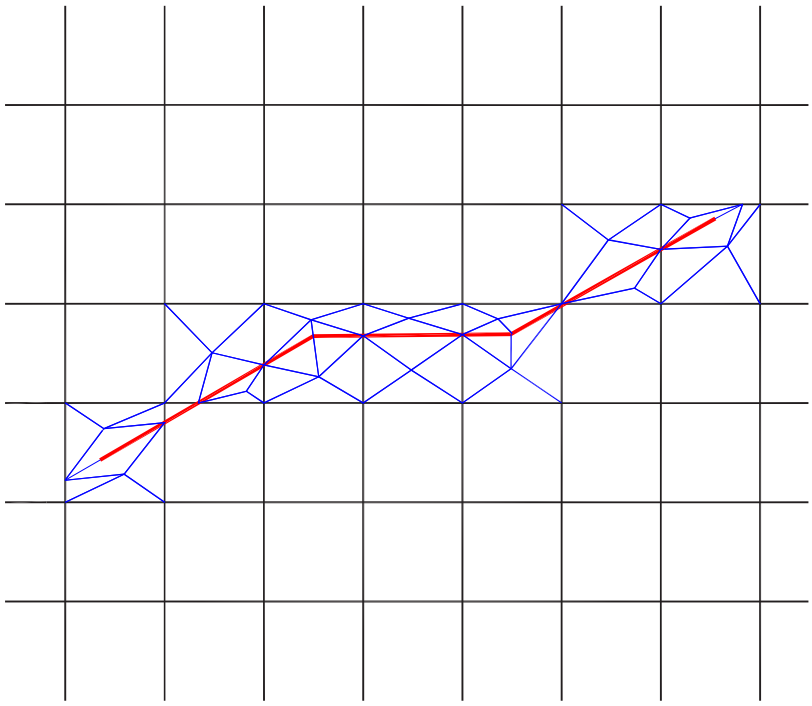


Figure 2.12: Sub-triangles used in numerical integration

2.2.6 Mesh geometry interaction

In the X-FEM, the discontinuities are not explicitly meshed but represented implicitly by enriching some nodes with appropriate enrichment functions. Therefore, one of the first tasks is to determine the finite elements that intersect the discontinuities.

The X-FEM usually has been used in conjunction with the Level Set Method, a technique introduced by Osher and Sethian (1988) to solve a variety of applied mechanics problems: crack growth in 2D (Stolarska et al., 2001) and 3D planar cracks (Sukumar et al., 2000; Sukumar et al., 2003) and Chopp and Sukumar (2003) where the problem of multiple planar cracks is examined. The hybrid method was also recently employed for non-planar crack growth (Moës et al., 2002; Gravouil et al., 2002), and in other fields of physics: solidification problems (Chessa et al., 2002; Ji et al., 2002; Merle and Dolbow, 2002), multi-phase flow (Chessa and Belytschko, 2003a; Chessa and Belytschko, 2003b) and chemically-induced swelling of hydrogels (Dolbow et al., 2003) and biofilm growth (Bordas, 2003; S. Bordas and Chopp, 2005). However, it was chosen for the work of this thesis to represent the crack by standard geometrical predicates –points, segments, polylines, etc.. In the following, the computational geometry techniques developed to find out elements interacting the crack, and updating the enriched nodes are discussed in detail. The interested reader can refer to Sukumar and Prévost 2003 for a similar presentation of these computational geometry issues.

In the current implementation, the orientation test is used to determine the nodal enrichment for a query point $\mathbf{x}$. For a query point $\mathbf{x} \equiv (\mathbf{x}, \mathbf{y})$, we consider the triangle with vertices $(\mathbf{x_1}, \mathbf{x_2}, \mathbf{x})$, where $\mathbf{x_1} \equiv (\mathbf{x_1}, \mathbf{y_1})$ and $\mathbf{x_2} \equiv (\mathbf{x_2}, \mathbf{y_2})$ are

coordinates that define one crack segment. Twice the signed area of this triangle is evaluated:

$$\Delta = (x_1 - x)(y_2 - y) - (x_2 - x)(y_1 - y) \tag{2.59}$$

$\mathbf{x}$ is above the crack if $\Delta > \varepsilon$, below the crack if $\Delta < -\varepsilon$ and is on the crack otherwise. In the code, a tolerance $\varepsilon = 10^{-6}$ was used.

To detect whether an element e is completely cut by the crack, equation (2.59) is used for all nodes of e. If the number of nodes which locate above or below the crack is different from the total number of nodes of e, then e is cut by the crack.

In order to find out if an element e contains the crack tip or not, equation (2.59) is used for every edges of e.

Normally, a fracture mechanics problem solved by the X-FEM consists of a fixed underlying finite element mesh and cracks approximated as connected sets of straight line segments. To find out elements (convex polygons) cut by the cracks (line segments) and elements containing the tips (2D points), firstly, we loop on all elements and detect sets of elements denoted by A_Γ cut by each crack. Then, each crack will find out elements containing its tips by looping on set A_Γ and using the aforementioned geometry predicates. Obviously, for multiple cracks and a very fined mesh, the mesh geometry interaction procedure using the previous way is computationally costly. To improve this, a good mesh database should be used and AOMD (Algorithm Oriented Mesh Database) (Remacle, Karamete, and Shephard, 2000) is a promising choice. In the results presented in sections on numerical experiments, the paradigm of Dunant et al. (2005) is used, and the mesher is incorporated into the solver, so as to best leverage the information needed to construct the mesh at solve time.

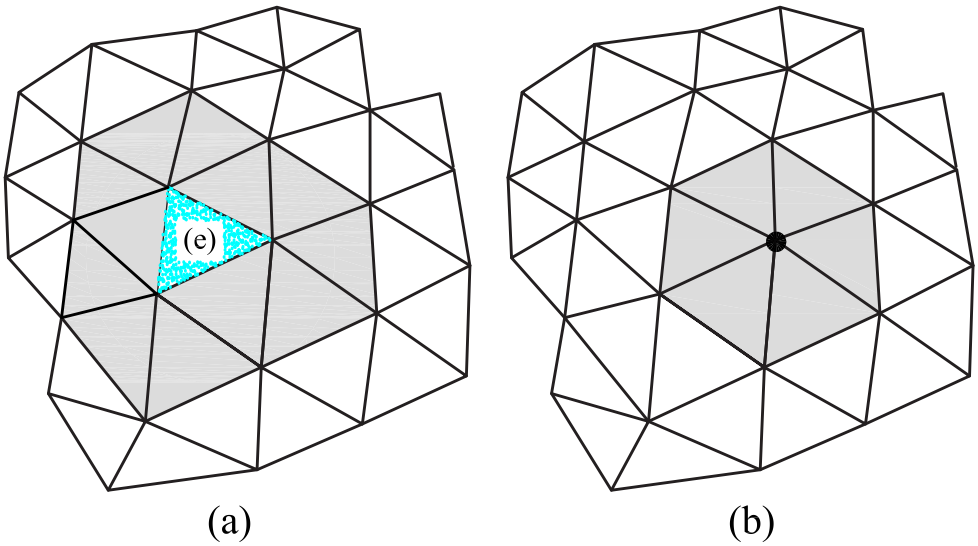


Figure 2.13: Neighbors of finite elements (a) and of a node(b)

The computation of the interaction integral requires a set of finite elements cut by a contour line –usually a circle centered at the crack tip, for convenience. To this end, neighboring elements (see Figure 2.13) of the tip element are detected. For each neighboring element, say e, it is checked whether this element is cut by the circle. If so, the same operation is repeated for the elements neighboring e. The neighbors of an element are built from the supports of its nodes. If the elements are stored in a tree (Delaunay tree for a triangular mesh, Quad tree for a quadrangular mesh, octree for a 3D hexahedral mesh), the interaction between the circle and the mesh is much less costly, as shown in Dunant et al. (2005).

2.2.7 Crack growth simulation

This section introduces the quasi-static crack growth simulation in 2D domains using the X-FEM. Since the crack is described independently of the mesh, no remeshing is needed at each stage of crack advance.

In the present formulation, the crack is represented as a connected set of straight line segments as shown in Figure 2.14. It is necessary to compute the propagation angle θ_c and increment length Δa for the new segment. The propagation angle θ_c is determined by (see Section 2.1.3)

$$\theta_c = 2\arctan\left[\frac{1}{4}\left(\frac{K_I}{K_{II}} - sign(K_{II})\sqrt{\left(\frac{K_I}{K_{II}}\right)^2 + 8}\right)\right] \tag{2.60}$$

If $K_{II} = 0$ then $\theta_c = 0$ (pure mode I) and by noting that if $K_{II} > 0$, the crack growth angle $\theta_c < 0$, and if $K_{II} < 0$ then, $\theta_c > 0$, a more efficient expression for θ_c

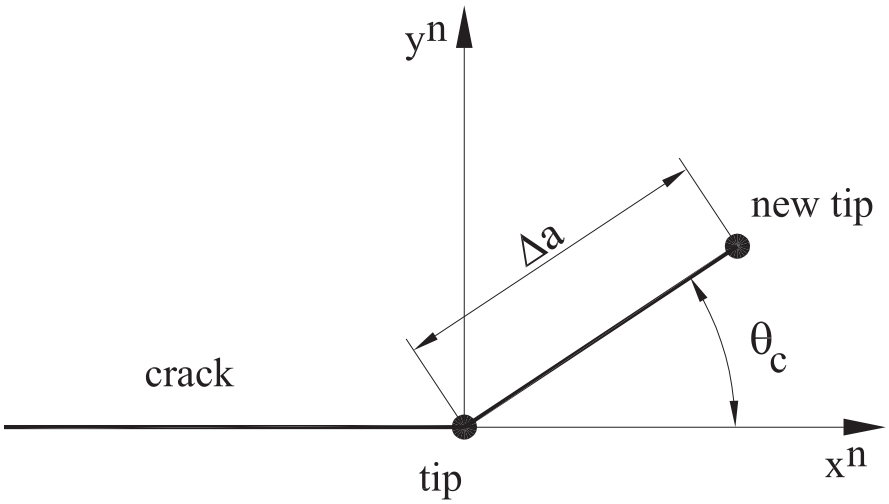


Figure 2.14: Discrete representation of crack segments

is implemented (Suo, 2002):

$$\theta_c = 2\arctan\left[\frac{-2K_{II}/K_I}{1+\sqrt{1+8(K_{II}/K_I)^2}}\right] \tag{2.61}$$

The increment crack length can be defined by Paris law or set in advance. In this thesis, the latter way was chosen.

The procedure to simulate the crack growth with the X-FEM is summarized in Figure 2.15. Initially, a relatively coarse mesh is used to obtain a rough estimate of the overall crack path. We then refine the mesh in the vicinity of the crack path and use a smaller crack increment $\triangle a$ to obtain a converged solution.

In the following, some issues concerning the efficient implementation of crack growth simulations are discussed. When the crack grows, some nodes change their enrichment status, for instance, from being tip-enriched to being enriched with the step function; some non-enriched nodes become enriched as shown in Figure 2.16. To quickly find out these nodes, a circle centered at the old tip with a radius $\triangle a$ is built. Elements belonging to this circle (shaded elements in Figure 2.16) are found. Finally, by looping on these elements and detecting those that intersect the crack advance line segment yields the newly enriched nodes.

It is not efficient to recompute the stiffness matrices of all elements at each time step. Therefore, a set of elements denoted by A (hashed elements in Figure 2.17) is built. As one could see in this figure, set A is composed of nodal supports of new enriched nodes. Elements belonging to this set must recompute the following:

- Gauss points used to compute the stiffness matrices
- Stiffness matrix

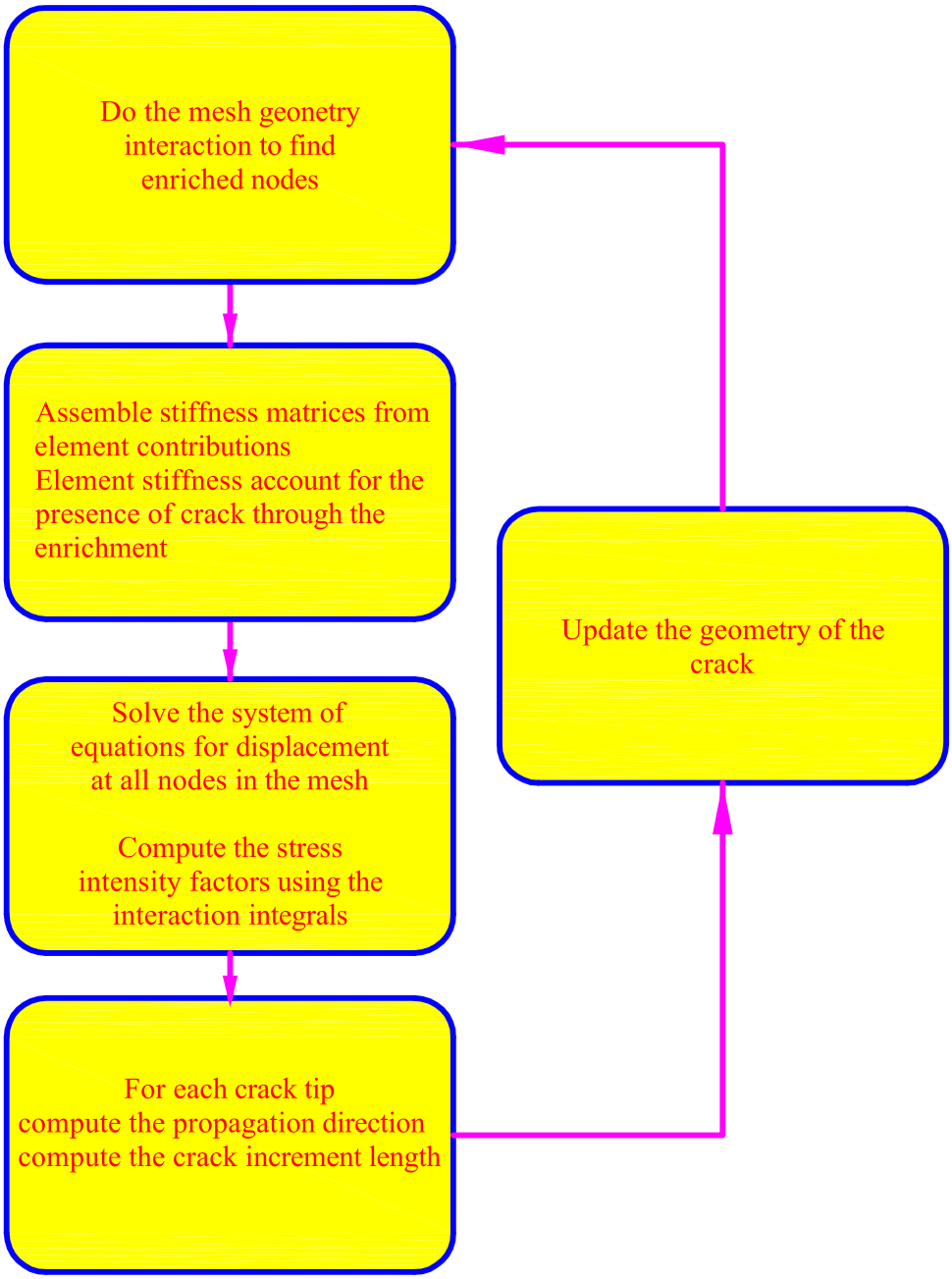


Figure 2.15: The X-FEM flowchart for crack growth simulation

- Location array (scatter array) used to do the assembly

Another possibility for the assembly is to use the static condensation technique. Since the standard degree of freedoms (DOFs) are fixed, we can keep them and condense all those DOFs onto enriched DOFs. It is just a proposal and requires further work to make sure it makes sense. Another possibility would consist in condensing the enriched dofs onto the standard dofs locally, in each finite element.

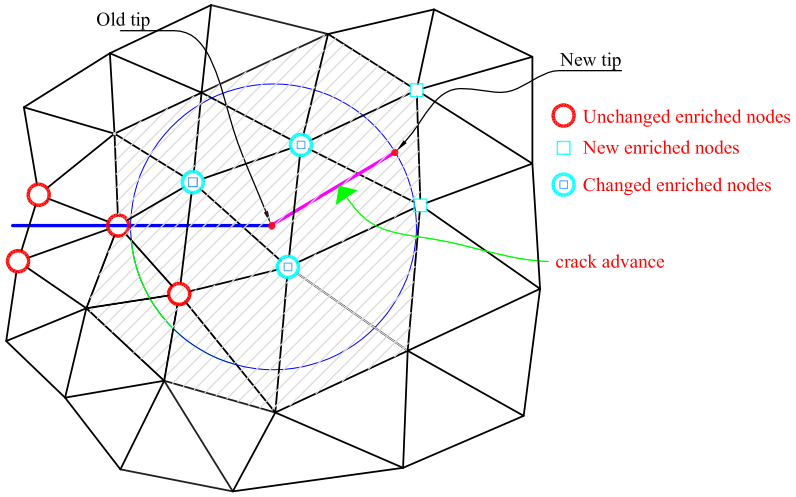


Figure 2.16: Elements (shaded) used to find new interacted elements

2.3 Improvement of the rate of convergence of the X-FEM

Although the X-FEM ensures a lower error than the classical finite element methods, the rate of convergence is not improved when the mesh parameter h is going to

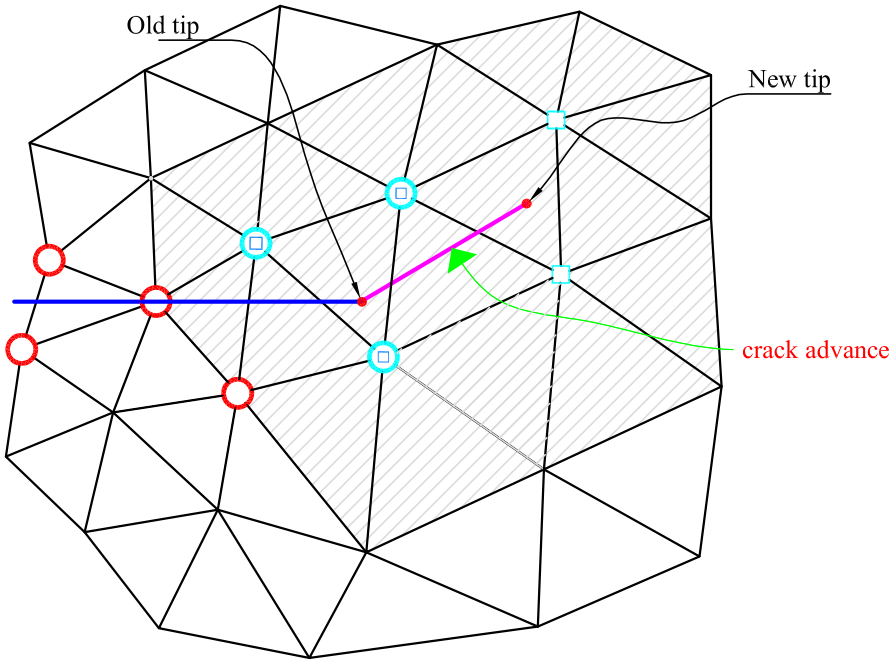


Figure 2.17: Elements whose status is changed after a crack grows

zero because of the presence of a singularity. This rate is lower than it is expected with classical finite element method for a smooth problem as pointed out in Stazi et al. 2003. In order to obtain the optimal accuracy, some methods have been proposed such as X-FEM with a fixed enrichment area, high order X-FEM and the construction of blending elements.

2.3.1 X-FEM with a fixed enrichment area

The usual enrichment scheme is to enrich only nodes of tip-elements. Consequently, the support of the enriched functions vanishes when h goes to zero. An alternative strategy, introduced in Laborde et al. 2004, consists in enriching a fixed area around the crack tip independently of h. Defining a circle $C(x_0, R)$ with predefined radius r whose center is the crack tip x_0. Then any node belonging to this circle is enriched with the crack tip functions (see Figure 2.18). The same enrichment scheme is to enrich a layer of elements around the tip-element.

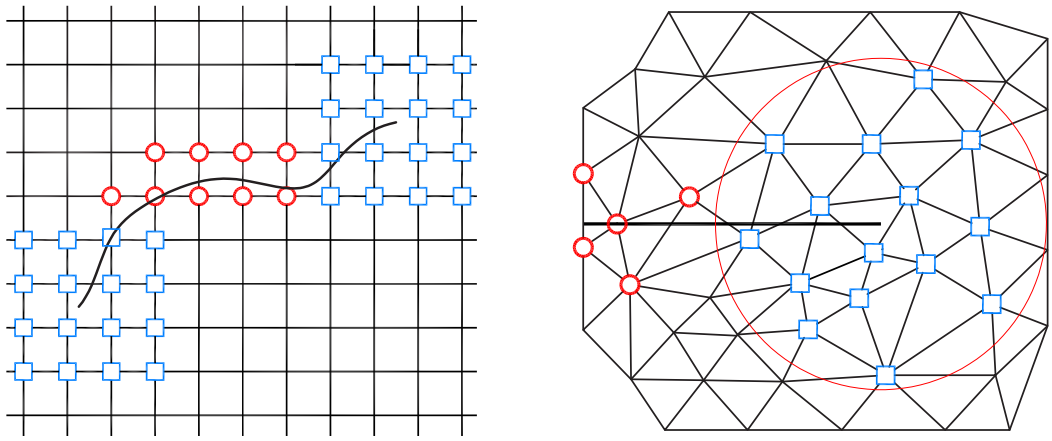
The new enriched approximation is written as

$$\mathbf{u}^h(\mathbf{x}) = \sum_{I\in\mathcal{N}} N_I(\mathbf{x})\mathbf{u}_I + \sum_{J\in\mathcal{N}^{disc}} \widetilde{N}_J(\mathbf{x})H_J(\mathbf{x})\mathbf{a}_J + \sum_{K\in I(R)} \widetilde{N}_K(\mathbf{x}) \sum_{\alpha=1}^{4} B_{\alpha K}(\mathbf{x})\mathbf{b}_{\alpha K} \tag{2.62}$$

where the nodal set $I(R)$ contains nodes locating inside the circle $C(x_0, R)$.

2.3.2 High order extended finite element method

It is well known that high order elements provide improved accuracy for sufficiently smooth problems. Due to higher rate of convergence, the decreased susceptibility to locking, and the ability to model curved boundaries, quadratic elements have



(a) (b)

Figure 2.18: Another scheme for selection of enriched nodes for 2D crack problem. Circled nodes are enriched by the step function whereas the squared nodes are enriched by the crack tip functions.

been elements of choice for most static and quasi-static problems. And X-FEM is not an exception. In Stazi et al. 2003, the high order enriched finite element method was presented and used to solve multiple crack problems with great success. This section aims at introducing this useful extension of the X-FEM.

For ease of reading, the extended finite element approximation is reintroduced

$$\mathbf{u}^h(\mathbf{x}) = \sum_{I\in\mathcal{N}} N_I(\mathbf{x})\mathbf{u}_I + \sum_{J\in\mathcal{N}^{disc}} \widetilde{N}_J(\mathbf{x})H_J(\mathbf{x})\mathbf{a}_J + \sum_{K\in\mathcal{N}^{asympt}} \widetilde{N}_K(\mathbf{x}) \sum_{\alpha=1}^{4} B_{\alpha K}(\mathbf{x})\mathbf{b}_{\alpha K} \tag{2.63}$$

where the shape functions N_I are quadratic whereas $\widetilde{N}_J$, $\widetilde{N}_K$, shape functions that construct the enriched part of the approximation, are chosen to be linear shape functions. The reason for this choice was explained in Chessa et al. (2003), which

is recalled in the next section. Note that, with quadratic elements, and if the finite element shape functions are used to model the crack geometry –using the level set method for instance– the curved cracks can be modeled more efficiently and accurately than with linear XFEM. The enriched nodes for unstructured quadratic triangular mesh is given in Figure 2.19.

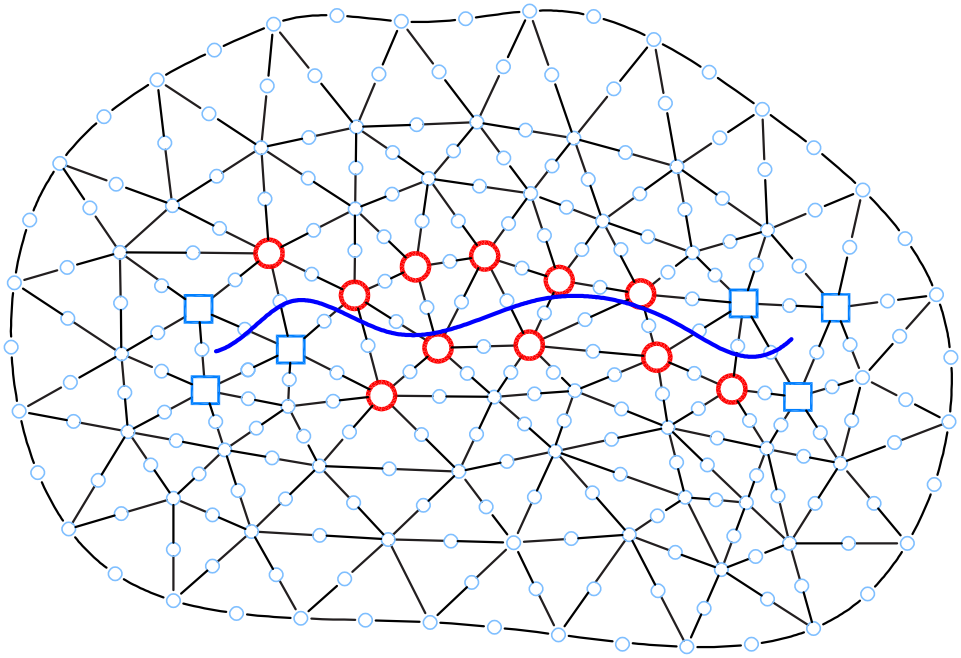


Figure 2.19: Enriched nodes for quadratic XFEM. Circled nodes are enriched by the step function whereas the squared nodes are enriched by the crack tip functions.

2.3.3 X-FEM - a local partition of unity enriched finite element method

The finite element shape functions form a partition of unity

$$\sum_{I \in N} N_I(\mathbf{x}) = 1 \tag{2.64}$$

It follows from the above that for an arbitrary function $\Psi(\mathbf{x})$, the following satisfies

$$\sum_{I \in N} N_I(\mathbf{x})\Psi(\mathbf{x}) = \Psi(\mathbf{x}) \tag{2.65}$$

Therefore any function Ψ can be reproduced by a set of functions $N_I\Psi$. This is the key property of enriched finite element methods based on a partition of unity.

Although one could enrich the entire domain, only a sub-domain is usually enriched since the features need to be modeled local – for instance, a crack compared to the plate containing it. Moreover, keeping enrichment local permits keeping the matrix banded. This is why X-FEM can be considered as *a local partition of unity enriched finite element method*. A partitioning of a typical domain into its non-enriched sub-domains and enriched sub-domains is shown in Figure 2.20.

In this local enrichment scheme, three types of elements are distinguished. The first types are the classical finite elements, those in which none of its nodes are enriched, these elements are grouped in Ω^{std}. The second type are fully enriched elements, i.e., all of its nodes are enriched. These elements are denoted as Ω^{enr}. The third type of elements, called partially-enriched elements, are those for which only some –but not all– of the nodes are enriched. These elements form the Ω^{blnd}

sub-domain.

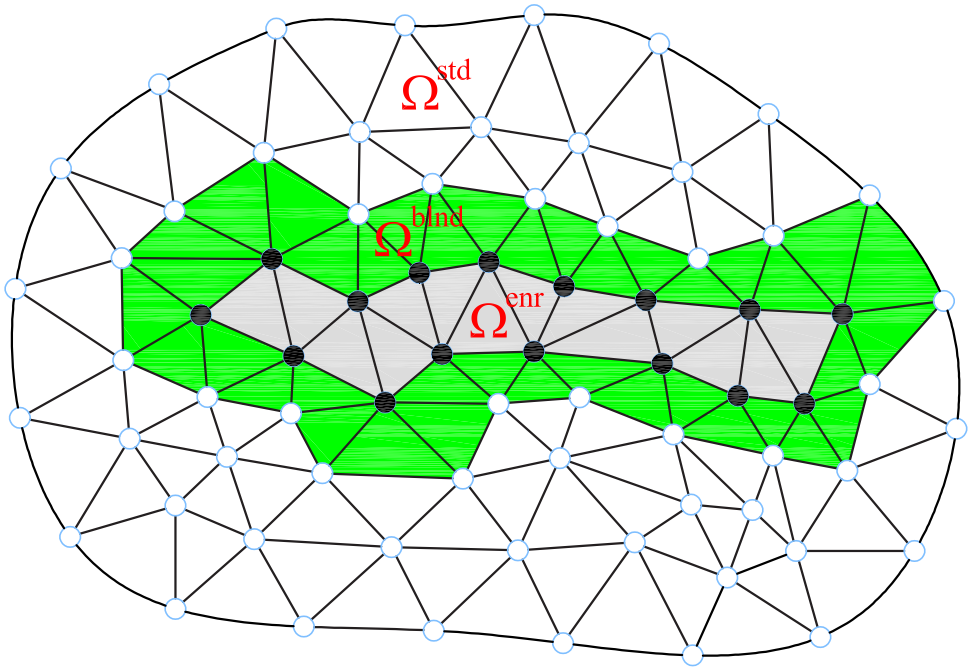


Figure 2.20: Typical discretization illustrating enriched domain, Ω^{enr}, transition domain, Ω^{blnd}, as well as enriched nodes(filled nodes).

Let $\mathbf{u}_I = 0$ and $\mathbf{a}_J = 1$ in the enriched finite element approximation (2.40), we have

$$\mathbf{u}^h(\mathbf{x}) = \sum_{J \in N^{enr}} \begin{cases} \widetilde{N}_J(\mathbf{x})\Psi(\mathbf{x}) = \Psi(\mathbf{x}) & \forall \mathbf{x} \in \Omega^{enr} \\ \widetilde{N}_J(\mathbf{x})\Psi(\mathbf{x}) \neq \Psi(\mathbf{x}) & \forall \mathbf{x} \in \Omega^{blnd} \\ \widetilde{N}_J(\mathbf{x})\Psi(\mathbf{x}) = 0 & \forall \mathbf{x} \in \Omega^{std} \end{cases} \tag{2.66}$$

Therefore the approximation can reproduce the enrichment in Ω^{enr} and it vanishes in Ω^{std}. However, in the blending domain, it consists of the product of a subset of the enriched shape functions $\widetilde{N}_J$ and the enrichment function Ψ so this enrichment

function cannot be reproduced. The *blending elements* or *transition elements* lead to a lower convergence rate for enriched finite element methods compared to standard finite element methods. The following example, extracted from Chessa et al. (2003), show why this is the case.

Note that relatively simple cures for this problem have been developed in (Chessa et al., 2003).

Consider a one dimensional mesh as shown in Figure 2.21 with a discontinuity in the derivative in element 0 as shown. The enrichment function is the ramp function (Sukumar et al., 2001)

$$\Psi(\mathrm{x}) = xH(x) \tag{2.67}$$

where H is the Heaviside step function. This enrichment adds a discontinuity in the gradient of the approximation at $x = 0$ (Sukumar, Chopp, Moës, and Belytschko, 2001). Linear shape functions are used for both the standard approximation and the partition of unity. Let element 0 be the fully enriched element and element 1 be the blending element to the right.

The approximation of element 1 is given by

$$\mathbf{u}^h(\mathrm{x}) = \sum_{I=1}^{2} N_I(\mathrm{x}) + N_1(\mathrm{x})(\mathrm{x}H(x) - \mathrm{x}_1 H(x_1))q_1 \tag{2.68}$$

$$\mathbf{u}^h(\xi) = u_1(1-\xi) + u_2\xi + q_1\xi h(1-\xi) \tag{2.69}$$

where

$$\xi = \frac{x - x_1}{h} \tag{2.70}$$

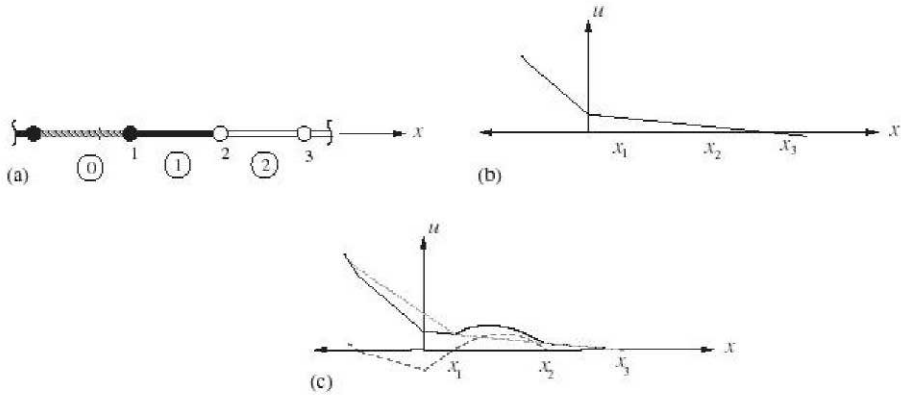


Figure 2.21: A 1D example of how a locally enriched finite element method fails to be able to reproduce a linear field. The desired piecewise linear field is shown in (b) and the enriched part (*dotted line*), standard part (*dashed line*), and the total (*solid line*) approximation are shown in (c). The discretization is shown in (a), where the enriched nodes denoted by filled circles, the two blending elements are filled and the fully enriched element is hashed (Chessa, Wang, and Belytschko, 2003).

and h is the length of element 1.

Let the finite element interpolation to the solution be given by u^{int} and denote the error in the interpolation by Δ, we have

$$\Delta \equiv u - u^{int} \tag{2.71}$$

The maximum error occurs at the point $\overline{x}$ where

$$\Delta_{,x|\overline{x}} \equiv \frac{d}{dx}\Delta(\overline{x}) = 0 \tag{2.72}$$

Then a Taylor expansion about $\overline{x}$ gives

$$\Delta(x) = \Delta(\overline{x}) + \Delta_{,x|\overline{x}}(x - \overline{x}) + \frac{1}{2}\Delta_{,xx|\overline{x}}(x - \overline{x})^2 + O(h^3) \tag{2.73}$$

or

$$\Delta(x) = \Delta(\overline{x}) + \frac{1}{2}\Delta_{,xx|\overline{x}}(x - \overline{x})^2 \tag{2.74}$$

If we let $x = x_1$, then $\Delta(x_1) = 0$ since u^{int} is the finite element interpolation of u, i.e., $u^{int}(x_I) = u(x_I)$. Therefore, from (2.74) we obtain

$$\Delta(x) = -\frac{1}{2}\Delta_{,xx|\overline{x}}(x - \overline{x})^2 \tag{2.75}$$

Since from (2.69)

$$\Delta(x) = u_{,xx} + \frac{2q_1}{h} \tag{2.76}$$

and since

$$\frac{1}{2}(x - x_1)^2 \leq \frac{1}{8}h^2 \tag{2.77}$$

it follows that

$$\Delta(\overline{x}) \leq \frac{1}{8}h^2 \max(u_{,xx} + \frac{2q_1}{h}) \tag{2.78}$$

The last term, $2q_1/h$, does not appear for standard finite elements. It increases the interpolation error in the blending elements from order h^2 to h. Although this occurs in only few elements, the effect is to reduce the rate of convergence of the entire approximation. The reason for this is that the partition of unity property (completeness) of the approximation is not verified in the whole domain. Therefore, the theoretical rate of convergence cannot be attained.

When the enrichment is a polynomial of order n, i.e., ξ^n, then for $n > 1$ the interpolation error in the blending elements is increased even further. If we go through the same steps as before, we find that

$$\Delta(\overline{x}) \leq \frac{1}{8}h^2 \max(u_{,xx} + \frac{2q_1}{h^n}) \tag{2.79}$$

At this time, we can understand why in the quadratic X-FEM formulation, the partition of unity shape functions have been chosen as linear. To get improved rate of convergence, Chessa proposed the enhanced strain formulation for the blending elements. By properly choosing an enhanced strain field, we can eliminate the unwanted terms in the enriched approximation. More details on this method can be found in Chessa (2002) and Chessa, Wang, and Belytschko (2003).

2.4 Conclusions

The first part of this chapter presented the basics of fracture mechanics and showed how domain form of interaction energy integrals can be used, in the context of Linear Elastic Fracture Mechanics, to compute numerical fracture parameters for cracks in 2D.

This chapter focuses on the presentation of the application of the X-FEM to linear elastic fracture mechanics. Essential tasks in the X-FEM include *the choice of enrichment functions, how to find out enriched nodes and how to deal with the integration of non-polynomial and discontinuous functions.*

In the context of LEFM, the enrichment functions can be easily found –derived from the asymptotic displacement fields. However, for ductile materials, the LEFM is no longer correct and the Elasto-Plastic Fracture Mechanics (EPFM) should be used. In this case, it is not straightforward to find out which enrichments are to be used. Bordas (2001) propose a technique where the asymptotic fields obtained from the Hutchinson-Rice-Rosengren (HRR) fields of elastic-plastic Ramberg-Osgood materials enrich the standard approximation. However, it is pointed out that only the radial dependence of the asymptotic expansion of the near-tip fields are known analytically and that the angular variation of the fields can be computed numerically by a one-dimensional finite element method.

Mesh-geometry interaction is a major part of the X-FEM. The geometry of the discontinuities can be represented by standard geometry (as done in this thesis) or by the level sets. Recently, the vector level set was proposed (Ventura, Xu, and Belytschko, 2001), for crack growth problems, as a simpler alternative to the standard level set method since no hyperbolic conservation law needs to be

solved in the vector level set method. Indeed, hyperbolic conservation laws lead to instabilities when discretized using Galerkin methods and require those methods to be stabilized. In addition, representing cracks requires using two orthogonal level set functions. As they evolve, these two functions need to be re-orthogonolized and re-initialized to signed-distance functions (Bordas, 2003; Moës, Gravouil, and Belytschko, 2002; Gravouil, Moës, and Belytschko, 2002).

Numerical integration of the weak form in the X-FEM is an interesting research topic. The frequently used approach is that elements split by the discontinuities are partitioned into subtriangles and instead of performing the integration on the element, each subtriangle is given a large number of standard integration points. This method, although effective, may not be the most practical nor the most efficient for problems with non-linear material laws as pointed out in (Dolbow, 1999). The difficulty involved in the discontinuous Gauss quadrature emanates from the singularity at the crack tip. To remove this singularity, some methods have been developed recently such as the polar integration (Laborde et al., 2004) or the singular mapping (Béchet, Minnebo, Moës, and Burgardt, 2005).

It is emphasized that, thanks to asymptotic enrichment, accurate results, say SIFs, areobtained with coarse meshes, remeshing is not necessary to model crack growth, a certain level of refinement is needed to get exact crack trajectory. The X-FEM combined with an automatic remeshing algorithm around the crack tip is a promising tool for complex crack simulations, especially in 3D, but is still an open question.

Chapter 3

Object oriented enriched finite element implementation

3.1 Introduction

Numerical methods play an important role to solve a variety of problems from those in academic environment to complex industrial ones. Powered with developments in computer softwares and hardware, domains of problems has been greatly extended to a level where analysis of a vast variety of challenging problems are successfully handled. To this end, software development becomes a critical issue to manage this demand. Conventional development approaches follow procedural techniques driven by Fortran or C programming languages. However, this technique requires high manageability costs with increasing complexity of engineering problems. Frequent code duplications, high couplings between modules, loose control on data security, severe penalties in global code management efforts and lack of flexibility for integrating new modules are among shortcomings of these techniques. An alternative solution to overcome these problems is object-oriented programming (OOP) that has been gaining popularity in the computational mechanics.

Forde BWR (1990) laid out essential abstractions for the applications of finite element analysis, Archer(1999) have been showed the advantages of object-oriented finite element analysis.

The goal of the thesis is to implement a reusable enriched FE code in which the addition of new enrichment functions, for instance, is seamless. It should be possible for a future developer to very simply add a new problem to the list of problems that can be solved by the code.

The outline of the remainder of this chapter is as follows. Section 3.2 briefly introduces the basic of the object oriented programming. The next section, Section 3.3 presents, in detail, the implementation of the X-FEM using an object oriented approach. The necessary classes to be added for incorporating concepts of the X-FEM into an available FE package is stated together with the modification of original FE classes. This section also points out that the extension of the code, to add new formulations, is not a major obstacle. At the end of the chapter, some conclusions are given.

3.2 Object-oriented programming

Abstraction and Encapsulation

One of the most important characteristics of OOP is abstraction. A typical abstraction for the finite element applications in solid mechanics can be *nodes, elements, loads, materials* and *nonlinear solution algorithms*. Another important merit of object modeling is *encapsulation* that provides a strict control on data. Access to data is restricted to certain objects so that data management is provided at the level of objects, not at the level of global definitions which are mostly em-

ployed in the procedural methods. This restrictive but efficient approach provides a data security and stable coding environment.

Inheritance

One of the most compelling features about C++ is code reuse. There are often two ways to accomplish this. The first is quite straightforward : we simply create objetcs of the existing class inside the new class. This is called composition because the new class is composed of objetcs of existing classes. For instance, a finite element object (an instance of class **Element**) stores its integration points (instances of class **GaussPoint**).

Another key feature of OOP is the possibility to create a new class of objects as a specialization of an existing class. This is called *inheritance*. For instance, a class for four-noded quadrilateral elements (Q4) would *derive from* the (so called 'base') **Element** class. The Q4 class (derived class) would share with the **Element** class its *protected* data members (for instance the array holding the list of nodes). The derived class specializes the base class in so far as it may implement methods (tasks) existing in the base class differently from this base class. More importantly, the derived class may be able to perform tasks that the base class cannot (such as computing the Jacobian matrix of the element).

Polymorphism

Polymorphism is the ability of the same pointer can be used to manipulate objects of several different forms. A programmer may declare a pointer that can validly point to any object from a set of classes in an inheritance tree. Using that pointer, all such objects can be manipulated equivalently without the knowledge of their particular class. Polymorphism is implemented in C++ with virtual functions

(declared with **virtual** keyword).

In fracture mechanics applications of X-FEM for instance, if a node is enriched by the Heaviside function and its support is split by the crack (object of class **EnrichmentItem**) into two parts whose areas are too small then this node should not be enriched anymore so that the resulting matrix is not singular. This is reflected in the following code

```
void Domain :: resolveLinearDependencyForEnrichment() {
 for(i = 1 ; i <= this -> giveNumberOfEnrichmentItems() ; i++)
    this -> giveEnrichmentItem(i) -> resolveLinearDependency() ;
}
```

We just defined the method *resolveLinearDependency()* as pure virtual method in the base class **EnrichmentItem** and only class **CrackInterior** implements this method.

Template

Inheritance provides a way to reuse object code. The template feature in C++ provides a way to reuse source code. A typical example on template, more precisely, function template is the one that returns the maximum of two values:

```
template <typename T>
inline T const& max (T const& a, T const&b){
   return a < b ? b : a;
}
```

This template defines a family of functions that returns the maximum of two values of arbitrary types which can be compared using the operator $<$.

Generic programming and STL

One reason for the success of C++ is that, today a large number of libraries is available which greatly facilitate code development because they offer reliable and well-proven components. A particularly carefully constructed library is the *Standard Template Library* (STL), which has been developed at Hewlett-Packard by Alexander Stephanov, Meng Lee, and their colleagues.

The emphasis of the STL is on data structures for containers, and the algorithms that work with them. A good reference on STL is the book written by Ulrich Breymann (Breymann, 2002).

In the present implementation, we made extensive use of the STL containers (vector, list, valarray, map, etc.), along with their associated iterators and algorithms. More importantly, we devised the so-called "function objects" or "functors". Functors are objects which behave like functions but have all the properties of objects. They can be generated, passed as arguments, or have their state modified. The change of state allows a flexible application which, with functions, would be only possible via additional parameters. Functors are especially useful to be used with the STL algorithms. As an illustration, we gave here one example extracted from Breymann (2002) : Considering a sequence of numbers, the first odd number is sought, with the criterion odd checked by means of the functor in Figure 3.1 :

```
class odd{
  public:
     bool operator()(int x){return x % 2;}
};
```

Figure 3.1: Example of a function object

A sequence of numbers is stored as std::vector<int> v. Then, to find the first odd number, just use the STL algorithm *find_if* :

```
std::vector<int>:const_iterator it =
  std::find_if(v.begin(),v.end(),odd())
```

A great reference on C++ is the book "The C++ Programming Language" (Stroustrup, 2002).

3.3 Object-oriented enriched finite element implementation

The process of object-oriented design involves two main steps. First, the key concepts in the *application world* need to be isolated. Second, those concepts should be transformed into classes, i.e., building blocks that can be used to formulate a problem in the application world. The problem at hand requires handling enrichment of the standard Finite Element Method with *a priori* knowledge about the solution to the boundary value problem. Some of the key concepts involved in the X-FEM are enrichment items (crack, hole, inclusion, material interface, biofilm, solid-fluid interface etc.), enrichment functions (Heaviside function and branch functions etc.) and geometry entities (polylines, points, circles etc.) which will be implemented through the classes **EnrichmentItem**, **EnrichmentFunction** and **GeometryEntity**, respectively. Numerical integration needs special attention for enriched elements (Moës, Dolbow, and Belytschko, 1999). This is handled through the **IntegrationRule** class. To have a flexible way for selection of enriched nodes (classical way or fixed enrichment area, see Section 2.3.1), the class **EnrichmentDetector** was designed. The crack growth law is imple-

mented through class **CrackGrowthDirectionLaw** and its derived class **MaxHoopStress** whereas the rules determining the crack advance length is designed with class **CrackGrowthIncrementLaw** and its derived classes **FixedIncrement** and **ParisLaw**.

Besides these classes, some of the existing FE classes such as **Domain, Element, Node** also needed modifications. The modification of such classes as well as the implementation of new ones are mentioned in detail next.

3.3.1 Additional classes used to describe an enriched finite element problem

Enrichment items

A general X-FEM problem may hold a number of features such as cracks, holes, material interfaces, sliding interfaces, contact interfaces, fluid-solid interfaces etc.. In our implementation, those features will be objects of class **EnrichmentItem**. Such an object holds its geometry, an object of class **GeometryEntity** described in Section 3.3.1, used to check whether the enrichment item interacts with a given element, its enrichment functions, object of class **EnrichmentFunction**, used to model it. The interface of the **EnrichmentItem** class is given in Figure 3.2.

In the present implementation, three derived classes of the base class **EnrichmentItem** were designed as shown in Figure 3.3. A 2D crack with two tips is modeled by an object of class **CrackInterior** (see Figure 3.4) and two objects of class **CrackTip** (see Figure 3.5). A **CrackInterior** object holds its tips thought the data member *myTips*. The method *treatMeshGeoInteractionForMyTips* is used to find elements containing these tips.

Some important data members of class **CrackTip** are presented. The mem-

```
class EnrichmentItem : public FEMComponent { public:
  EnrichmentItem(int,Domain*);
  virtual ~EnrichmentItem();

  void                              getGeometry();
  vector< EnrichmentFunction* >*    giveEnrFuncVector();
  bool                              interactsWith(Element*);
  virtual void                      treatEnrichment();
  EnrichmentDetector*               defineMyEnrDetector();
  EnrichmentDetector*               giveMyEnrDetector();
  GeometryEntity*                   giveMyGeo();
  vector<Element*>*                 giveElementsInteracWithMe();
  void                              setListOfInteractedElements(Element*);
  EnrichmentItem*                   typed () ;
  EnrichmentItem*                   ofType (char*) ;
  virtual void                      resolveLinearDependency(){}
  virtual void                      updateYourGeometry(){;}
  virtual void                      updateEnrichment(){;}
protected:
  GeometryEntity*                   myGeometry;
  vector<EnrichmentFunction*>*      myEnrichFns ;
  EnrichmentDetector*               myEnrDetector;
  vector<Element*>*                 interactedElements ;
} ;
```

Figure 3.2: The interface of class **EnrichmentItem**

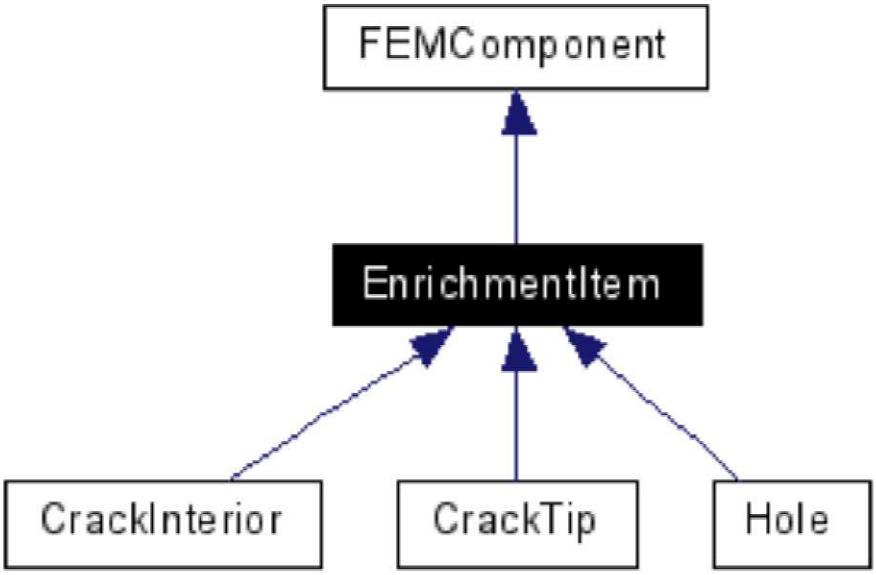


Figure 3.3: The inheritance tree of class **EnrichmentItem**

```
class CrackInterior : public EnrichmentItem
{
public:
   CrackInterior(int,Domain*);
   ~CrackInterior(){;}

   void                    getMyTips();
   std::vector<CrackTip*> giveMyTips();

   void     treatMeshGeoInteractionForMyTips() ;
   void     resolveLinearDependency();

   void     updateEnrichment(){;}
   void     updateMyGeometry();
private:
   std::vector<CrackTip*>  myTips;
} ;
```

Figure 3.4: C++ header file of a 2D crack interior

```
class CrackTip : public EnrichmentItem {
public:
  CrackTip(int,Domain*);
  ~CrackTip();

  void                  computeSIFs(TimeStep* stepN);
  void                  computeK_eq(TimeStep* stepN);
  std::list<Element*>   buildIntegrationDomain();
  std::valarray<double> computeInteractionIntegral(TimeStep* stepN);

  bool  giveState()const {return isActive;}
  void  kill(){isActive = false;}
  void  crackTypeInitialization();
  void  crackTypeUpdate();

  std::vector<Material*>*   giveMatArray();

  Mu::Segment*   giveTipSegment();
  FloatArray*    computeLocalCoordOf(Mu::Point* p);

  double         giveRadiusOfDomainIntegration();
  double         giveEnrichRadius();

  void           resolveLinearDependency(){}

  void                 updateMyGeometry();
  Mu::Circle*          DefineDomainForUpdatedEnrichment();
  std::list<Element*>  defineUpdatedElements();
  void                 updateEnrichment();

  void    setMyAssociatedCrackInterior(CrackInterior* crInt);

private:
    CrackType               tipID;
    FieldType               field;
    std::vector<Material*>* matArray;
    double                  K_i,K_ii,K_eq ;
    Mu::Segment*     tipSegment ;
    bool             isActive;
    CrackInterior*   myAssociatedCrackInterior ;
    Mu::Circle*      makeBall();
} ;
```

Figure 3.5: C++ header file of a 2D crack tip

ber *myAssociatedCrackInterior* stores the associated crack interior of this tip. A **CrackTip** object uses this member to update the geometry, the list of interacted elements of its associated crack interior. The stress intensity factors (of mode I, mode II and the equivalent SIF) are stored in *K_i, K_ii, K_eq* .

For crack growth problems and especially multiple crack growth problems, a crack tip could be no longer a tip (not an object of class **CrackTip** anymore) whenever it reaches a free boundary or another crack. To do this, the member *bool isActive* was implemented. Initially, this member is set to *true*. After the crack grows, its tips's positions are checked and if these tips touch one free boundary, for instance, their *isActive* are set to *false*.

The key method of class **CrackTip** is the one to compute the interaction integral, *computeInteractionIntegral(TimeStep*)*. The pseudo code is given in Figure 3.6.

```
valarray<double> CrackTip::computeInteractionIntegral(TimeStep* stepN)
{
 detection of the elements on which we integrate
 loop over these elements
   loop over Gauss points
     computation of displacement, stress, strain in local coordinates
     computation of the auxiliary fields
     computation of I1 and I2
   end of loop on GPs
 end of loop on elements
}
```

Figure 3.6: Computation of the interaction integral

Enrichment functions

This class implements specific functions holding a priori knowledge about the solution (branch functions for linear elastic fracture mechanics) or particular functions used to model the discontinuities (Heaviside function for displacement discontinuity and ramp function for strain discontinuity).

```
class EnrichmentFunction : public FEMComponent { public:
   EnrichmentFunction(Domain*,int);
   ~EnrichmentFunction();

   virtual double      EvaluateYourSelfAt(Mu::Point*){return NULL;}
   virtual FloatArray* EvaluateYourGradAt(Mu::Point*){return NULL;}

   void                setMyEnrichmentItem(EnrichmentItem*);
   void                findActiveEnrichmentItem(EnrichmentItem*);

   EnrichmentFunction* typed();
   EnrichmentFunction* ofType(char*);
   char*               giveClassName (char* s)
       { return strcpy(s,"EnrichmentFunction");

protected:
   int number;
   std::vector<EnrichmentItem*> *myEnrItems;
   EnrichmentItem               *activeEnrItem;
} ;
```

Figure 3.7: Interface of class **EnrichmentFunction**

An object of class **EnrichmentFunction** should know how to evaluate itself (*EvaluateYourSelfAt(Point*)*) and its gradient (*EvaluateYourGradAt(Point*)*) at a given point in space. It can do this through the geometry and the nature of the enrichment item with which it is associated. A crack tip, for instance, knows which

coordinate system to use to compute the asymptotic enrichment functions through its geometry. If several instances of the same enrichment item are to be modelled in the same problem (for instance, multiple cracks problem), it can happen that some nodes be enriched with two identical enrichment functions, belonging to two different enrichment items. Consequently, we chose to tell each enrichment item know which enrichment function it is associated with, and vice versa. Therefore, an object of class **EnrichmentFunction** must hold a list of objects of **EnrichmentItem** being modeled by this enrichment function. The member data *myEnrItems* (see Figure 3.7) was designed for this purpose. The enrichment functions are being computed for which **EnrichmentItem**? To answer this question, we designed the member data *activeEnrItem*.

The **EnrichmentFunction** class serves as an *interface* to the code, because it specifies all methods that should be implemented by an enrichment function, but actually implements none of them (abstract class). This is a safe method to make sure a new programmer will implement *all* required methods, since this check is made at *compile time*.

An example of a derived class (i.e., specialized forms of enrichment functions), is the class **HomoElastCrackAsymp**, which specializes into the computation of the asymptotic crack tip fields for an homogeneous linear elastic material. For a future programmer to add enrichment functions for, say, a Neo-Hookean material, it is sufficient to implement a **HomoNeoHookeanCrackAsym** class, which will differ from the **HomoElastCrackAsymp** class, by the implementation of its evaluation methods.

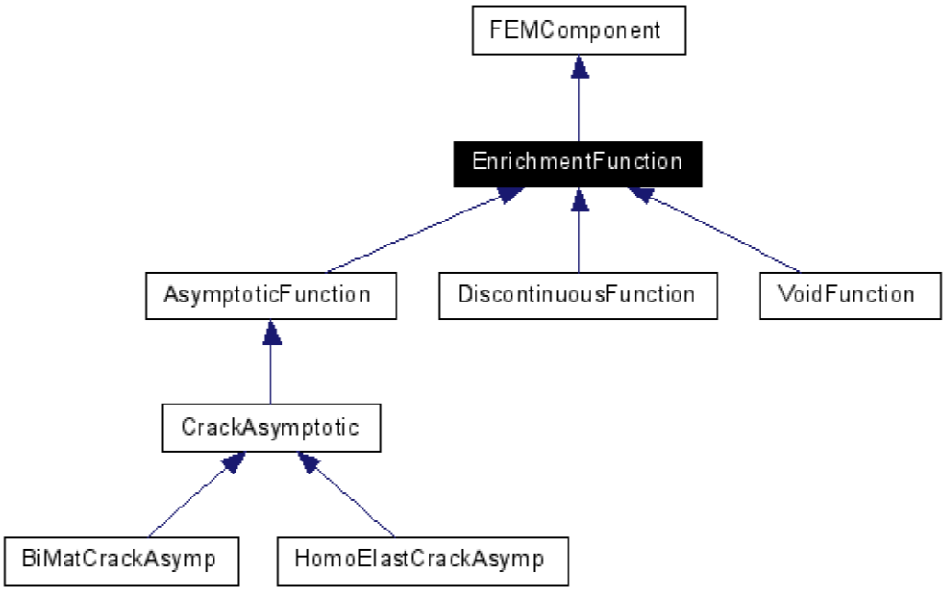


Figure 3.8: The inheritance tree of class **EnrichmentFunction**

Numerical integration

For split elements, to exactly integrate the weak form, we need to partition the element into subelements, to avoid ill-conditioning of the resulting system of equations. The task of class **SplitGaussLegendreQuadrature** lies in finding the coordinates and weights of the quadrature points to be used to integrate the weak-form accurately. First, the intersection points of the element edges with the geometry of the enrichment item are computed. Then, they are fed into a Delaunay mesher together with the element's nodes. Finally, for each sub-element, the Gauss points are obtained, and their coordinates are transformed into the parent coordinate system of the split element. The pseudo code is given in Figure 3.9.

```
void SplitGaussLegendreQuadrature::SetUpPointsOn2dDomain(Element* e)
{
  get the subtriangles of e
  loop on these triangles
  get the Gauss points for each subtriangle
  convert these Gauss points into the parent coordinate system
  of e
}
```

Figure 3.9: Computation of Gauss points for split elements

In order to make seamless the addition of new integration techniques, this class is derived from an interface abstract class: **IntegrationRule**.

Geometry handling

In our implementation, geometries can be handled a number of ways (Standard, Level Set, Vector Level Set). Similarly, the update of the geometry can easily be chosen. The X-FEM is naturally coupled with the level set method (Stolarska,

Chopp, Moës, and Belytschko, 2001; Sukumar, Chopp, Moës, and Belytschko, 2001) and, more recently, the vector level set was proposed, for crack growth problems, as a simpler alternative to the standard level set method. Standard geometry handling for the **EnrichmentItem** is also possible, and used in the numerical examples, along with a Tree structure to manage both geometrical entities and finite elements.

Each **EnrichmentItem** knows its geometry as an object of class **GeometryEntity**, say *myGeo*. Through *myGeo*, the enrichment item knows whether it interacts with an element. In turn, *myGeo*, through its **GeometryDescription** member, knows how to perform this interaction. This piece of code is given as follows:

```
bool GeometryEntity :: interactsWith(Element* e) {
   return this->giveMyGeoDescription()->interactsWith(e,this);
}
```

For example, for a **CrackInterior** represented by standard geometry (line segments), the usual geometry predicates are used as in (Sukumar and Prévost, 2003) to check the intersection between an element and the crack.

Classes specific to the enriched nodes detection

It was shown (Laborde et al., 2004; Béchet et al., 2005) that, for linear elastic fracture mechanics problems, enriching several layers of elements around the crack tip increases the rate of convergence of the X-FEM and also improves the accuracy in the computation of the Stress Intensity Factors. Therefore, a flexible X-FEM code should allow the user to decide the criterion based on which enriched nodes are selected. Inclusion in a ball of radius r centered at the crack tip is a possible

criterion. Alternatively, the programmer/user may decide to enrich a specified number of layers of elements around the tip element.

Similarly, only nodes belonging to an element split by a discontinuity should be enriched with the Heaviside function, while, for biofilm problems (Bordas, 2003), elements within the biofilm, and below a given distance from the biofilm/water interface often need to be asymptotically enriched in order to track the boundary layer.

The abstract class **EnrichmentDetector** and its derived classes serve the purpose of making the selection of enriched nodes flexible. Abstract class **EnrichmentDetector** is designed with only one pure virtual method as follows :

```
class EnrichmentDetector{
public:
  virtual void setEnrichedNodes(EnrichmentItem *enrItem)=0;
};
```

It is obvious that we can not have objects of class **EnrichmentDetector** since a general **EnrichmentDetector** does not know how to select enriched nodes. That's why we designed this class as an abstract class. Therefore, its derived classes must implement the method *setEnrichedNodes(EnrichmentItem*)*. For example, the code to enrich nodes belong to split elements is given in Figure 3.10.

For linear elements, just enrich all of its nodes. However, for high order elements, say quadratic triangular elements, the choice of which nodes should be enriched depends on the partition of unity shape functions to be used. If linear PUM shape functions are used, then just enrich the three corner nodes, not enrich the three midside nodes. This is what is implemented in method *setEnrichmentForMyNodes(EnrichmentItem *enrItem)* of class **Tri6** as follows

```
void SplitElementDetect :: setEnrichedNodes(EnrichmentItem *enrItem)
{
if(! (typeid(*enrItem) == typeid(CrackInterior*)) )
{
   assert(false);
}

vector<Element*>* elemList = enrItem->giveElementsInteractWithMe();

for(size_t i = 0 ; i < elemList->size() ; i++){
   (*elemList)[i]->setEnrichmentForMyNodes(enrItem) ;
}
}
```

Figure 3.10: Method *setEnrichedNodes* of derived class **SplitElementDetect**

```
void Tri6_U::setEnrichmentForMyNodes(EnrichmentItem *enrItem){
 size_t intOrder =
  this->giveXFEInterpolation()->giveInterpolationOrder();
 if(intOrder == 1)          // just enrich three corner nodes
 {
    for(size_t i = 0 ; i < 3 ; i++)
    {
       this->giveNode(i+1)->isEnrichedWith(enrItem)  ;
    }
 }
 else                       // enrich all nodes
 {
    for(size_t i = 0 ; i < numberOfNodes ; i++)
    {
       this->giveNode(i+1)->isEnrichedWith(enrItem)  ;
    }
 }
}
```

```
class Domain {
 private :

  List*                  enrichmentFunctionList ;
  List*                  enrichmentItemList ;
  List*                  geoEntityList ;
  size_t                 numberOfEnrichmentFunctions ;
  size_t                 numberOfEnrichmentItems     ;
  size_t                 numberOfGeoEntities ;
  bool                   isFEM ;
  bool                   isXFEM ;
  CrackGrowthDirectionLaw* directionLaw ;
  CrackGrowthIncrementLaw* incrementLaw ;

 public :

  EnrichmentItem*        giveEnrichmentItem(int n);
  EnrichmentFunction*    giveEnrichmentFunction(int n)
  GeometryEntity*        giveGeoEntity(int n) ;

  void                   solveFractureMechanicsProblem ();
  void                   solveFractureMechanicsProblemAt(TimeStep*);

  void                   treatMeshGeoInteractionPhase1();
  void                   treatMeshGeoInteractionPhase2();
  void                   treatEnrichment() ;
  void                   resolveLinearDependencyForEnrichment();

  std::map<Node*,std::vector<Element*> >  buildNodalSupports();
  std::map<Node*,std::vector<Element*> >  giveNodalSupports();
} ;
```

Figure 3.11: Added data and methods for class **Domain** to account for discontinuities

3.3.2 Modification of standard finite element classes

Class Domain

The domain (Zimmermann, Pelerin, and Bomme, 1992) can be considered as the main object that contains all the problem's components: list of nodes, elements, materials, loads... It serves as a link between the components needed to describe the physical and numerical problem. To account for the discontinuities, the following data members and methods were added as shown in Figure 3.11.

The main method of this class is the solution procedure *solveFractureMechanicsProblemAt(TimeStep* stepN)* as shown in Figure 3.12. Given the aforementioned classes of objects, solving an enriched finite element problem now reduces to handling the mesh geometry interaction to find out elements that interact with the **EnrichmentItem**s and enrich the corresponding nodes with the approriate **EnrichmentFunction**s. If there are conflicts, those are resolved by the method *resolveLinearDependencyForEnrichment()*. Finally, the **Domain** object asks its **NLSolver** object to *Solve()* the problem using the appropriate scheme (Zimmermann, Pelerin, and Bomme, 1992).

Class Element

The data members that were added for the **Element** class include the following:

FEInterpolation* standardFEInterpolation used as interpolation functions for classical finite approximation.

FEInterpolation* enrichmentFEInterpolation This is finite interpolation used to approximate the enriched fields. It was shown that using higher order

```
void  Domain :: solveFractureMechanicsProblemAt(TimeStep* stepN)
{
    if (unknownArray) {
        delete unknownArray;
    }

    this -> treatMeshGeoInteractionPhase1();
    this -> treatMeshGeoInteractionPhase2();
    this -> treatEnrichment();
    this -> resolveConflictsInEnrichment();
    this -> resolveLinearDependencyForEnrichment();

    unknownArray = this -> giveNLSolver() -> Solve();

    this -> terminate(stepN) ;
}
```

Figure 3.12: Method *solveFractureMechanicsProblemAt(TimeStep* stepN)*

shape functions for this approximation decrease the error, but make the convergence rate erratic, because of the loss of the reproducing condition in partially enriched elements (Laborde, Pommier, Renard, and Salaun, 2004; Chessa, Wang, and Belytschko, 2003). The current code uses the linear interpolation.

std::list<EnrichmentItem*>* enrichmentItemListOfElem Every **EnrichmentItem** objects interacting with a given element are stored in this list. **Element** e will use this list to do the partitioning for numerical integration.

bool isUpdated a marker to indicate whether a given element should recompute its stiffness matrix when the **EnrichmentItem** evolves in time.

New methods are added :

```
bool              isEnriched();
void              isEnrichedWith(EnrichmentItem *enrItem);
void              treatGeoMeshInteraction();
virtual void      setEnrichmentForMyNodes(EnrichmentItem*){}

virtual FEInterpolation*    giveFEInterpolation(){return NULL;}
virtual FEInterpolation*    giveXFEInterpolation(){return NULL;}

void    setUpIntegrationRule();
virtual vector<DelaunayTriangle*>* PartitionMySelf(){return NULL;}
virtual GaussPoint** setGaussQuadForJ_Integral(){return NULL;}

void    setStateOfElement(){isUpdated = true ;}
bool    isUpdatedElement(){return isUpdated ;}
```

The method *isEnriched()* consists in looping over the receiver's nodes (the current instance of the **Element** object), if at least one node is enriched, then the element is enriched. The **Element** needs to know if it is enriched or not to correctly compute the B matrix (equation (3.1)) and use the appropriate integration rule.

When an element is enriched by an **EnrichmentItem** then the method *isEnrichedWith(EnrichmentItem *enrItem)* inserts *enrItem* into its enrichmentItemListOfElem.

The definition of the elemental stiffness matrix K_e writes

$$K_e = \int_e B^T D B dV_e \tag{3.1}$$

To compute the stiffness matrix for both enriched and non-enriched elements, the method *computeBMatrixAt(Gausspoint *gp)* was modified as shown in Figure 3.13. The first advantage of this approach is that the code to compute the stiffness matrix of any element (enriched or not) is unchanged as shown in Figure 3.14. Secondly, a node can be enriched with any number of enrichment functions. For

instance, an element may be split by a crack and also contain the tip of another crack. In this case, the nodes of this element are enriched by both the step function (of the first crack) and the crack asymptotic functions (of the second one).

```
FloatMatrix*  Element :: ComputeBmatrixAt(GaussPoint *aGausspoint){
 Compute the standard part of B matrix, called Bu
 If this element is not enriched, return Bu and stop.
 Otherwise, loop on nodes, if node is enriched then
 Loop on enrichment items of this node, say enrItem
    Each enrItem knows which enrichment functions should be used.
    Loop on these enrichment functions and calculate their
    contribution to B matrix.
 The final result is B = [Bu Ba]
}
```

Figure 3.13: Method *ComputeBmatrixAt(GaussPoint*)*

```
FloatMatrix*  Element :: computeTangentStiffnessMatrix(){
double K = 0.0 ;
for_each Gauss point, gp, in the GaussPointArray
    compute the B matrix using this->ComputeBmatrixAt(gp)
    compute the transpose of B, BT
    compute the constitutive matrix D
    compute dV = detJ*t (t is the thickness of the element)
    K += BT*D*B*dV ;
end for
}
```

Figure 3.14: The unchanged method *computeTangentStiffnessMatrix()*

To do the element partitioning for finding Gauss points for split and tip elements, method *PartitionMySelf()* is implemented. It is a pure virtual method, therefore, its derived classes such as **TriU, QuadU**, and **Tri6** have their own implementation.

Since the Gauss points used for the computation of the interaction integral are different from those used in the stiffness matrices computation, the method *setGaussQuadForJIntegral()* was designed. It is a pure virtual method since each element type requires different Gauss points.

It is obvious that when **EnrichmentItem** objects change, for example, when cracks grow, only some nodes and elements around the crack tip change status. The rest of the elements are unchanged so it is not efficient to recompute their stiffness matrices. To solve this problem, the data member *isUpdated* is implemented. Initially, this member is set to false, after the crack growth, if a given element is detected to be changed then its *isUpdated* is set to true. Thanks to this its stiffness matrix is not recomputed as reflected in the method *giveStiffnessMatrix()* as given in Figure 3.15.

```
FloatMatrix*  Element :: giveStiffnessMatrix ()
{
  if (! stiffnessMatrix)
     this -> computeTangentStiffnessMatrix() ;
  else if(isUpdated)
     this -> computeTangentStiffnessMatrix() ;

  return stiffnessMatrix ;
}
```

Figure 3.15: Method *giveStiffnessMatrix()* of class **Element**

Class Node

To handle nodal enrichment, the following data members are added to this class

int isEnriched a marker to differentiate non-enriched and enriched nodes.

list<EnrichmentItem*>* enrichmentItemListOfNode This is the list of all **EnrichmentItem** objects acting on the node.

and here are added methods

```
void                isEnrichedWith(EnrichmentItem* enrItem);
void                resolveConflictsInEnrichment();
void                resolveLinearDependency(EnrichmentItem*);

int                       getIsEnriched(){return isEnriched;}
list<EnrichmentItem*>* giveEnrItemListOfNode();
```

A node should not be enriched with both the Heaviside function and branch functions. Therefore, whenever the data member *enrichmentItemListOfNode* contains objects of both classes **CrackInteiror** and **CrackTip**, one should remove object of **CrackInteiror** so that this node is just enriched with branch functions. This is performed by method *resolveConflictsInEnrichment()* as given in Figure 3.16.

```
void Node :: resolveConflictsInEnrichment() {
 list<EnrichmentItem*> ::iterator iter1,iter2;

 iter1 = find_if(enrichmentItemListOfNode->begin(),
 enrichmentItemListOfNode->end(),IsType<CrackTip,EnrichmentItem>());
 iter2 = find_if(enrichmentItemListOfNode->begin(),
 enrichmentItemListOfNode->end(),IsType<CrackInterior,EnrichmentItem>());

 if (iter1 != enrichmentItemListOfNode->end() &&
     iter2 != enrichmentItemListOfNode->end())
    enrichmentItemListOfNode->remove(*iter2);
}
```

Figure 3.16: Method *resolveConflictsInEnrichment()* of class **Node**

```
template<class U,class V>
class IsType
{
public:
  bool operator ()(V* t)
  {
     return (typeid(*t)==typeid(U))? true:false ;
  }
};
```

Figure 3.17: A template functor

One can see that this method is simple and efficient thanks to the STL algorithm *find_if* and the functor *IsType* which implemented as shown in Figure 3.17.

For any node enriched by the Heaviside function $H(\mathbf{x})$, its support is fully cut into two disjoint pieces by the crack. If for a certain node n_I, one of the two pieces is very small compared to the other, then the function H is almost a constant over the support, leading to an ill-conditioned stiffness matrix (Moës et al., 1999). In this case, node n_I should no longer be enriched with function H. This is exactly what is performed by method *resolveLinearDependency(EnrichmentItem*)*.

With enriched finite elements, the number and nature of the degrees of freedoms (dof information) associated with a node may vary from node to node and in addition, evolve with time. Therefore, the way to compute the number of DOFs and the location of Dofs in the global matrix must be modified. Below are modified classes and added classes implemented for this purpose :

```
size_t          computeNumberOfDofs () ;
size_t          giveNumberOfTrueDofs () ;
size_t          giveNumberOfDofs () ;
```

```
IntArray*       giveStandardLocationArray () ;
IntArray*       giveEnrichedLocationArray () ;
```

For plane elasticity problems, the number of DOFs, n_{DOF}, of any node is determined by

$$n_{DOF} = 2 + 2n_{enr} \tag{3.2}$$

where n_{enr} is the number of enrichment functions used to enrich this node. n_{enr} is the sum of enrichment functions of every enrichment items acting on this node. The piece of code to compute n_{DOF} is given in Figure 3.18

3.4 Extension to new problems

This section explains how to extend the current code to include new problems. It shows that this task can be carried out very easily. Here, assume that we want to solve interfacial crack problems. First, the asymptotic functions associated with this problem need to be added (Sukumar, Huang, Prévost, and Suo, 2003). To do so, it suffices to build a new class called **BiMaterialElastCrackAsymp** in which the near tip asymptotic functions for interfacial cracks are implemented. It is emphasized that this new class is completely similar to class **HomoElastCrackAsymp**.

Finally, to compute the SIFs, we need to implement the auxiliary fields used in the domain integral computations. Here, there are two possibilities : (1) use inheritance, i.e., one would implement an abstract class, say AuxiliaryFields, with pure virtual methods to compute the components of any auxiliary field and two derived classes, one for homogeneous crack and one for interfacial crack. This ap-

```
size_t  Node :: computeNumberOfDofs ()
{
   numberOfDofs = this->readInteger("nDofs") ;

   if ( isEnriched == 0 )
      return numberOfDofs ;

   size_t enrichedDofs = 0 ;
   list<EnrichmentItem*>* enrItemList=this->giveEnrItemListOfNode();

   list<EnrichmentItem*>::iterator iter ;
   vector<EnrichmentFunction*>* enrFnVector ;
   for(iter=enrItemList->begin();iter != enrItemList->end();iter++)
   {
     enrFnVector = (*iter)->giveEnrFuncVector();
     enrichedDofs += 2 * enrFnVector->size();
   }

   numberOfDofs += enrichedDofs ;

   return numberOfDofs ;
}
```

Figure 3.18: Method *computeNumberOfDofs ()* of class **Node**

proach is simple but code is duplicated; (2) use the template technique. Following this way, one only needs to implement a so-called template class. Below is what should be implemented for the inheritance approach:

An abstract class for any auxiliary field:

```
class AuxiliaryField {
 virtual void ComputeComponentsOfAuxField(CrackTip* tip, Mu::Point* p,
   FloatMatrix& AuxGradDisp,FloatArray& AuxEps,FloatArray&AuxStress){}
};
```

A derived class, a specialization of class AuxiliaryField, for example, one for homogeneous crack :

```
class HomoCrackAuxiliaryField {
 void ComputeComponentsOfAuxField(CrackTip* tip, Mu::Point* p,
   FloatMatrix& AuxGradDisp,FloatArray& AuxEps,FloatArray&AuxStress);
};
```

Now, if we would like to solve interfacial crack, the following class should simply be added:

```
class InterfacialCrackAuxiliaryField {
 void ComputeComponentsOfAuxField(CrackTip* tip, Mu::Point* p,
   FloatMatrix& AuxGradDisp,FloatArray& AuxEps,FloatArray&AuxStress);
};
```

In the current implementation, we chose the template approach as given in Figure 3.19 :

Then, when we need the auxiliary field of homogeneous crack (plane strain condition), the following declaration suffices:

```
AuxiliaryField<Material,NullMaterial,PlaneStrain> *auxFieldHomo
   = new AuxiliaryField<Material,NullMaterial,PlaneStrain>();
```

```
template<class M1, class M2, const FieldType field=PlaneStrain>
class AuxiliaryField
{
public:

   AuxiliaryField(){};
   virtual ~AuxiliaryField(){};

   void ComputeComponentsOfAuxField(CrackTip*,Point*,ModeType,
       FloatMatrix&,FloatArray&,FloatArray&);
   void ComputeComponentsOfOneMat(CrackTip* tip,Point*,ModeType,
       FloatMatrix&,FloatArray&,FloatArray&);
   void ComputeComponentsOfBiMat(CrackTip* tip,Point*,ModeType,
       FloatMatrix&,FloatArray&,FloatArray&);

protected:

   M1 *material1;
   M2 *material2;

   size_t giveNumberOfMaterials() const
   {
      return (size_t)(typeid(M2) != typeid(NullMaterial))+1 ;
   }

};
```

Figure 3.19: Interface of class **AuxiliaryField**

If the auxiliary field of bimaterial crack are needed, then one declares as follows

```
AuxiliaryField<Material,Material,PlaneStrain> *auxFieldBiMat
   = new AuxiliaryField<Material,Material,PlaneStrain>();
```

3.5 Conclusions

An object-oriented approach to enriched finite element methods was presented. Issues pertaining to the modification of classic finite element classes as well as the implementation of new classes were addressed. Based on this approach, a C++ package for enriched finite elements has been implemented and used to solve numerous fracture problems illustrated in Chapter 4. This package serves as a starting package for further development of the X-FEM.

Chapter 4

Numerical examples

This chapter is composed of two parts. The first part presents numerical results of the computation of the stress intensity factors (SIFs) for numerous two dimensional linear elastic fracture mechanics benchmark problems. The convergence of SIFs with mesh refinement as well as the influence of some of parameters involved in this computation are also studied. The second part introduces the numerical examples of crack growth simulation. The unstructured triangle meshes used in this thesis are obtained with the program Gmsh (Remacle and Geuzaine, 1998) whereas the cross triangle meshes and rectangular meshes are created by Matlab utilities from Northwestern university. Although the stress analysis is performed with the C++ code, the post processing, for instance, displacement and stress plot is done with Matlab.

4.1 Static crack problems

In all of the following examples, plane strain conditions are assumed through out. The calculation of the stress intensity factors is performed with the domain form

of the interaction integral as detailed in Section 2.1.4.

4.1.1 Infinite plate

Consider an infinite plate containing a straight crack of length $2a$ and loaded by a remote uniform stress field σ. Along ABCD the closed form solution in terms of polar coordinates in a reference frame (r, θ) centered at the crack tip is:

$$\begin{aligned} u_x &= \frac{2(1+\upsilon)}{\sqrt{2\pi}} \frac{K_I}{E} \sqrt{r} \cos\frac{\theta}{2}\left(2 - 2\upsilon - \cos^2\frac{\theta}{2}\right) \\ u_y &= \frac{2(1+\upsilon)}{\sqrt{2\pi}} \frac{K_I}{E} \sqrt{r} \sin\frac{\theta}{2}\left(2 - 2\upsilon - \cos^2\frac{\theta}{2}\right) \end{aligned} \tag{4.1}$$

where $K_I = \sigma\sqrt{\pi a}$ is the stress intensity factor, υ is Poisson's ratio and E is Young modulus. ABCD is a square of $10 \times 10\ mm^2$, $a = 100\ mm$; $E = 10^7 N/mm^2, \upsilon = 0.3$, $\sigma = 10^4 N/mm^2$.

The geometry and finite element mesh of domain ABCD are shown in Figure 4.1. This structured triangle mesh consists of 324 elements and as one can see, the crack geometry is not aligned with the mesh. Displacement of nodes on bottom, right and top edges are prescribed by equation (4.1).

The Gauss quadrature for X-FEM with three-noded triangular elements is as follows, see Figure 4.2

- Non-enriched elements : one Gauss point (GP)
- Partially tip enriched elements : thirteen Gauss points
- Partially step enriched elements : three Gauss points

- Split and tip elements : thirteen GPs for each sub-triangle

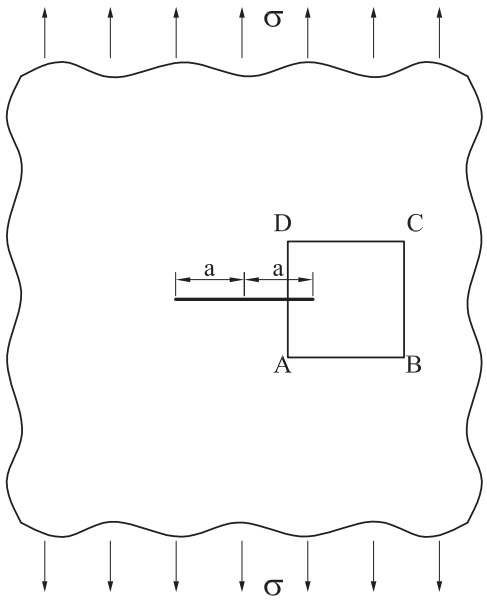
For the computation of the interaction integral, seven Gauss points are used for elements not cut by the crack. Elements cut by the crack are partitioned into sub-triangles for which seven Gauss points are used (see Figure 4.4).

The deformed configuration of the plate is plotted and compared with the exact solution as shown in Figure 4.3.

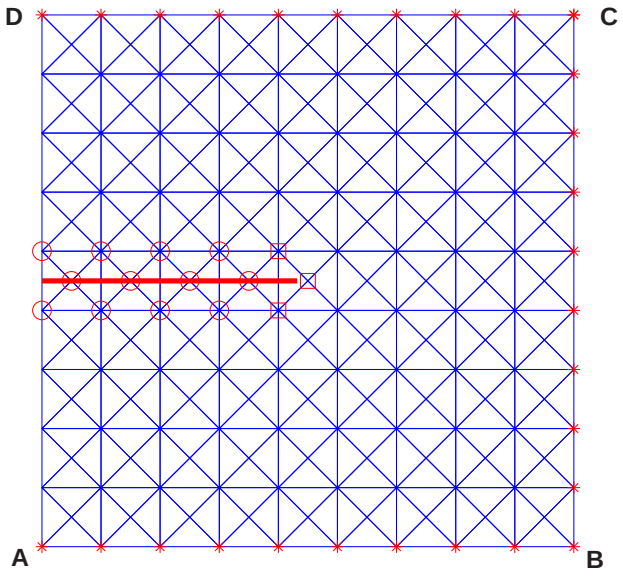
The stress intensity factors are computed using the domain form of the interaction integral (see Appendix C for details). The domain size is characterized by the local mesh spacing at the crack tip h_{local} : $r_d = r_k h_{local}$ with r_k is a scalar multiple and h_{local} is the square root of the area of the tip element. The normalized stress intensity factors for various domain sizes are given in Table 4.1. From this result, we observe that if the domain size is small, i.e., $r_k \leq 2$, then the corresponding SIFs are incorrect. This is due to the singularity at the crack tip. When the domain size is big enough, $r_k > 2$, the effect of the singularity is small, the J integral are independent of the domain contour.

In the following, the X-FEM with fixed enrichment area scheme, see Section 2.3.1, is also studied with the purpose of finding the reasonable J-integral radius r_d for a given enrichment radius r_{enr}. The numerical model is a structured mesh of 324 elements. The enrichment radius is $r_{enr} = 3h_{local} \simeq 1.8$. Any node within the circle centered at the crack tip and radius equal to r_{enr} is enriched by the branch functions (see Figure 4.5). The mode I SIF was computed with various r_d. From Figure 4.6, one can observe that, to get good results, the domain radius r_d should be chosen equal or greater than the enrichment radius r_{enr}.

The von Mises stress contour is plotted in Figure 4.7. To get this relatively



(a)



(b)

Figure 4.1: Infinite cracked plate under remote tension: (a) Geometry and loads ; (b) discretization around the crack tip, nodes labeled with a circle are enriched with the step function and nodes indicated with a square are enriched with the branch functions.

smooth stress contour, the very refined mesh of 28000 elements was used.

r_d/h_{local}	1.5	2.0	2.5	3.0	3.5	4.0	4.5
K_I/K^{exac}	0.9601	1.0085	1.0040	0.9943	0.9933	0.9934	0.9938

Table 4.1: Normalized K_I values for various domain sizes.

4.1.2 Edge crack in tension

A plate of dimension 1×2 is loaded by a tension $\sigma = 1.0$ psi over the top edge and bottom edge as shown in Figure 4.8. The displacement along the y-axis is fixed at the bottom right corner, and the plate is clamped at the bottom left corner. The material parameters are 10^3 psi for Young's modulus and 0.3 for Poisson's ratio. The reference mode I SIF as given in Tada, Paris, and Irwin (1973) is:

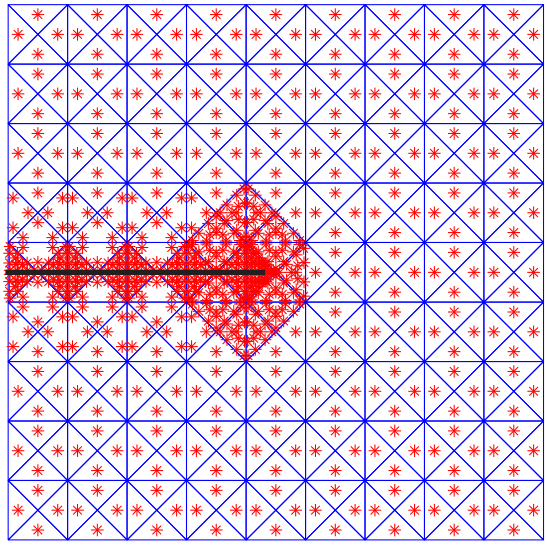
$$K_I = F\big(\frac{a}{b}\big)\sigma\sqrt{\pi a} \tag{4.2}$$

where a is the crack length, b is the plate width, and $F\big(\frac{a}{b}\big)$ is an empirical function. For $a/b \leq 0.6$, the function F is:

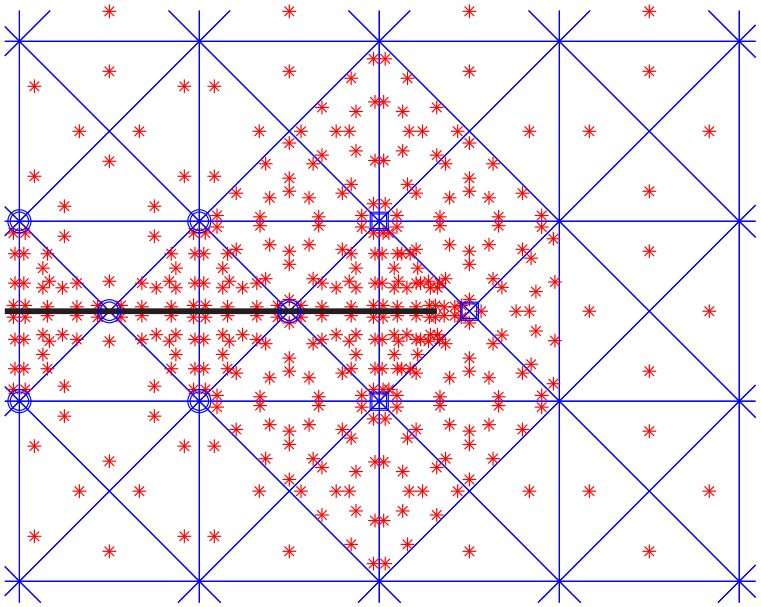
$$F\left(\frac{a}{b}\right) = 1.12 - 0.231\left(\frac{a}{b}\right) + 10.55\left(\frac{a}{b}\right)^2 - 21.72\left(\frac{a}{b}\right)^3 + 30.39\left(\frac{a}{b}\right)^4 \tag{4.3}$$

with $a = 0.45$, the reference stress intensity factor is 2.8766.

The stress intensity factor is computed for various discretizations and domain sizes and tabulated in Table 4.2. One can conclude, based on this table, that the SIFs are independent of the choice of the domain size $r_d > 2h_{local}$, where h_{local} is the crack tip element size. Although the obtained SIFs are relatively good with



(a)



(b)

Figure 4.2: Gauss points used in the numerical computation of the stiffness matrices

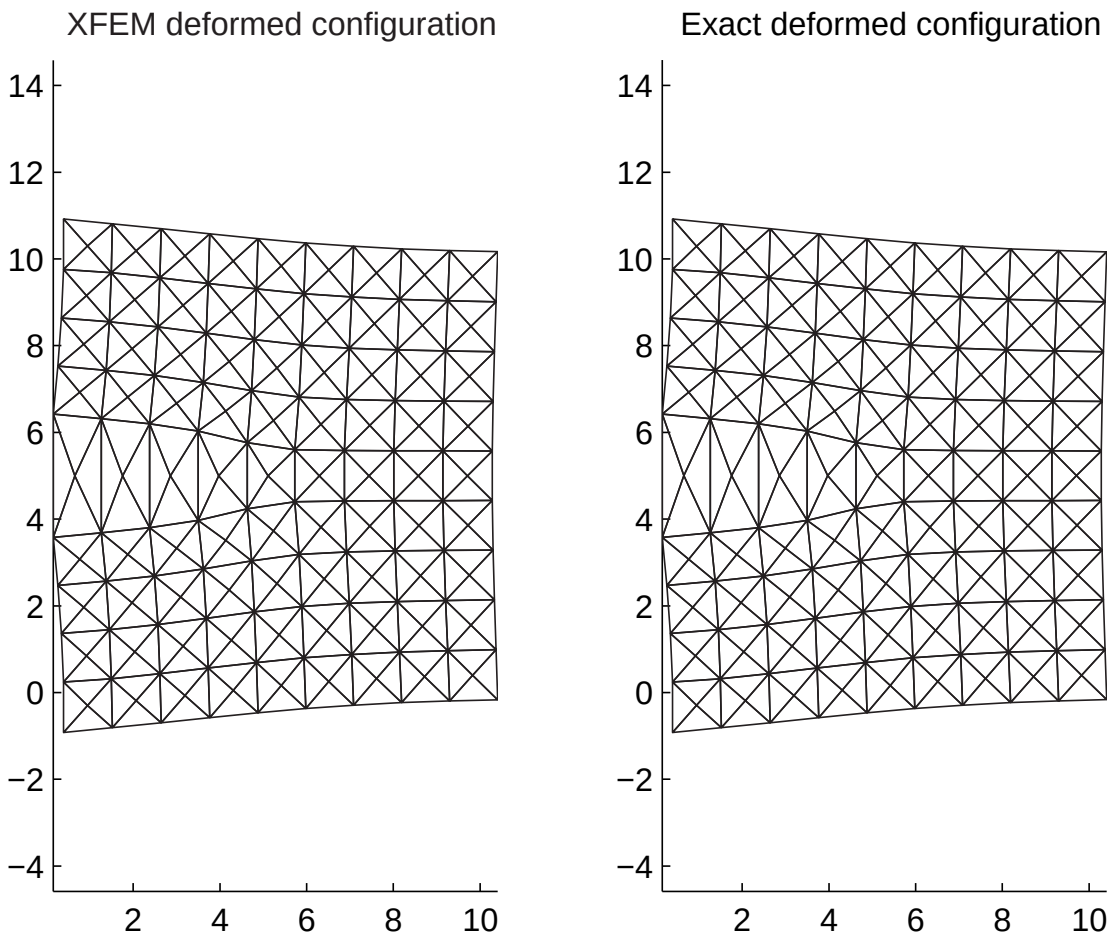


Figure 4.3: Comparison of the deformed mesh.

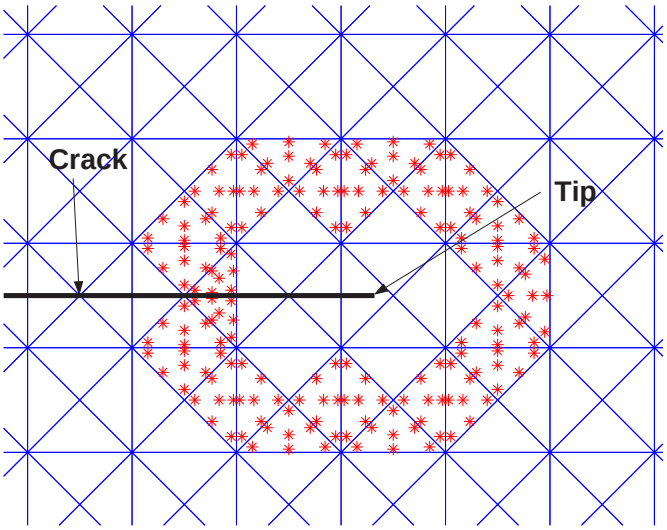


Figure 4.4: Elements used to compute the interaction integral ($r_d = 3h_{local}$). The stars are Gauss points to numerically compute the integral.

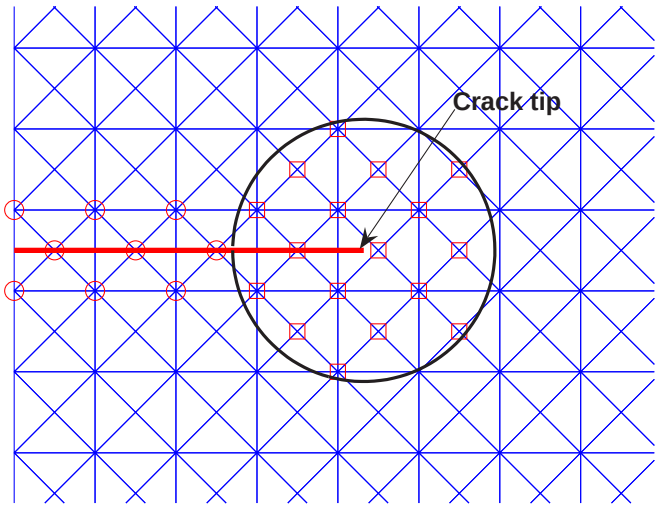


Figure 4.5: X-FEM with fixed enrichment area, X-FEM-f.a. Enrichment radius is $r_{enr} = 3h_{local}$.

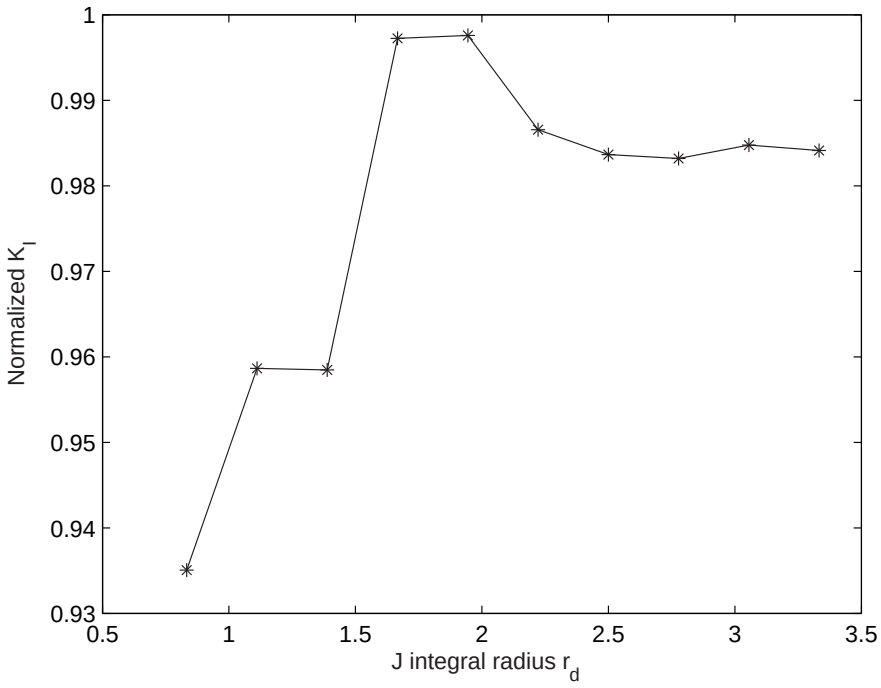


Figure 4.6: Normalized K_I versus J-integral radius r_d.

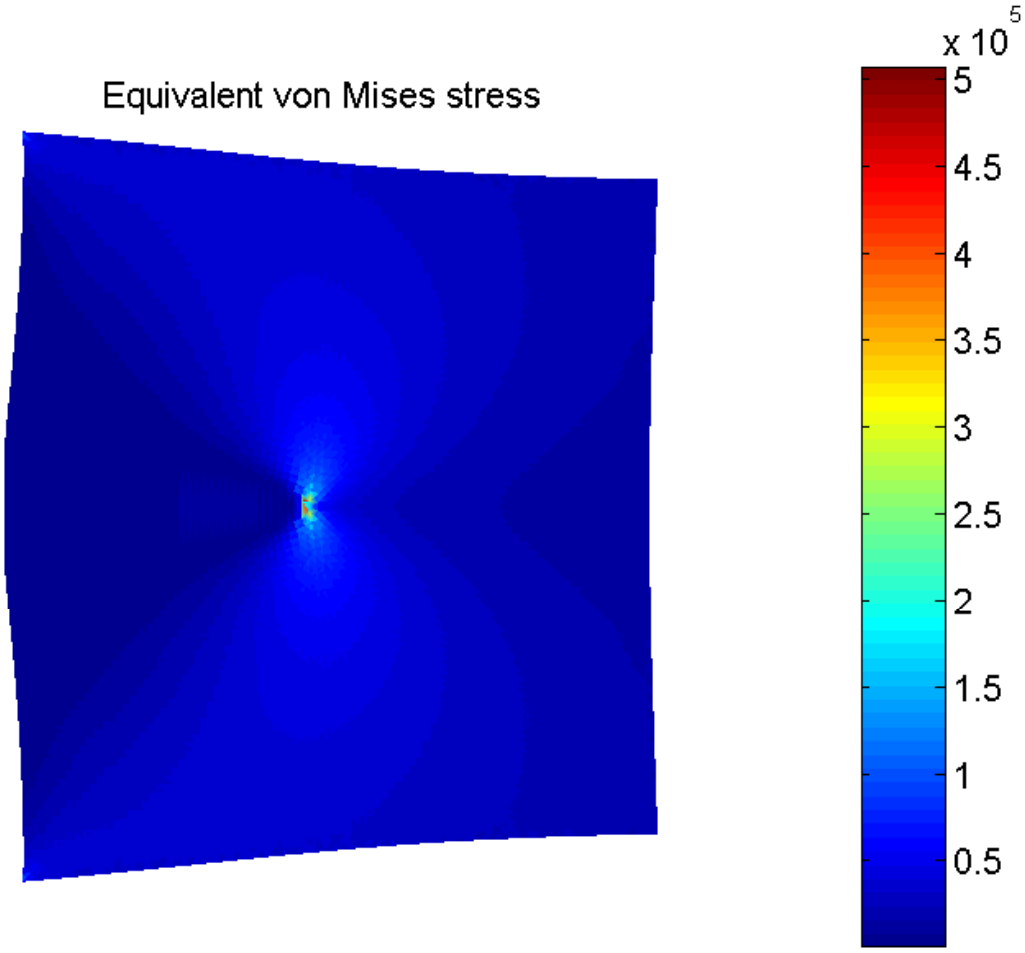
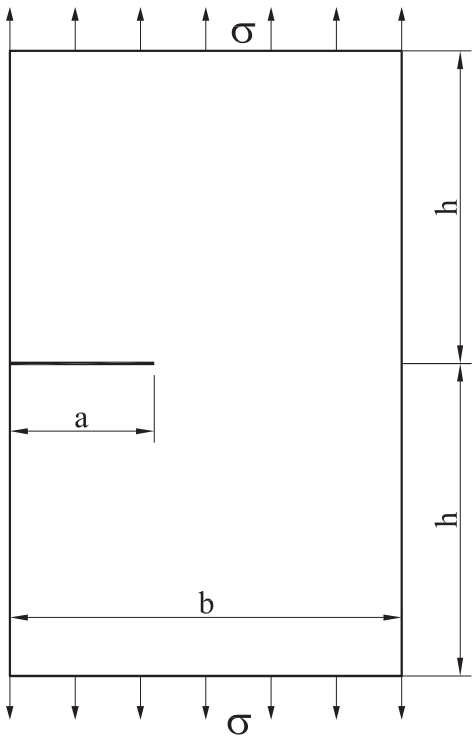
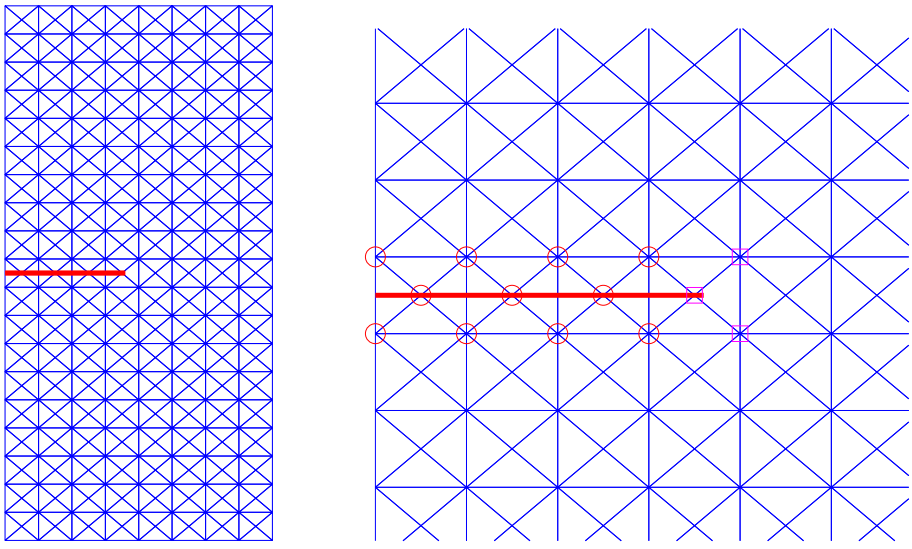


Figure 4.7: The von Mises stress contours.



(a)



(b) (c)

Figure 4.8: Edge-cracked plate in tension: (a) Geometry and loads; (b) finite element mesh; and (c) zoom on the crack area; note that only nodes of tip-element are enriched with the branch functions.

such coarse meshes, it can be seen that the accuracy is not good irrespective of the fineness of the mesh. Blending elements (see Section 2.3.3) contribute to decreasing the rate of convergence.

Table 4.2: Normalized K_I values for various discretizations and domain sizes.

Mesh	Num. elements	r_d/h_{local}						
		1.5	2.0	2.5	3.0	3.5	4.0	4.5
Mesh1	608	0.9659	0.9427	0.9422	0.9395	0.9399	0.9428	0.9412
Mesh2	2808	0.9642	0.9668	0.9748	0.9688	0.9671	0.9672	0.9689
Mesh3	5664	1.0212	1.0286	0.9854	0.9844	0.9784	0.9798	0.9820

The X-FEM using fixed enrichment area is examined with the purpose of finding the optimal J-integral radius r_d for a given enrichment radius r_{enr}. The finite element mesh used is a structured triangular mesh consisting of 680 elements and the enrichment radius $r_{enr} = 0.2$. All nodes belonging to the circle centered at the crack tip with radius r_{enr} are enriched with the branch functions, see Figure 4.9. The normalized stress intensity factors are computed for various J integral radii r_d. Table 4.3 shows again that the J integral radius r_d should be chosen equal or greater than the enrichment radius r_{enr} to get good SIFs.

Table 4.3: Normalized K_I values for domain sizes using normal enrichment and fixed enrichment area .

r_d/h_{local}	1.5	2.0	2.5	3.0	3.5	4.0	4.5
r_d	0.0860	0.1434	0.1721	0.2008	0.2294	0.2868	0.3155
X-FEM	0.9649	0.9427	0.9369	0.9395	0.9366	0.9411	0.9401
X-FEM-f.a.	0.8788	0.9525	0.9745	0.9746	0.9739	0.9561	0.9560

The stress intensity factors for various crack lengths are computed using normal enrichment and fixed enrichment area. It is emphasized that only one mesh

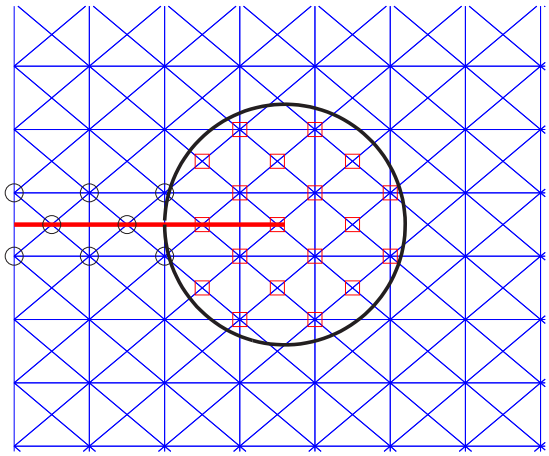


Figure 4.9: Fixed enrichment area scheme. Circled nodes are enriched with $H(x)$ whereas squared nodes are enriched by the branch functions.

(structured triangular mesh of 2802 elements) is used for all crack lengths. The enrichment domain is the circular patch of radius $r_{enr} = 3 \times h_{local} = 0.1$. The domain sizes for the J integral computation are $r_d/h_{local} = 5.0$, and 3.5 for normal enrichment and fixed area enrichment, respectively. Figure 4.10 shows good agreement between the numerical solution and the reference solution, especially results obtained with the X-FEM using fixed enrichment area.

To check the convergence of the X-FEM, the stress intensity factors are computed for various structured meshes. The standard enrichment scheme was used (only the nodes of the tip-element are enriched with the branch functions). The domain size is $r_d = 3h_{local}$ for all meshes. The result is given in Table 4.4 and also plotted in Figure 4.11. From this figure, one could observe that, although the result is converged, the convergence rate is very slow when the mesh is fine. It is due to the blending elements (partially enriched elements), see Section 2.3.3. Another possible reason is that the domain size used to compute the SIFs is proportional to

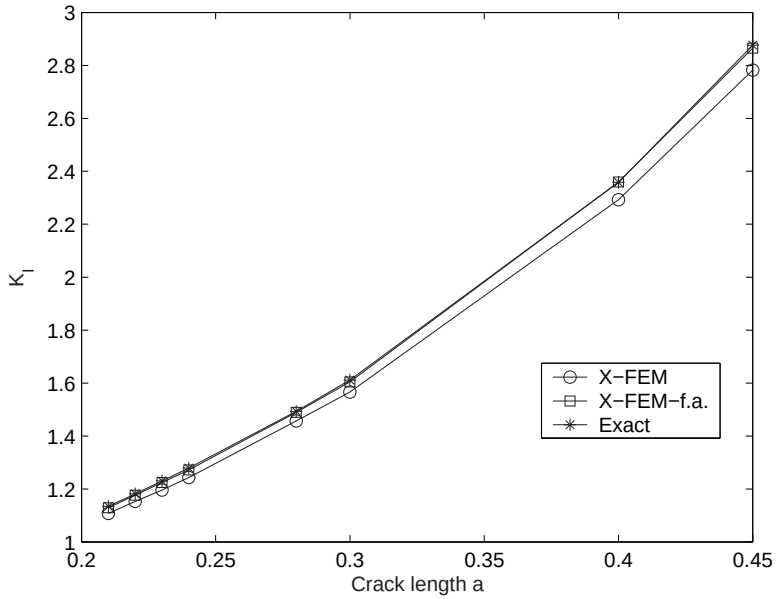


Figure 4.10: SIFs of various crack lengths.

tip-element's size h_{local} which makes it small for very refined meshes. To improve the rate of convergence, we could use one of the following methods : high order elements (not yet implemented) ; the X-FEM with fixed enrichment area (refer to Laborde et al. (2004), Béchet et al. (2005)); the domain size should be fixed for all meshes. In this way, the enriched area is independent of the mesh size during mesh refinement, which improves the convergence rate, as is also shown in S. Bordas and Chopp (2005).

4.1.3 Edge crack under shear stress

A plate is clamped on the bottom edge and loaded by a shear traction $\tau = 1.0$ psi over the top edge as shown in Figure 4.12. The material parameters are $3 \times 10^7 psi$ for Young's modulus and 0.25 for Poisson's ratio. The reference mixed mode stress

Table 4.4: Convergence of SIF of an edge crack plate in tension

Num. nodes	K_I	K_{exact}	$\frac{K_I - K_{exact}}{K_{exact}}(\%)$
332	2.7026	2.8766	6.0488
1462	2.7869	2.8766	3.1183
2916	2.8317	2.8766	1.5609
6186	2.8545	2.8766	0.7672
8080	2.8568	2.8766	0.6901
11546	2.8642	2.8766	0.4328

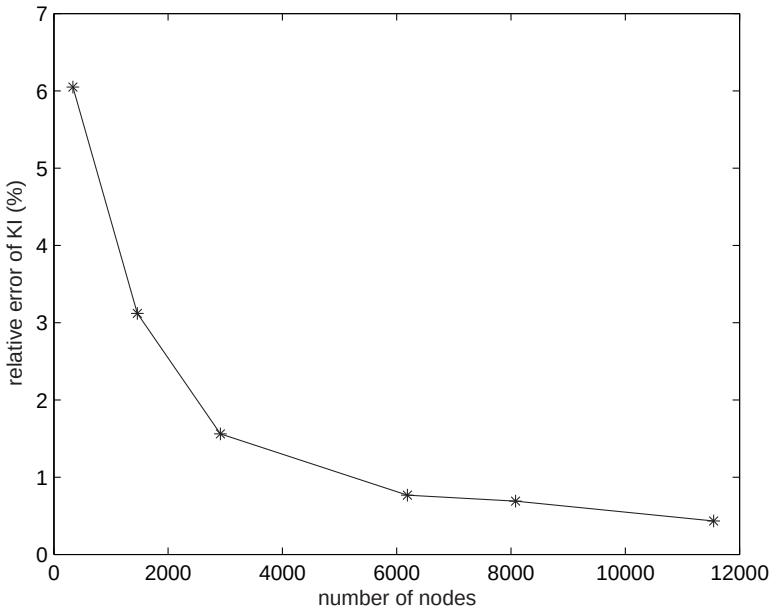


Figure 4.11: Convergence of K_I of edge cracked plate in tension.

intensity factors as given in J Yau, and Corten (1980) are:

$$\begin{aligned} K_I &= 34.0 \quad psi\sqrt{in} \\ K_{II} &= 4.55 \quad psi\sqrt{in} \end{aligned} \tag{4.4}$$

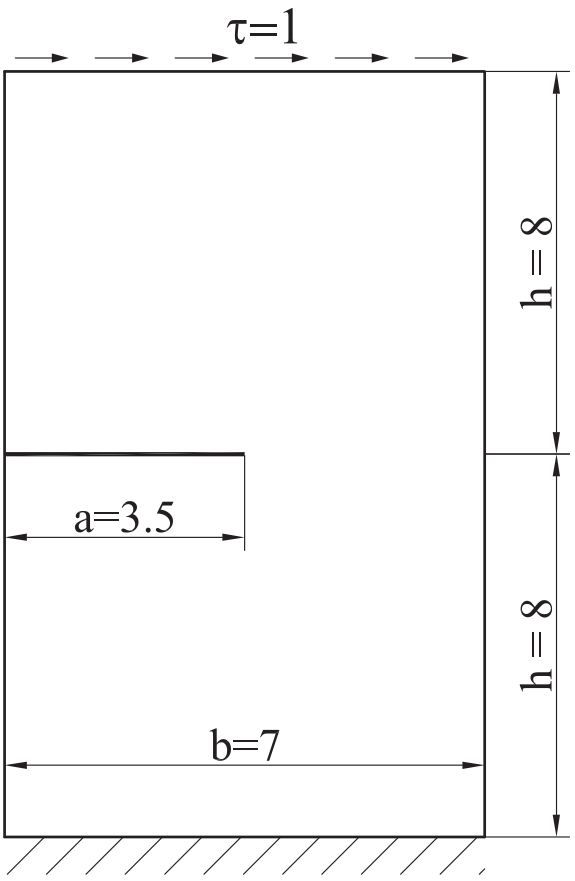
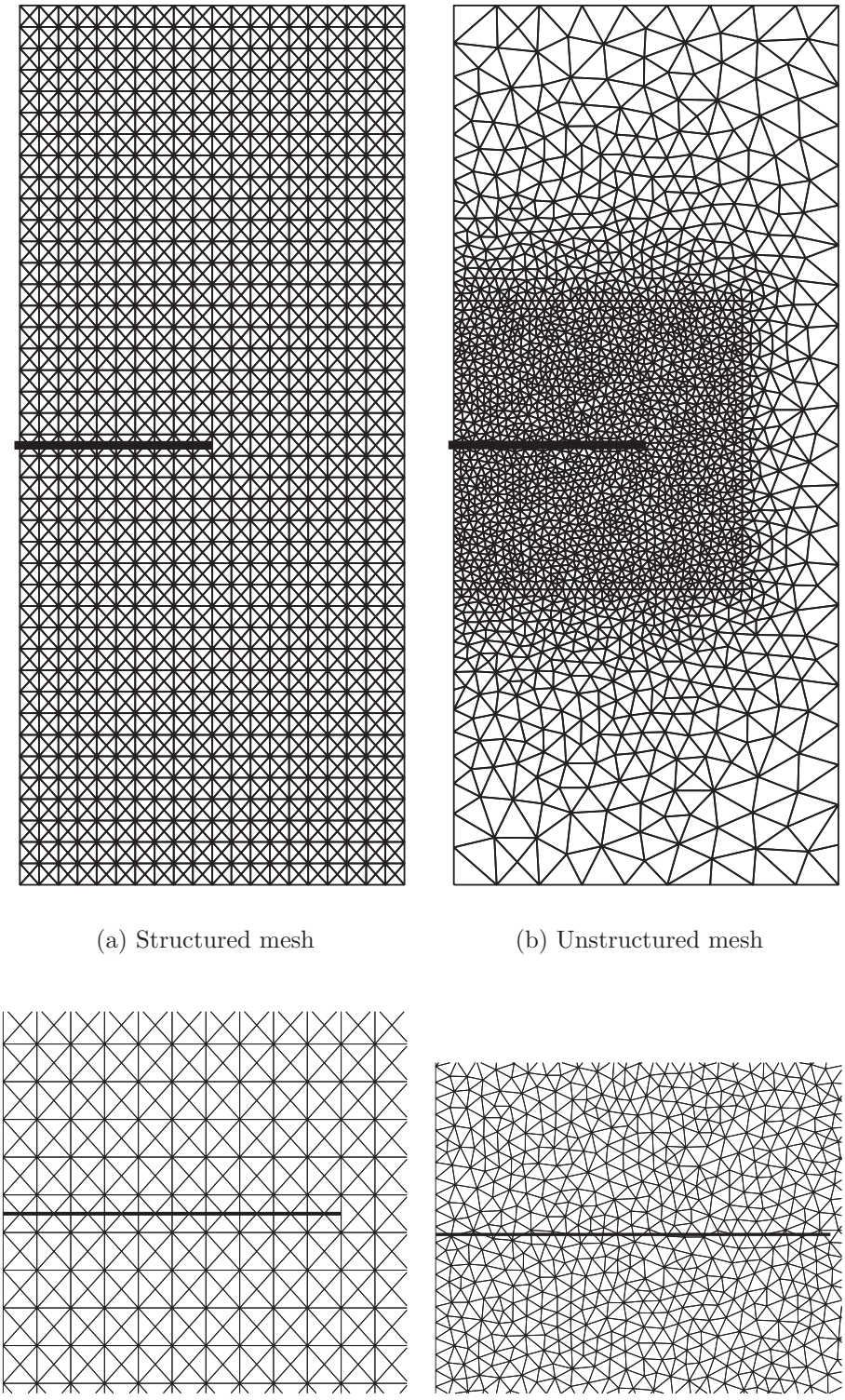


Figure 4.12: Geometry and load of the shear edge crack problem

In this example, the effects of the mesh refinement, the domain independence of



(a) Structured mesh

(b) Unstructured mesh

(c) Structured mesh, zoom on crack area

(d) Unstructured mesh, zoom on crack area

Figure 4.13: Finite element meshes used for shear edge crack problem.

the J integral, and fixed enrichment area scheme are studied on the mixed mode SIFs. Firstly, the domain independence of the J integral computation is studied with both structured mesh and unstructured mesh. The structured mesh consists of 3280 triangle elements, while unstructured mesh consists of 4379 elements as shown in Figure 4.13. In the table of result which follows, these meshes are denoted by *struct* and *unstr*. From Table 4.5, one can easily see the independence of the SIFs with respect to the domain contours used.

Table 4.5: Normalized SIFs $F_1 = K_I/34$ and $F_2 = K_{II}/4.55$ for various meshes and domain sizes.

Mesh	Normalized SIFs	r_d/h_{local}					
		1.5	2.0	2.5	3.0	3.5	4.0
struct	F1	0.9900	0.9900	0.9645	0.9641	0.9664	0.9592
	F2	1.0076	1.0076	0.9815	0.9796	0.9788	0.9850
unstr	F1	1.0053	0.9856	0.9844	0.9768	0.9788	0.9767
	F2	0.9504	1.0336	1.0176	1.0096	0.9933	0.9919

The convergence of the stress intensity factors is studied for this example. The stress intensity factors are computed for various structured meshes with a domain size of $r_d = 3h_{local}$. The result is tabulated in Table 4.6 and also illustrated in Figure 4.14. From this figure, one can, one more time, observe that, although the result is converged, the convergence rate is very slow when the mesh is fine.

This example is solved again but with uniform Q4[1] meshes as shown in Figure 4.15. The purpose is: (1) to present the Gauss quadrature rules used for rectangular Q4 mesh and (2) to analyze the influence of the location of the crack tip with respect to the mesh on the values of the computed SIFs.

The Gauss quadrature for X-FEM with Q4 elements is as follow, see Figure

[1]four-noded quadrilateral elements

Table 4.6: SIFs results of an edge crack plate under shear.

Num. elements	K_I	$\frac{K_I - 34}{34}(\%)$	K_{II}	$\frac{K_{II} - 4.55}{4.55}(\%)$
352	28.6171	15.83	4.3005	5.48
1176	31.5152	7.31	4.3992	3.31
3280	32.7794	3.59	4.4572	2.04
6936	34.0952	0.28	4.5837	0.74
10736	34.0272	0.08	4.5559	0.13
20032	33.9251	0.22	4.5180	0.70

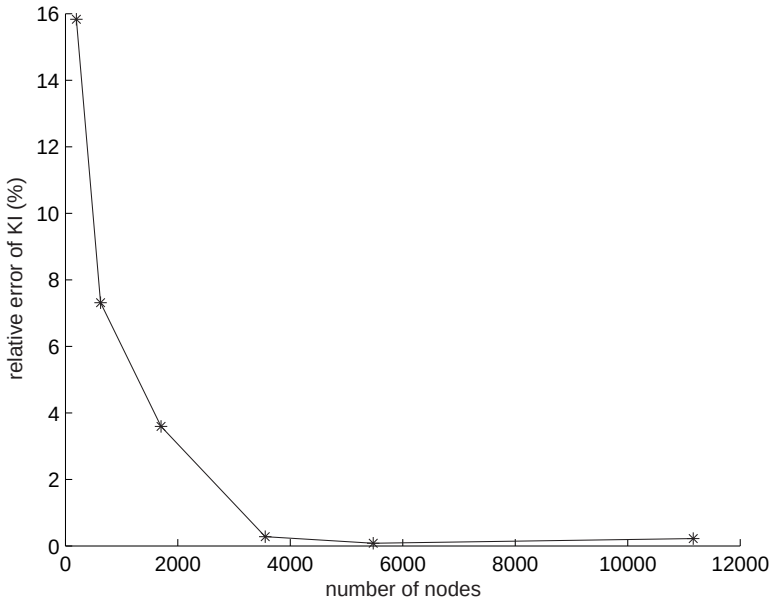
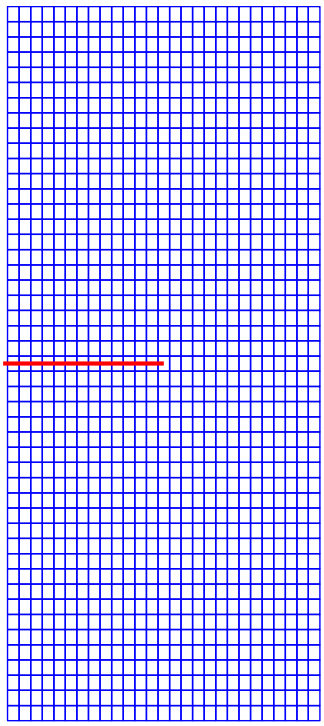
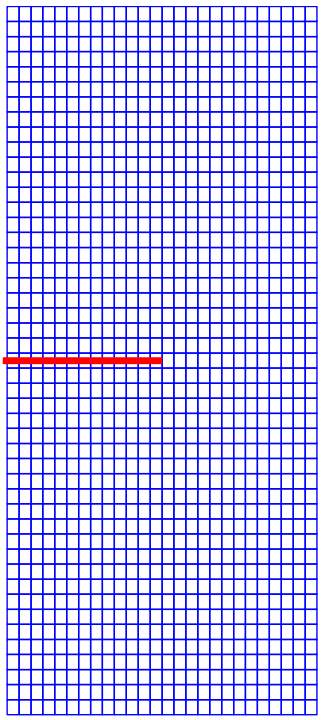


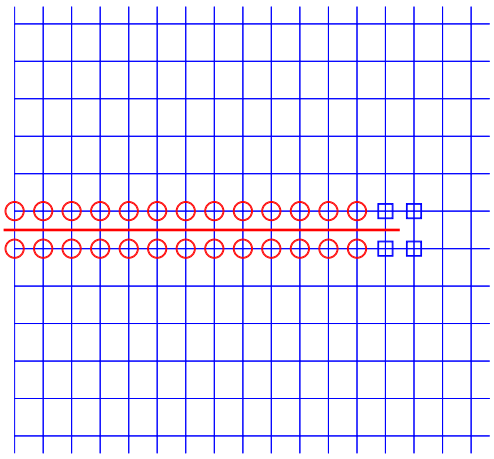
Figure 4.14: Convergence of K_I of shear edge crack



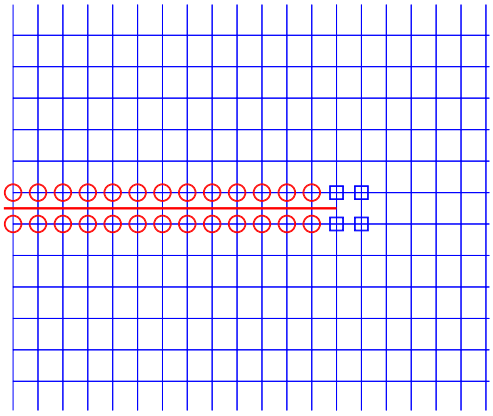
(a) Tip falls within element (27x48)



(b) Tip touches element edge (28x48)

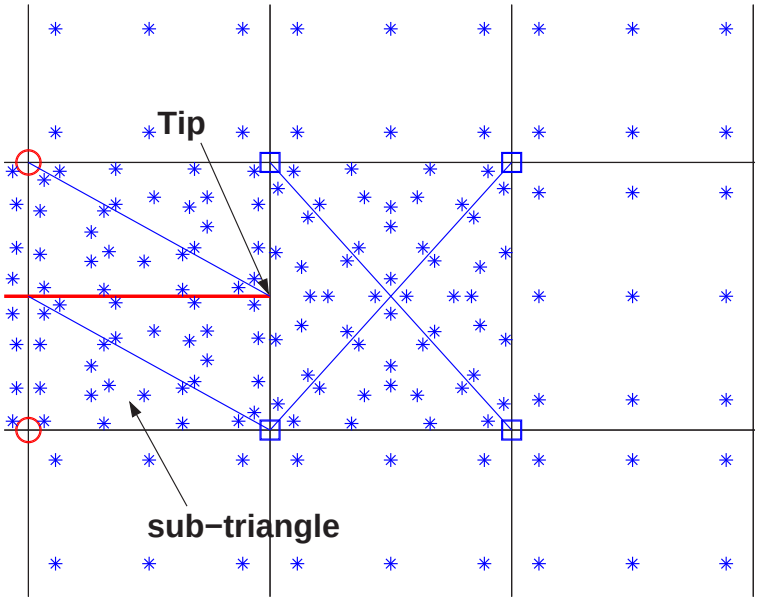


(c) zoom on crack area for left mesh

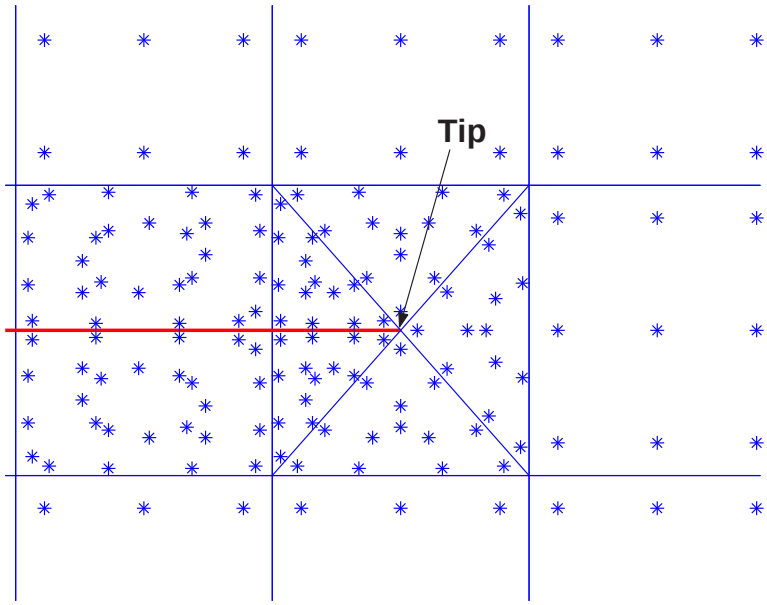


(d) zoom on crack area for right mesh

Figure 4.15: Uniform Q4 finite element meshes used for shear edge crack problem.



(a) Tip touches element edge



(b) Tip falls within element

Figure 4.16: Gauss points used for Q4 elements : (a) Tip element is partitioned into four sub-triangles; and (b) Tip element is decomposed into five sub-triangles. For split elements, four sub-triangles are built and 13 GPs were used for each sub-triangle.

4.16 :

- Non-enriched elements : standard 2×2 Gauss points (GP)
- Partially tip enriched elements : 3×3 Gauss points
- Partially step enriched elements : 2×2 Gauss points
- Split elements : thirteen GPs for each sub-triangle, the total number of GP is $13 \times 4 = 42$
- Tip elements : if the tip falls within an element, five sub-triangles are built and thirteen GPs used for each sub-triangle. Therefore, the total number of GP is $13 \times 5 = 65$. If the tip touches an element edge, then, to have enough GPs for the numerical integration, this element is partitioned into four triangles (the center of element is used in the Delaunay triangulation). The total GPs are $13 \times 4 = 42$.

For the computation of the interaction integral, 3×3 Gauss quadrature rule is used for elements not cut by the crack. Split elements are decomposed into 4 sub-triangles in which 13 GPs are used. The stress intensity factors are computed for various discretizations. The domain size is $r_d = 3h_{local}$ for all meshes. The results are given in Table 4.7 allow us conclude that the X-FEM is robust, i.e., the location of the crack tip with respect to the mesh (here we examined two cases, the tip falls within element and the tip touches element edge) has no considerable impact on the results.

Probably, similar results could be obtained for a crack aligned with element edges. Our present implementation does not allow the crack to be aligned with

Table 4.7: Normalized SIFs of shear edge crack with Q4 elements

mesh	F_1	F_2
28 x 48	0.9734	0.9915
27 x 48	0.9942	0.9939
48 x 96	0.9894	0.9929
47 x 96	0.9981	0.9967

element edges. This is a limitation of the code and will be improved in the near future.

4.1.4 Center-crack in tension

Considering a finite plate with a center crack in tension. The geometry of the plate is described in Figure 4.17. The analitycal stress intensity factors for this problem are:

$$K_I = \sigma\sqrt{\left(\pi a sec\left(\frac{\pi a}{2w}\right)\right)} \quad K_{II} = 0 \tag{4.5}$$

where a is the half crack length and $w = W/2$ is the half width of the plate, and σ is the tensile load applied at the top of the plate.

In Figure 4.18, the structured and unstructured meshes used for the analysis are illustrated. The symmetry was not taken into account to verify the same behavior of two crack tips. The SIFs obtained with structured meshes of 836, 3572, and 10800 elements are tabulated in Table 4.8 and the SIFs computed with unstructured meshes are given in Table 4.9. It is important to mention that the difference between the stress intensity factors computed at the two crack tips is below 0.5%.

The next part of this example is about the fixed enrichment area. Both

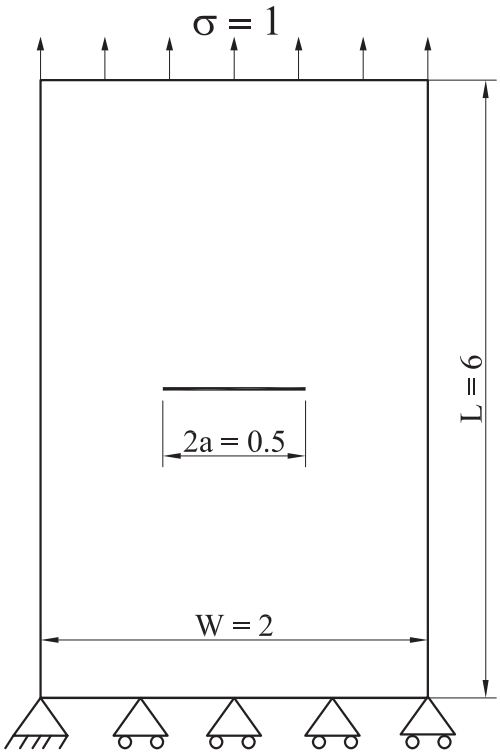
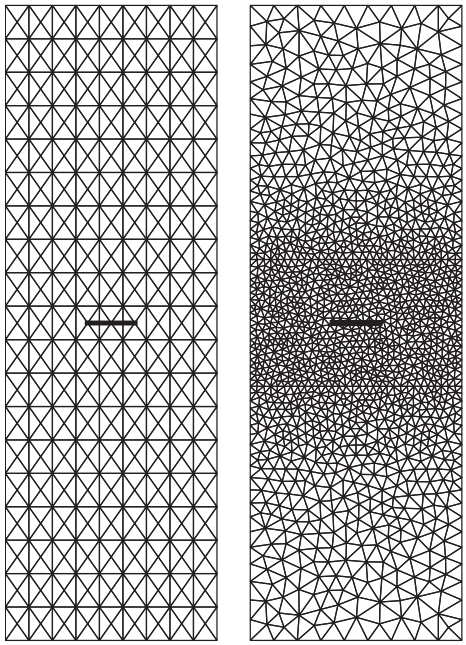


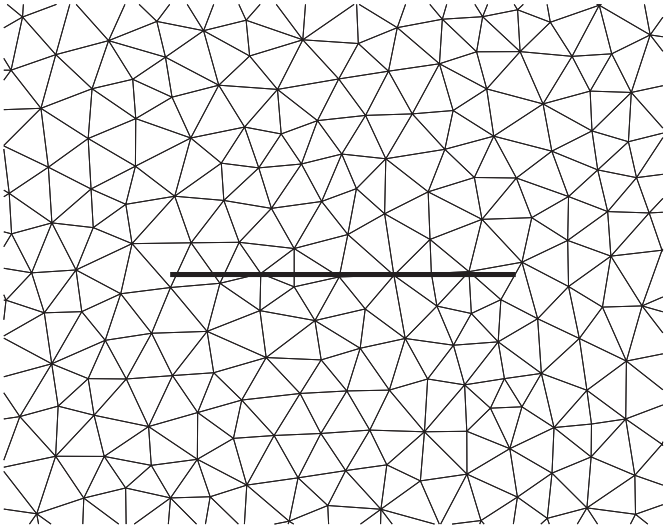
Figure 4.17: Geometry and loads of a center-cracked plate in tension

Table 4.8: Normalized SIFs for center cracked plate in tension (structured mesh)

Num. elements		r_d/h_{local}					
		1.5	2.0	2.5	3.0	3.5	4.0
836	tip1	1.0759	1.0520	1.0611	1.0686	1.0874	1.1292
	tip2	1.0759	1.0520	1.0611	1.0686	1.0874	1.1292
3572	tip1	1.0110	0.9770	0.9907	0.9907	0.9922	0.9868
10800	tip1	0.9736	1.0003	0.9980	0.9976	0.9973	0.9940



(a) (b)



(c)

Figure 4.18: Finite element meshes for the center-cracked plate in tension: (a) Structured mesh ; (b) Unstructured mesh and (c) unstructured mesh (vicinity of the crack)

Table 4.9: Normalized SIFs for center cracked plate in tension (unstructured mesh)

Num. elements		r_d/h_{local}					
		1.5	2.0	2.5	3.0	3.5	4.0
906	tip1	0.9754	0.9826	0.9826	0.9782	0.9754	0.9777
	tip2	1.0286	0.9857	0.9834	0.9837	0.9810	0.9830
3314	tip1	1.0108	0.9977	0.9927	0.9930	0.9902	0.9908
	tip2	1.0107	0.9983	0.9944	0.9888	0.9909	0.9927
11218	tip1	1.0161	1.0138	0.9945	0.9948	0.9968	0.9957
	tip2	1.0047	1.0046	0.9952	0.9906	0.9926	0.9942

structured and unstructured meshes are investigated. The enrichment radius is $r_{enr} = 3h_{local}$. From Table 4.10, one more time, we observe that to get good SIFs, the domain radius r_d should be chosen equal or greater than the enrichment radius r_{enr}.

The convergence of the stress intensity factors is examined with unstructured meshes. The normal enrichment scheme was used. The domain size is $r_d = 3h_{local}$ for all meshes. The result is tabulated in Table 4.11 and also illustrated in Figure 4.19. From this figure, one can observe that, although the result is converged, the convergence rate is very slow when the mesh is fine.

Table 4.10: Center cracked plate using the fixed enrichment area ($r_{enr} = 3h_{local}$)

Mesh	Num. elements		r_d/h_{local}				
			1.5	2.0	2.5	3.0	3.5
struct	3572		0.9493	0.9926	1.0042	1.0042	0.9998
unstr	3314	tip1	0.9575	0.9754	0.9968	1.0085	1.0042
		tip2	0.9702	0.9766	0.9976	1.0058	1.0008

Table 4.11: SIFs convergence of the center cracked plate

Num. nodes	K_{exact}	crack tips			
		tip1		tip2	
		K_I	$\frac{K_I - K_{exact}}{K_{exact}}(\%)$	K_I	$\frac{K_I - K_{exact}}{K_{exact}}(\%)$
198	0.9220	0.9783	5.60	0.9783	6.11
488	0.9220	0.9019	2.18	0.9070	1.63
579	0.9220	0.9101	1.29	0.9085	1.46
1713	0.9220	0.9156	0.69	0.9116	1.13
2555	0.9220	0.9146	0.80	0.9145	0.81
5716	0.9220	0.9172	0.52	0.9134	0.93
10005	0.9220	0.9207	0.14	0.9207	0.16

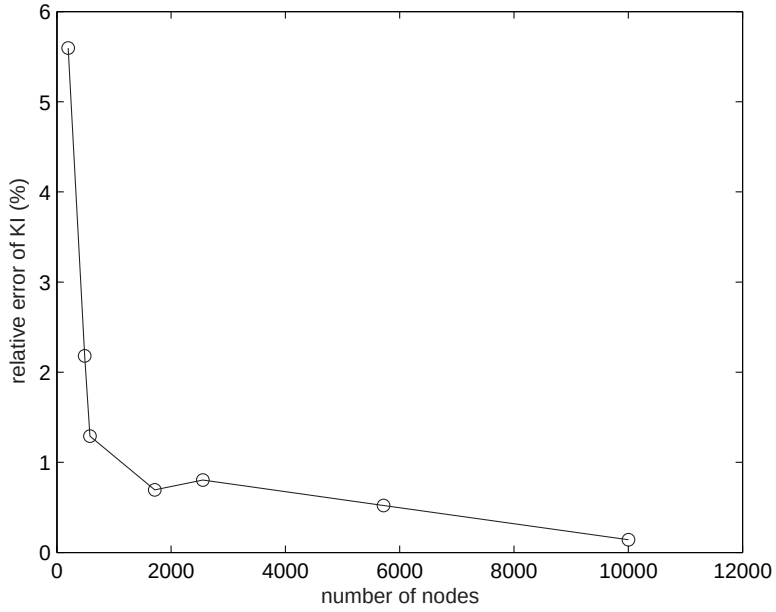


Figure 4.19: Convergence of K_I of center cracked plate in tension

4.1.5 Inclined crack in tension

To illustrate the versatility and effectiveness of the X-FEM, stress intensity factors are calculated for a plate with an angled center crack shown in Figure 4.20. The plate is subjected to a far field uniaxial stress as shown. In this example, K_I and K_{II} are obtained as a function of the crack angle β for a structured triangular mesh. It is emphasized that the same mesh shown in Figure 4.21, is used for all angles considered.

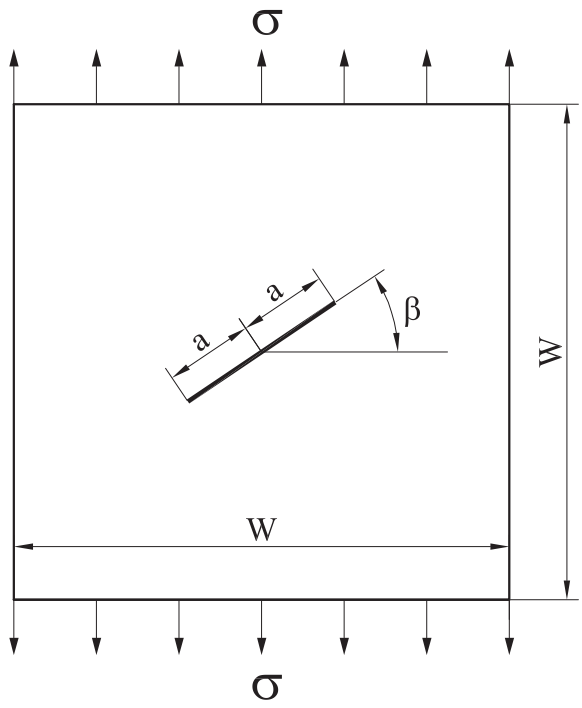
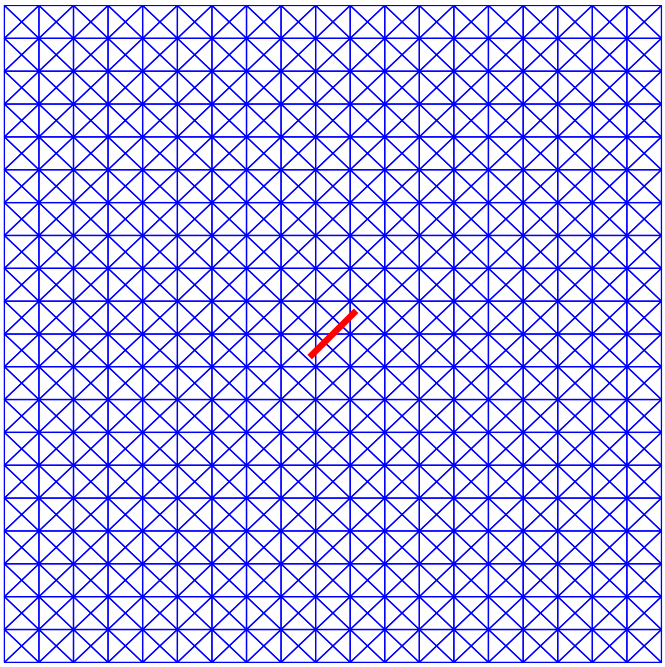
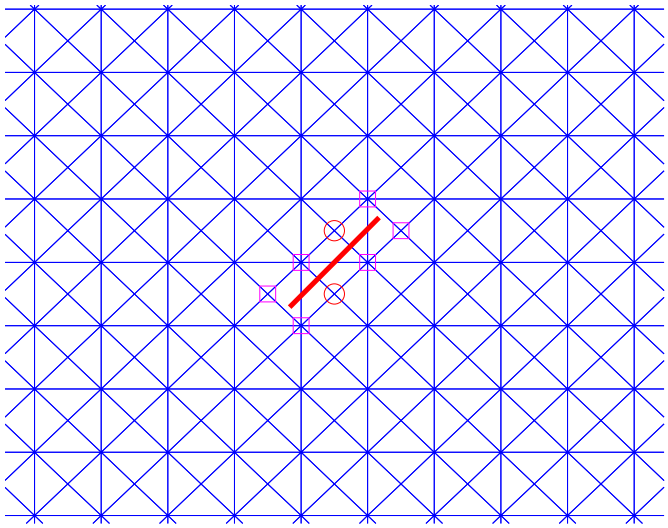


Figure 4.20: Inclined crack in tension

In this example, the plate dimensions are taken to be $W = 10$ in. with a half crack length of $a = 0.5$ in. As the plate dimensions are large in comparison to



(a) The fixed mesh used for all angles β



(b) Enriched nodes for $\beta = 45^o$

Figure 4.21: Coarse structured mesh of 1520 elements for inclined crack problem

the crack length, the numerical solution can be compared to the solution for an infinite plate. For the load shown, the exact stress intensity factors are given by (Sih et al.,1962)

$$K_I = \sigma\sqrt{\pi a}\cos^2\beta, \quad K_{II} = \sigma\sqrt{\pi a}\sin\beta\cos\beta \tag{4.6}$$

Numerical results for the SIFs are obtained for $\beta = 15^o$, 30^o, 45^o, 60^o, 70^o, and domain independence of the J integral computation is also studied. In Table 4.12, the normalized SIFs are compared to the exact solution. Excellent agreement between the numerical solution and the exact solution is obtained. However, also from this table, one can observe that with $r_d/h_{local} = 3.0$ and 3.5, the obtained SIFs are not good. The reason for this is that the domain size is big enough to include the tip-element as shown in Figure 4.22.

The SIFs are also computed for other angles with the domain size used in the interaction integral computation is $r_d = 2.5h_{local}$ and plotted in Figure 4.23. The result shows excellent agreement with the exact solution for the entire range of β.

The convergence of the stress intensity factors is examined with structured meshes for $\beta = 45^o$. The standard enrichment scheme was used and the domain size was $r_d = 3h_{local}$ for all meshes. The result is given in Table 4.13 and also illustrated in Figure 4.24. Again, one can notice that the convergence rate is very slow. The reason for this is most likely the fact that the area of the enrichment zone decreases with mesh size.

Table 4.12: Normalized SIFs for the inclined crack problem

β	SIFs	Exact	X-FEM				
			$r_d/h_{local} = 1.5$	2.0	2.5	3.0	3.5
15^o	$\frac{K_I}{\sigma\sqrt{\pi a}}$	0.9330	0.9313	0.9316	0.9312	0.9791	0.9882
	$\frac{K_{II}}{\sigma\sqrt{\pi a}}$	0.2500	0.2760	0.2512	0.2489	0.2607	0.2613
30^o	$\frac{K_I}{\sigma\sqrt{\pi a}}$	0.7500	0.7232	0.7486	0.7484	0.7787	0.7770
	$\frac{K_{II}}{\sigma\sqrt{\pi a}}$	0.4330	0.4028	0.4413	0.4413	0.4455	0.4427
45^o	$\frac{K_I}{\sigma\sqrt{\pi a}}$	0.5000	0.4836	0.4897	0.5010	0.5159	0.5132
	$\frac{K_{II}}{\sigma\sqrt{\pi a}}$	0.5000	0.4767	0.5009	0.5022	0.5132	0.5108
60^o	$\frac{K_I}{\sigma\sqrt{\pi a}}$	0.2500	0.2000	0.2581	0.2549	0.2582	0.2596
	$\frac{K_{II}}{\sigma\sqrt{\pi a}}$	0.4330	0.3783	0.4406	0.4366	0.4563	0.4547
75^o	$\frac{K_I}{\sigma\sqrt{\pi a}}$	0.0670	0.0589	0.0673	0.0690	0.0692	0.0689
	$\frac{K_{II}}{\sigma\sqrt{\pi a}}$	0.2500	0.2115	0.2526	0.2535	0.2565	0.2560

Table 4.13: Convergence of SIFs for a plate with an angle center crack

Num. nodes	K_I	$\frac{K_I - K_{exact}}{K_{exact}}\%$	K_{II}	$\frac{K_{II} - K_{exact}}{K_{exact}}\%$
338	0.7288	16.29	0.7384	17.82
578	0.6657	6.22	0.6701	6.93
1250	0.6242	0.40	0.6209	0.93
2178	0.6315	0.76	0.6262	0.08
3042	0.6238	0.46	0.6154	1.80
6050	0.6308	0.65	0.6250	0.27
8450	0.6299	0.51	0.6241	0.41
11250	0.6282	0.24	0.6258	0.14

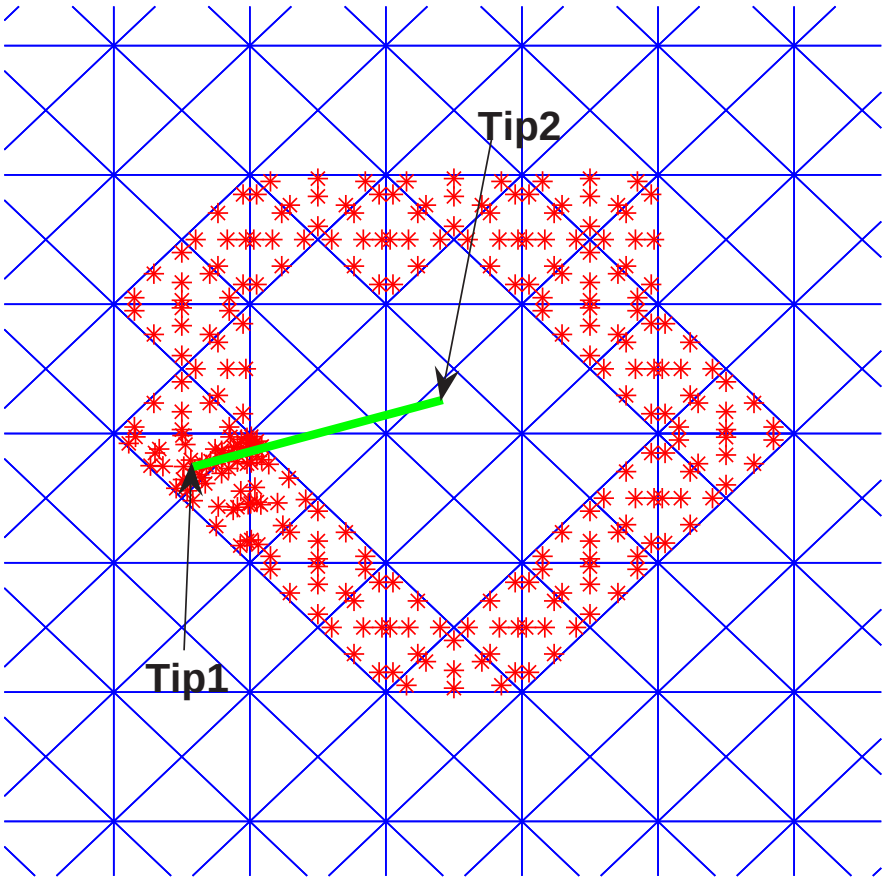


Figure 4.22: The J integral domain contains the tip-element gave incorrect SIFs due to the singular nature of the tip

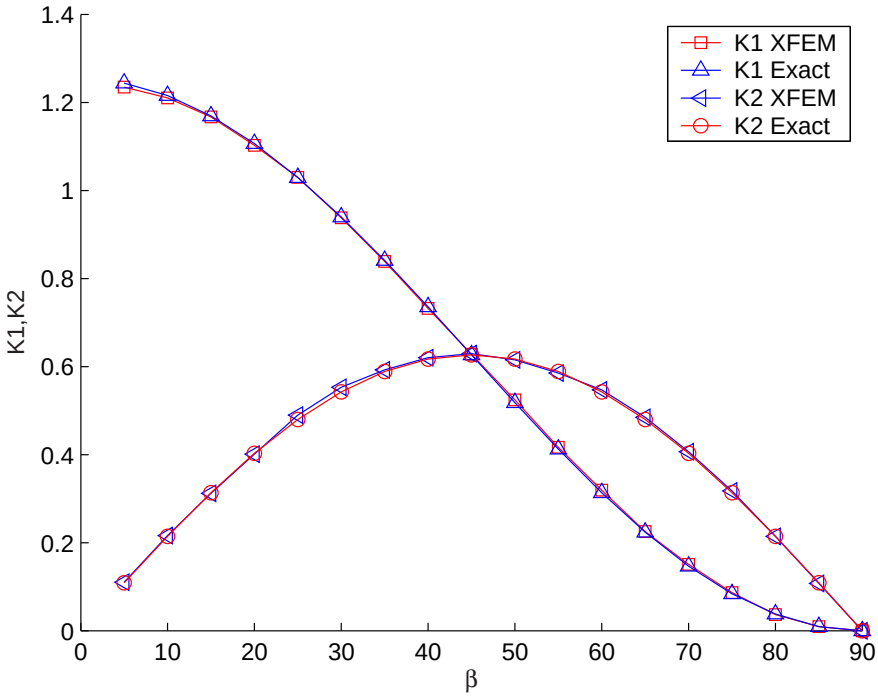


Figure 4.23: K_I and K_{II} vs. β for a plate with an angle center crack

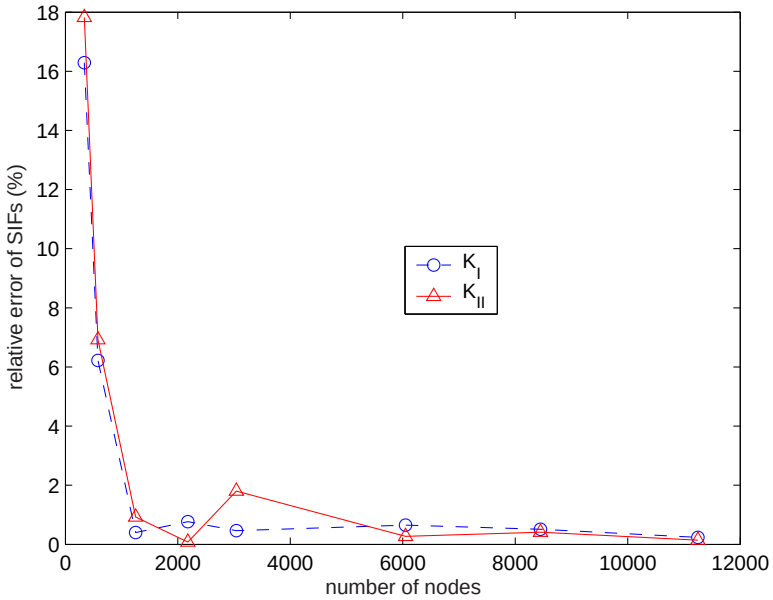


Figure 4.24: Convergence of SIFs of inclined crack in tension

4.1.6 Curved crack in an infinite plate

A curved crack in an infinite plate is considered. A finite plate model with a large edge length to crack length ratio (> 10) was used as shown in Figure 4.25. The analytical stress intensity factors, as given in Gdoutos (1979), are:

$$\begin{aligned} K_I &= \frac{\sigma}{2}\sqrt{\pi R \sin\beta}\left[\frac{[1-\sin^2(\beta/2)\cos^2(\beta/2)]\cos(\beta/2))}{1+\sin^2(\beta/2)} + \cos(3\beta/2)\right] \\ K_{II} &= \frac{\sigma}{2}\sqrt{\pi R \sin\beta}\left[\frac{[1-\sin^2(\beta/2)\cos^2(\beta/2)]\sin(\beta/2))}{1+\sin^2(\beta/2)} + \sin(3\beta/2)\right] \end{aligned} \tag{4.7}$$

where R is the radius of the circular arc and 2β is the subtended angle of the arc. The computations were run for $R = 4.25$ and $\beta = 28.0725^o$. The reference stress intensity factors are then $K_I = 2.0146$ and $K_{II} = 1.1116$.

In order to capture the curvature of the crack, a relatively fine mesh (Figure 4.26a) was used together with the division of the crack into several line segments. Here, the crack is represented by a polyline of 20 segments (see Figure 4.26b).

The stress intensity factors are computed for various domain sizes and given in Table 4.14. One can observe that the SIFs are not independent to the domain used. This is not surprise since the J integral is path-independent only for straight cracks. The use of appropriate path-independent integrals for curved (circular arc-shaped) cracks (M. Lorentzon, 2000) is required to attain domain independence in the SIF computations.

4.1.7 Cracks emanating from circular hole in infinite plate

As a last example, considering a symmetric double crack at a circular hole of radius $r = 2$ in the middle of an infinite plate as shown in Figure 4.27. The reference

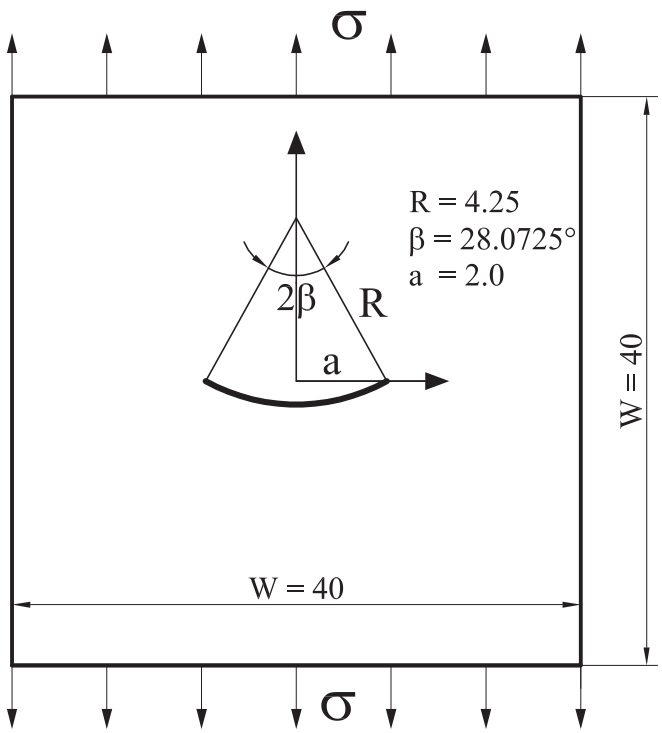
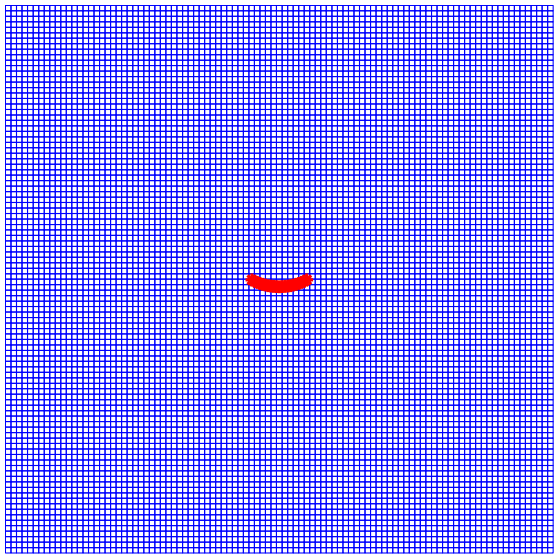


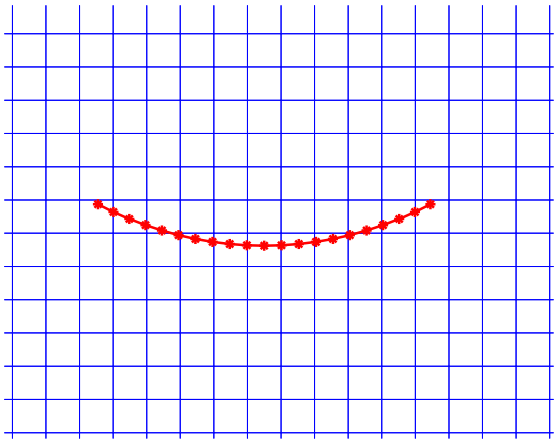
Figure 4.25: Curved crack in an infinite plate in tension

r_d/h_{local}	K_I	Error on K_I (%)	K_{II}	Error on K_{II} (%)	Error on K_{eq} (%)
1.5	2.0876	3.62	1.0882	2.10	4.18
2.0	2.0656	2.54	1.1039	0.69	2.63
2.5	2.0650	2.50	1.1090	0.24	2.51
3.0	2.0529	1.90	1.1510	3.54	4.02
3.5	2.0443	1.48	1.1808	6.22	6.39
4.0	2.0122	0.12	1.2516	12.59	12.59
4.5	2.0100	0.23	1.2545	12.85	12.85
5.0	2.0000	0.72	1.2769	14.87	14.89
5.5	1.9893	1.26	1.2917	16.20	16.25
6.0	1.9883	1.30	1.2976	16.73	16.78

Table 4.14: Stress intensity factors of curved crack problem



(a)



(b)

Figure 4.26: Discretization of the plate: (a) Uniform mesh consisting of 9900 elements; and (b) Zoom around crack area

solution used here is given in (Abdel-Rahman Ragab, 1999):

$$K_I = \left(\frac{r}{a} + 1\right)^{1/2} \sigma\sqrt{\pi a} \quad \text{with } a > 0.12r \tag{4.8}$$

for $a = 1$ and with the above data, the reference mode I stress intensity factor is $K_I = 3.06998$. Note that the expression given by equation (4.8) is just an approximation.

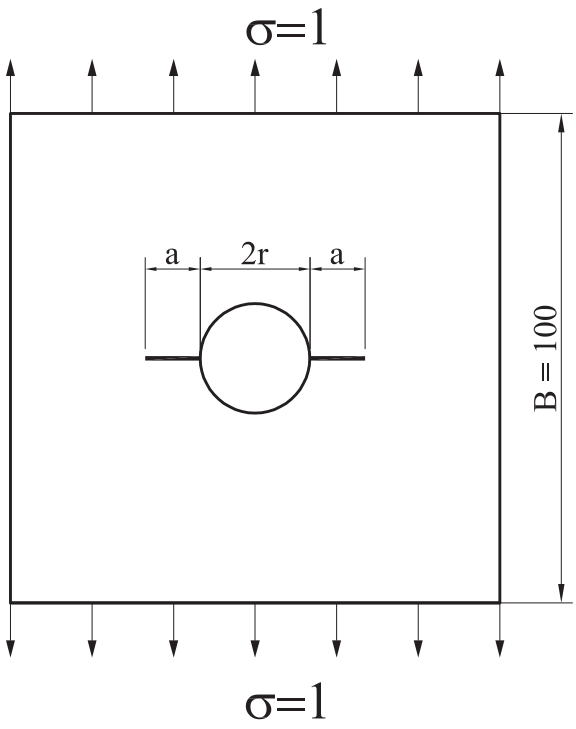
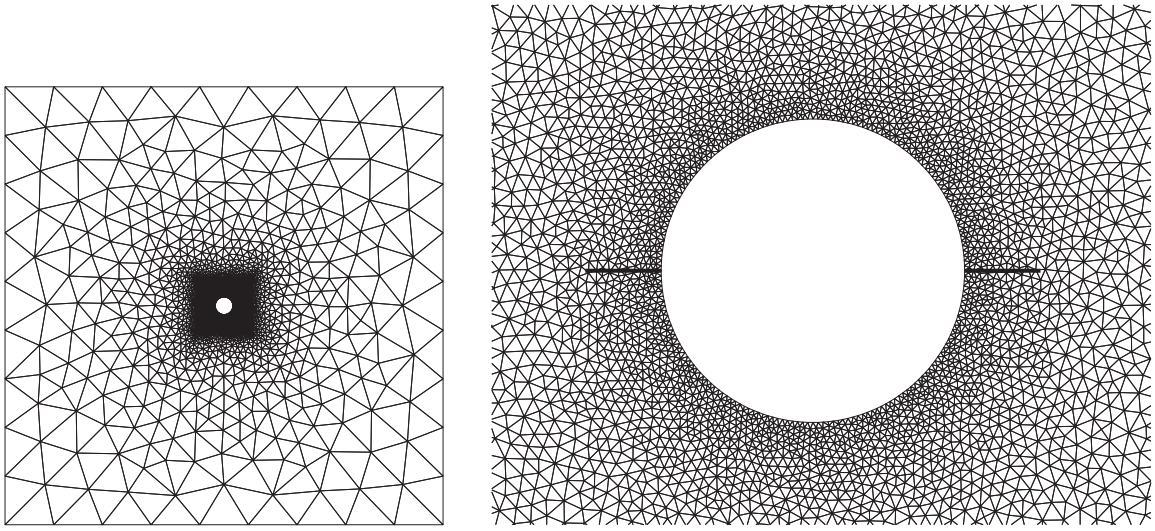


Figure 4.27: Symmetric double crack at a circular hole in an infinite plate

The traction boundary conditions are applied on the bottom and top edges and the plate is adequately restrained to preserve the symmetry of the problem while avoiding rigid body motion. The domain size used to compute the SIFs is

$r_d = 5 * h_{local}$. The result given in Table 4.15 shows the same behavior of the two crack tips. The obtained SIF is not good (relative error about 5%). The result will be improved with finer mesh.

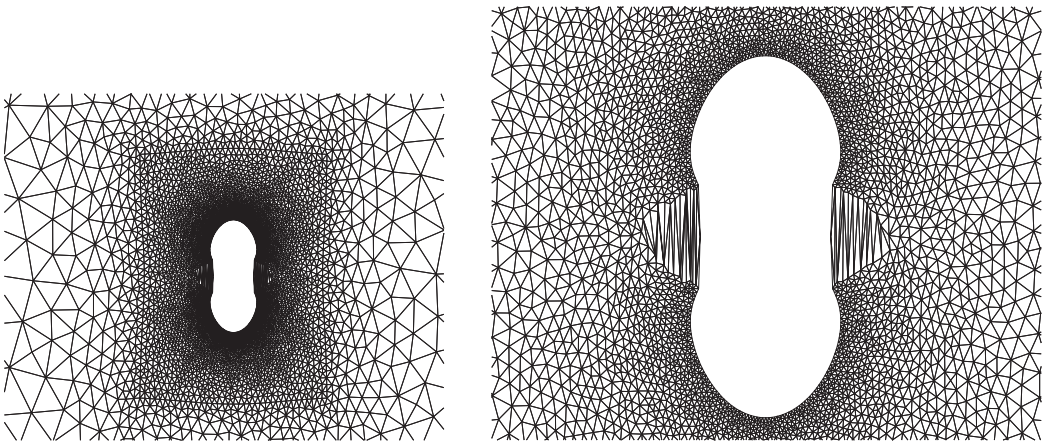


(a) (b)

Figure 4.28: Cracks emanating from circular hole: (a) Unstructured mesh consisting of 10216 elements; and (b) zoom on the crack area

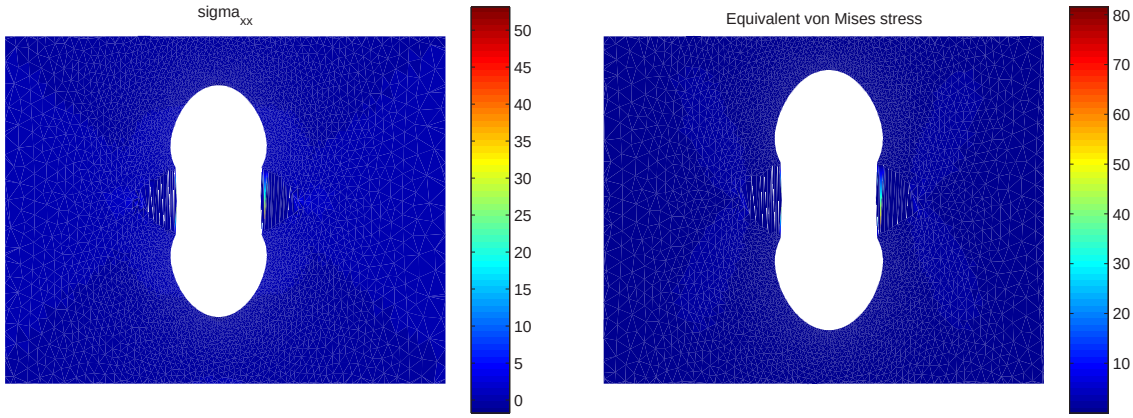
Table 4.15: Stress intensity factors of cracks emanating from hole problem.

Num. nodes	K_{exact}	crack tips			
		tip1		tip2	
		K_I	$\frac{K_I - K_{exact}}{K_{exact}}(\%)$	K_I	$\frac{K_I - K_{exact}}{K_{exact}}(\%)$
5212	3.0699	3.2472	5.76	3.2521	5.94



(a) (b)

Figure 4.29: Deformed configuration of cracks-hole problem



(a) (b)

Figure 4.30: Zoom of the stress σ_{xx} and von Mises stress contours near the hole.

4.2 Crack growth problems

This section will illustrate the main advantage of the X-FEM over other numerical methods for crack growth simulations. This is the ability to model crack growth without remeshing. Examples given here are only quasi-static crack growth problems. However, note that it is not a limitation of the X-FEM. Dynamic crack growth with X-FEM was studied in Chen and Belytschko 2003. It is also emphasized that the current implementation does not allow cracks to intersect as they grow. Again, it is not a drawback of the X-FEM. Multiple crack growth simulation using X-FEM which allows intersecting cracks was examined in the work of Budyn (2004).

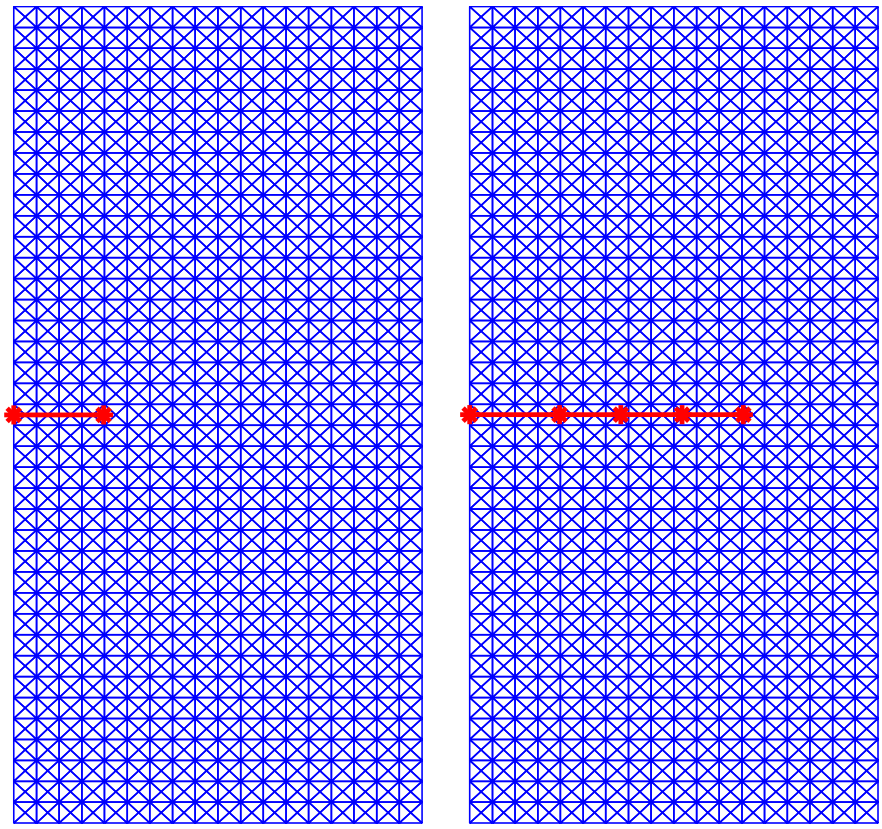
4.2.1 Growth of an edge crack in tension

As the first example on numerical simulation of crack growth, considering the edge cracked plate in tension (see Section 4.1.2). All information is identical to that example except for the initial crack length, which is now 0.22.

The plate is modeled by a structured triangular mesh consisting of 2808 elements. The crack growth direction is computed from the stress intensity factors in according to the maximum hoop stress criteria. In each step, the crack grows a distance of 0.15. The example is run for a total of four steps.

Step	Tip position	K_I	Exact SIF	Error (%)
1	(0.22,1.0)	1.1557	1.1803	2.08
2	(0.37,1.0)	2.0539	2.1000	2.19
3	(0.52,1.0)	3.7474	3.8607	2.93
4	(0.67,1.0)	7.5327	7.6784	1.90

Table 4.16: SIFs and evolution of crack tips for the edge crack



(a) (b)

Figure 4.31: Propagation of an edge crack in tension: (a) Initial crack ; and (b) crack path after four steps.

4.2.2 Inclined crack in tension

The static analysis of an angled center crack in an infinite plate was considered in Section 4.1. The geometry and loads are given again for convenience (see Figure 4.32). For the computation, β was chosen to be 45^o, the structured mesh consisting of 1520 elements (see Figure 4.33(a)). The domain size used to compute the interaction integral is $r_d = 2.5h_{local}$. The example is run for a total of four steps. The crack increment length for each step is 0.38. We can see in Figure 4.33(b) that the crack grows at both tips symmetrically towards the preferential growth direction, mode I. Similar results were obtained in Stolarska, Chopp, Moës, and Belytschko (2001). Note that these good results were obtained for the very coarse mesh of Figure 4.33, featuring only two elements along the crack length, which is quite remarkable.

4.2.3 Double cantilever beam

In Figure 4.34, a double cantilever beam (DCB) is illustrated. The specimen dimensions are 6×2, and the initial pre-crack with length of $a = 2.05$ is placed slightly above the mid-plane of the beam. The material properties are taken to be $E = 100$, and $\nu = 0.3$, the load P is taken to be 1. By symmetry, a crack on the mid-plane of the plate is under pure mode I, and it would propagate straight ahead. However, this crack path is unstable, small perturbations in either loading or crack geometry trigger curvilinear crack growth.

Quasi-static crack growth is governed by the maximum hoop stress criterion, and the crack growth increment is $\Delta a = 0.15$. In Figure 4.36, a representative

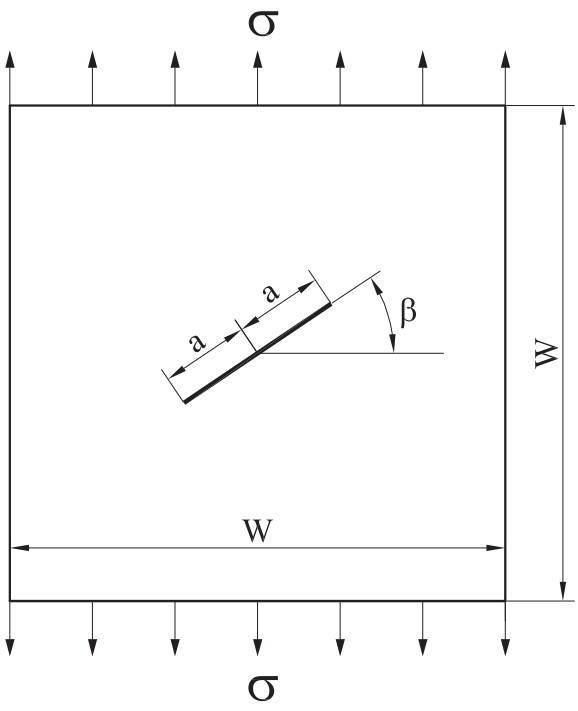
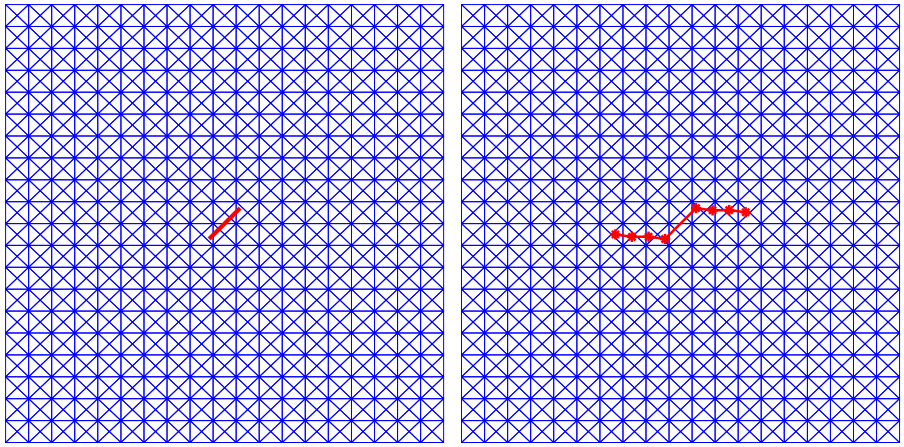


Figure 4.32: Inclined crack in tension



(a) (b)

Figure 4.33: Propagation of an inclined crack in tension: (a) Initial crack; and (b) crack path after four steps.

simulated crack growth for 11 steps is shown. The specimen was discretized by a structured mesh consisting of 1200 elements (see Figure 4.35). The SIFs are computed with domain size taken to be of $r_d = 2.5h_{local}$. The simulated crack path qualitatively agrees with published results, for example, ones of (Belytschko and Black, 1999).

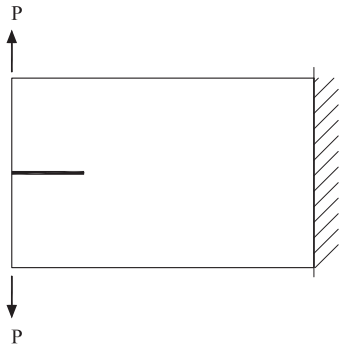
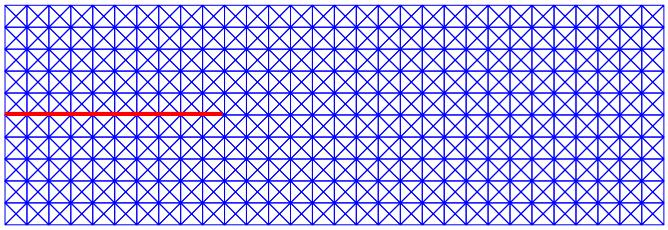
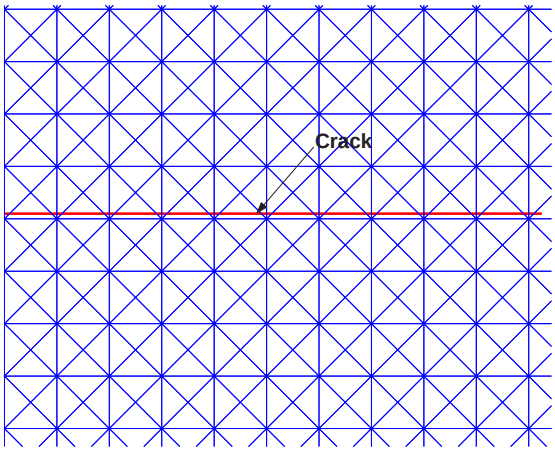


Figure 4.34: Geometry-loads of a double cantilever beam speciment



(a)



(b)

Figure 4.35: Fixed structured mesh used for the crack growth simulation of a DCB: (a) overall view ; (b) zoom on the crack area (the crack is placed slightly above the mid-plane of the beam)

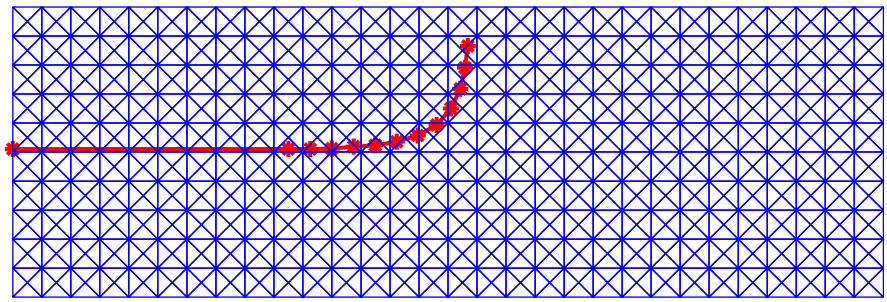
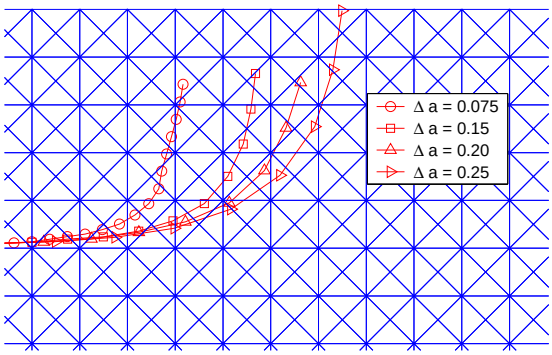


Figure 4.36: Curvilinear crack growth in double cantilever beam specimen

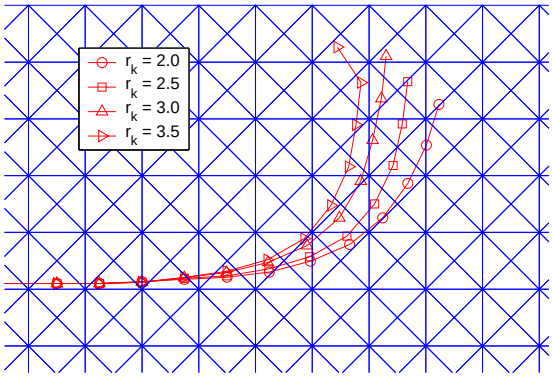
Parameters which effect the accuracy of the simulated crack path are (1) mesh refinement; (2) the domain size $r_d = r_k h_{local}$; and (3) the crack growth increment Δa. A series of computations were performed to study these effects. The numerical results are illustrated in Figure 4.37. In Figure 4.37a, crack growth paths obtained with four different crack growth increments are presented (1200 elements, $h_{local} = 0.1$). From this figure, it is observed that to get a converged crack path, the crack increment length should be between about half the element size h_{local} and one and half times the element size, i.e., $0.5h_{local} < \Delta a < 1.5h_{local}$. It is obvious that the crack increment length should be smaller than an upper limit ($1.5h_{local}$) since we are modeling a curve by straight segments. Accuracy is improved by using smaller Δa's. However, if the crack increment is too small compared to the element size, multiple changes in the direction of the crack path may occur. In addition, the element partitioning for numerical integration also becomes time consuming.

As pointed out in the example of the curved crack problem, the J integral is not path-independent for curved cracks. Therefore, the domain size r_d also effects the simulated crack path. This effect is shown in Figure 4.37(b). Although the effect of the domain size is small, the adoption of appropriate path-independent integrals for curved cracks is necessary for improvements in the crack growth capabilities of the X-FEM.

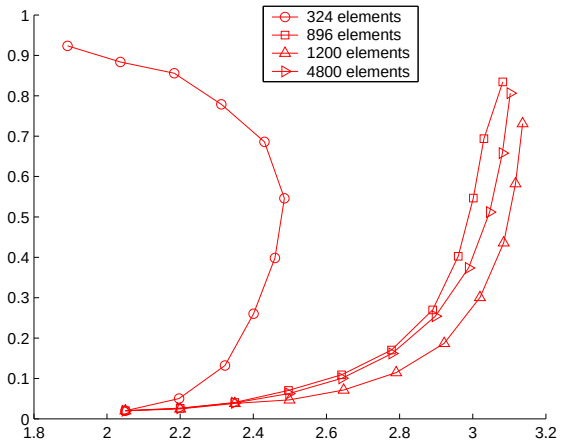
The influence of mesh refinement on the simulated crack path is examined using four different meshes. Results shown in Figure 4.37c allow us to conclude that, for a sufficiently refined mesh, the crack path is not mesh-sensitive.



(a)



(b)



(c)

Figure 4.37: Parametric study of simulated crack paths in double cantilever beam specimen: (a) influence of crack growth increment Δa (1200 elements, $r_k = 2.5$); (b) influence of domain radius (1200 elements, $\Delta a = 0.15$); and (c) influence of mesh refinement($r_k = 2.5$, $\Delta a = 0.15$)

4.2.4 Crack growth from a fillet

This example shows the propagation of a crack from a fillet in a structural member. The configuration to be studied is taken from experimental work found in (Sumi, Yang, and Wang, 1995) and shown in Figure 4.38. The purpose of this example is to investigate the effect of the thickness of the lower I-beam on crack growth. Only limiting cases for the bottom I-beam of a rigid constraint (very thick beam) and flexible constraint (very thin beam) are examined.

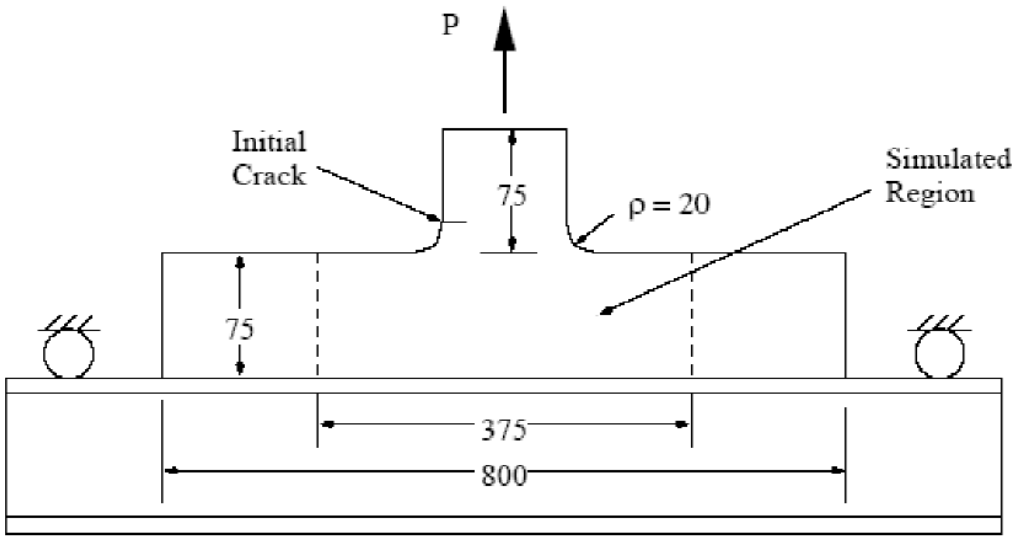


Figure 4.38: Experimental configuration for crack growth from a fillet, taken from (Dolbow, 1999)

The structure is loaded with a traction of $P = 20kN$, and the initial crack length is taken to be of $a = 5mm$. The computational domain is outlined by dashed lines. This domain is discretized with 7108 three-noded triangular elements

(see Figure 4.39). The effects of the thickness are incorporated into the problem through the Dirichlet boundary conditions. For a rigid I-beam, the displacement in the vertical direction is fixed on the entire bottom edge. A flexible beam is idealized by fixing the vertical displacement at only both endpoints of the bottom edge. For both sets of boundary conditions, an additional degree of freedom is fixed to prevent rigid body motions.

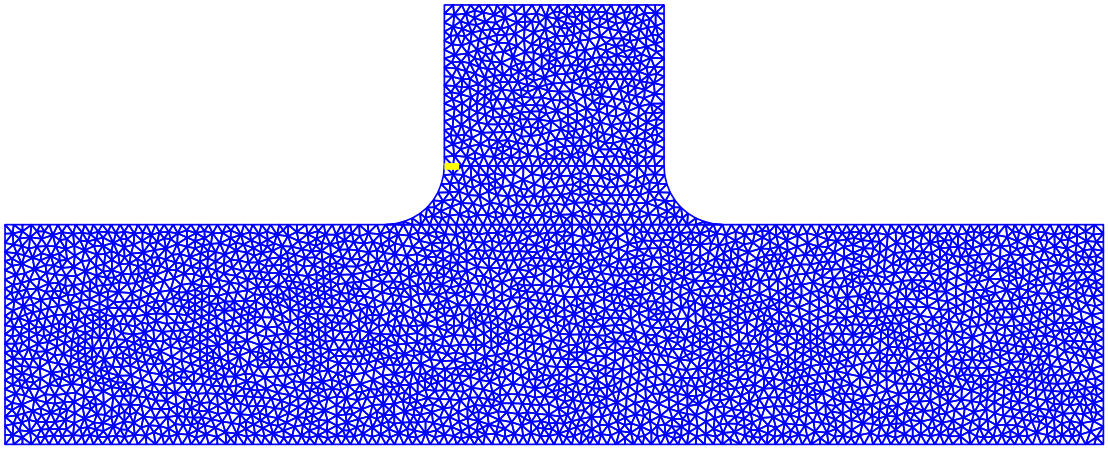


Figure 4.39: Fixed unstructured mesh used for the crack growth simulation

Crack growth is simulated for a total of 12 steps, with a constant crack increment length of $5mm$ for each step. Figure 4.39 shows the mesh in the vicinity of the fillet and compares the crack paths for the cases of a thick I-beam and (upper crack) and thin I-beam (lower crack). The results are consistent with both experimental (Sumi, Yang, and Wang, 1995) and previous numerical results using the EFG method (Fleming M. and Belytschko, 1997) and remeshing method (Bouchard, Bay, and Chastel, 2003) (see Figure 4.41).

The examples presented here are just a few of many applications which could be solved using the OpenXFEM++ code. Interested reader could refer to Nguyen

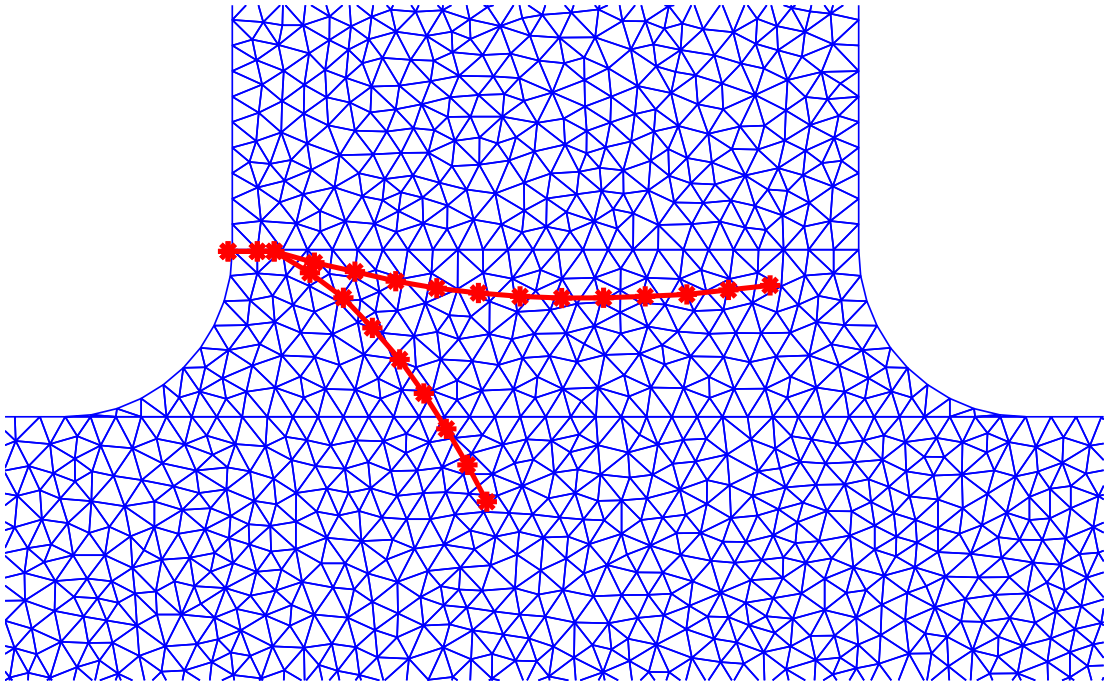
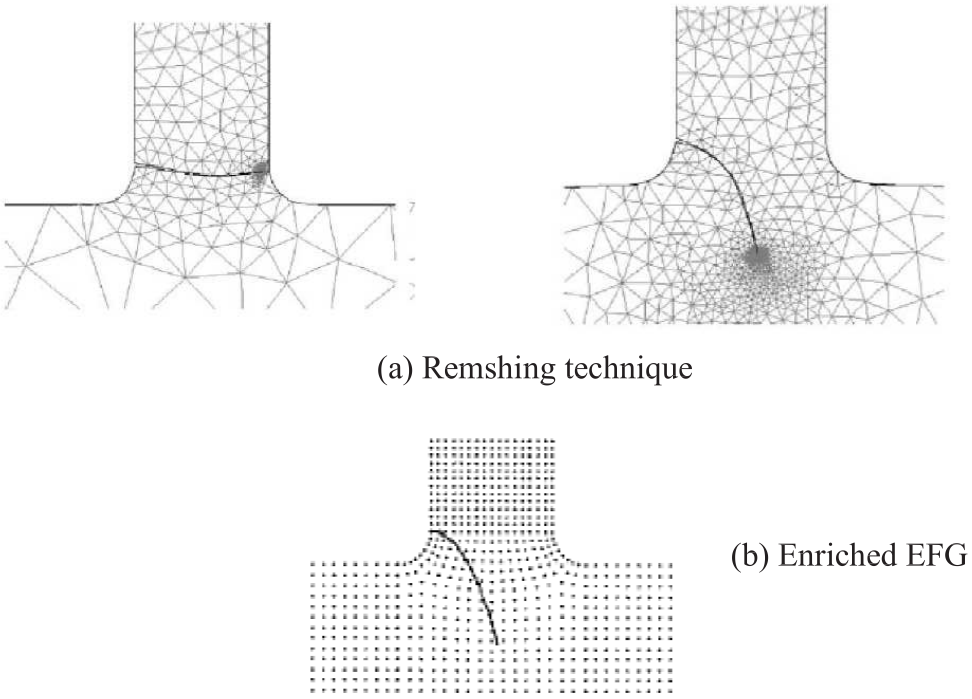


Figure 4.40: Zoom of crack paths for the case of a thick (top crack) and thin (bottom crack) I-beam.



(a) Remshing technique

(b) Enriched EFG

Figure 4.41: Comparison between various numerical methods

et al. (2005) for more interesting crack growth problems with full treatment on crack propagation modeling issues using the X-FEM.

4.3 Conclusions

This section presented the stress analysis of various benchmark fracture mechanics problems using the eXtended Finite Element Method. Accurate stress intensity factor computations (mode I and mixed-mode) were obtained for problems such as infinite cracked plate, edge cracked plate, center cracked plate and inclined cracked plate under uniaxial tension. Excellent domain independence in the SIF computations was realized for straight crack problems. For curved crack problems, a fine mesh, and an appropriate choice of domain size and crack representation leads to good SIFs. The convergence of the method was also verified. One can observe that, although good results were obtained with relatively coarse meshes, the rate of convergence cam be slow. This is due to the fact that, in our convergence studies, the enrichment radius was decreasing linearly with the mesh size.

The crack growth capabilities of the X-FEM were demonstrated through crack growth simulations in the edge cracked plate, angled center cracked plate, double cantilever beam specimen and crack growth from a fillet. Through these examples, the major advantage of the X-FEM has been pointed out: crack growth simulation without remeshing. To improve the capabilities of crack propagation simulation of the X-FEM, two things could be explored, namely (1) the crack could be represented smoothly, i.e., not by straight segments as in the present implementation; and (2) an appropriate path-independent integral for curved cracks should be used.

To smoothly approximate the crack, the level set method combined with high order X-FEM has been shown to be a very efficient method (Stazi et al., 2003). The crack growth law chosen to be implemented in the present code is the maximum hoop stress (Erdogan and Sih, 1963) simply because it is easy to implement. It would be interesting to compare with other criteria as in Bouchard, Bay, and Chastel 2003.

Also, the crack growth increment could valuably be made a dynamic variable/parameter of the problem. For instance, the local crack tip element length could be chosen. If the mesh were refined in the vicinity of the most critical areas of the structures, the computational cost would be decreased, and unnecessary growth steps avoided.

Moreover, it is obvious that coupling the X-FEM with a local remeshing technique based on some a-posteriori error estimate would allow for improved crack paths.

Chapter 5

Conclusions and future work

5.1 Summary on the completed work

In the context of the linear elastic fracture mechanics, the enriched finite element method, named eXtended Finite Element Method (X-FEM), was presented. The OpenXFEM++, an flexible object oriented C++ library for X-FEM, has been developed. The method has been successfully applied to static crack, quasi-static crack growth problems.

To model the crack in LEFM, the classical displacement-based finite element approximation is enriched with the discontinuous function and the linear elastic asymptotic near-tip fields. The cracks are represented independently of the underlying mesh which makes the pre-processing step easier and especially, no remeshing is required when cracks grow.

In order to build an extendable code in which new problem formulations can be easily added, the object oriented approach was chosen. The implementation of key classes such as **EnrichmentItem, EnrichmentFunction, GeometryEntity, IntegrationRule, EnrichmentDetector** was presented. In addition, the mod-

ifications to the classical finite element classes such as **Domain, Element, Node** were also stated. This thesis pointed out that, with the object oriented approach, new techniques in X-FEM, for instance, new numerical integration methods, can be incorporated without major obstacles.

The advantages and drawbacks of the code are given as follow. A major advantage of the code is that it allows problems with arbitrary enrichment items ; the addition of new enrichment functions is easy and straightforward. Another strength of the code is that one can use normal enrichment scheme or fixed enrichment area scheme just by adjusting the input file. Although the code just works for linear finite elements, high order X-FEM is not so difficult to implement since the six-noded triangle element has been implemented. Some limitations of the present implementation are : (1) the crack is represented by pure geometry not by level sets (note that for 2D cracks it is not necessary to use level set) ; (2) the code does not allow the cracks align with the element edge and (3) the detection of elements intersecting with cracks is not efficient for very fined mesh.

5.2 Possible lines of future work

This thesis could be considered as a start step in the long way to learn and apply the X-FEM to real applications. Much efforts need be done to reach this purpose, and the followings are some of issues necessary to research.

The method has been developed here in two dimensions, and a more challenging project would be to extend to three dimensions. Although this should not pose major obstacles, some issues still needed to be carefully investigated:

Geometry description of cracks For 3D cracks, it is obvious that the Level Set Method(LSM) should be used. The level sets are updated by solving the hyperbolic partial differential equations which made LSM not best suited to complex 3D cracks. Therefore, the vector level set method developed in (Ventura, Xu, and Belytschko, 2001) should be used since the update of the vector level set field involves only a few geometric equations.

Plane stress and/or plane strain In all published works on 3D cracks analysis using the X-FEM, either plane stress or plane strain condition is assumed for the entire crack front. This is not completely true since for crack front near the free surface, the condition is plane stress while for crack front in the solid body, the plane strain should be used.

A key advantage of the FEM over the Boundary Element Method (BEM) is the ability to model nonlinear material laws. The application of the X-FEM to ductile fracture is therefore an area of considerable promise. In (Legrain, Moës, and Verron, 2005), the X-FEM for large strain hyperelastic fracture mechanics was developed.

Numerical integration is indeed an ongoing research topic. The method used in this thesis requires the partition of elements interacted with the crack into sub-triangles, then in each sub-triangle a high order Gauss quadrature must be employed. This method, obviously , increases the computational cost a lot. The need of a better quadrature scheme is clear. Authors in (Béchet, Minnebo, Moës, and Burgardt, 2005) developed a very promise quadrature scheme for asymptotic functions. Stéphane Bordas proposed the transformation of the domain integral to contour integral based on divergence theorem. This is nice idea to be exploited.

The X-FEM with fixed enrichment area is proved to give better results. However, what the value of enrichment radius should be is still an opening question. To answer this question, an error estimator should be implemented. After getting the solution at the first step, the error is estimated, then the updated value of enrichment radius as well as the new mesh size are given (by the error estimator). The mesh is refined automatically and the new enrichment radius is used to detect enriched nodes. So, the better result is obtained.

Multiple cracks modeling is another interesting area where the X-FEM was successfully applied (Budyn, 2004). However this code was written in Matlab which makes further development very difficult. An object-oriented implementation for multiple cracks using X-FEM is really an interesting subject.

It has been shown that it is blending elements which made the rate of convergence of the X-FEM low. Therefore, incorporation of appropriate blending elements into the present code should be done in the near future.

Concerning the computer implementation, there are a lot of improvements need to be done, as cited below, to make OpenXFEM++ more efficient :

Better mesh database will do the mesh geometry interaction more efficient and quickly. The task of incorporating AOMD (the Algorithm Oriented Mesh Database) (Remacle et al., 2000) into the present code is essentially necessary.

Pre-processing The current way to build the input file is indeed a drawback of the code (just a temporary solution). Additional work should be done to make the pre-processing step better.

Post-processing Graphic functions of Matlab are great but plotting stress, dis-

placement field directly in the C++ code is, of course, more convenient. Open GL could be a choice.

Available useful C++ libraries useful libraries available on the web which should be used such as the Matrix Template Library (MTL), developed at Notre Dame university, which will improve the numerical computation on matrices.

References

Abdel-Rahman Ragab, S. E. B. (1999). *Engineering solid mechanics: Fundamentals and Applications*. CRC Press.

Babuška, I. and M. Rosenzweig (1972). A finite element scheme for domains with corners. *Numer. Math. 20*, 1—21.

Barsoum, R. S. (1977). Triangular quarter-point elements as elastic and perfectly-plastic crack tip elements. *International Journal for Numerical Methods in Engineering 11*, 85–98.

Béchet, E., H. Minnebo, N. Moës, and B. Burgardt (2005). Improved implementation and robustness study of the x-fem for stress analysis around cracks. *International Journal for Numerical Methods in Engineering*.

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

Bordas, S. (2001, November). An extended finite element method for elastic and elastic-plastic cracks in complex components.

Bordas, S. (2003, December). *Extended Finite Element and level set methods with applications to growth of cracks and biofilms*. Ph. D. thesis, Northwestern University.

Bouchard, P., F. Bay, and Y. Chastel (2003). Numerical crack modeling of crack propogation:automatic remeshing and comparison of different criteria. *Computer methods in applied mechanics and engineering*.

Breymann, U. (2002). *Designing Components with the C++ STL, a New Approach to Programming* (third ed.). Addison Wesley.

Budyn, E. (2004, June). *Multiple Crack Growth by the eXtended Finite Element Method*. Ph. D. thesis, Northwestern University.

Chen, H. and T. Belytschko (2003). An enriched finite element method for elastodynamic crack propagation. *International Journal for Numerical Methods in Engineering*. accepted for publication.

Chessa, J. (2002, December). *The Extended Finite Element Method for Free Surface and Two Phase Flow Problems*. Ph. D. thesis, Northwestern University.

Chessa, J. and T. Belytschko (2003a). An enriched finite element method for axisymmetric two-phase flow with surface tension. *Journal of Computational Physics*. submitted.

Chessa, J. and T. Belytschko (2003b). The extended finite element method for two-phase fluids. *ASME Journal of Applied Mechanics 70*(1), 10–17.

Chessa, J., P. Smolinski, and T. Belytschko (2002). The extended finite element method (x-fem) for solidification problems. *International Journal of Numerical Methods in Engineering 53*, 1957–1977.

Chessa, J., H. Wang, and T. Belytschko (2003). On the construction of blending elements for local partition of unity enriched finite elements. *International Journal of Numerical Methods in Engineering 57*, 1015–1038.

Chopp, D. L. and N. Sukumar (2003). Fatigue crack propagation of multiple coplanar cracks with the coupled extended finite element/fast marching method. *International Journal of Engineering Science 41*(8), 845–869.

Dolbow, J. E. (1999, July). *An Extended Finite Element Method with Discontinuous Enrichment for Applied Mechanics*. Ph. D. thesis, Northwestern University.

Dolbow, J. E., E. Fried, and H. Ji (2003). Chemically induced swelling of hydrogels. *Mechanics and Physics of Solids*. in press.

Duarte, C. A. and J. Oden (1996). An h-p adaptive method using clouds. *Comp Meth Appl Mech Eng 139*, 237—262.

Dunant, C., S. Bordas, P. Nguyen, A. Guidoum, and H. Nguyen-Dang (2005). Architecture trade-offs of including a mesher in an object-oriented extended finite element code. *Computational Mechanics*. in preparation.

Erdogan, F. and G. Sih (1963). On the crack extension in plates under plane loading and transverse shear. *Journal of Basic Engineering 85*, 519–527.

Fleming M., Y.A. Chu, B. M. and T. Belytschko (1997). Enriched element free galerkin methods for singular fields. *IJNME 40*, 1483—1504.

Forde BWR, Foschi RO, S. S. (1990). Object-oriented finite element analysis. *Computers and Structures 6*, 1—15.

Forth, S. and W. Keat (1996). Three-dimensional nonplanar fracture model using the surface integral method. *International Journal of Fracture 77*, 243–262.

Gdoutos, E. (1979). *Fracture mechanics*. Boston:Kluver Academics Publisher.

Gravouil, A., N. Moës, and T. Belytschko (2002). Non-planar 3d crack growth by the extended finite element and level sets. part II: level set update. *International Journal for Numerical Methods in Engineering 53*, 2569–2586.

Grisvard, P. (1985). *Elliptic Problems in Nonsmooth Domains*. Boston: Pitman Publishing, Inc.

Ji, H., D. Chopp, and J. E. Dolbow (2002). A hybrid extended finite element / level set method for modeling phase transformations. *International Journal for Numerical Methods in Engineering 54*(8), 1209–1233.

Kanninen, M. and C. Popelar (1985). *Advanced fracture mechanics*, Volume Oxford Engineering Science Series. Oxford University Press.

Laborde, P., J. Pommier, Y. Renard, and M. Salaun (2004). High order extended finite element method for cracked domains. *International Journal for Numerical Methods in Engineering 190*(47), 6183—6200.

Legrain, G., N. Moës, and E. Verron (2005). Stress analysis around the crack tips in finite strain problems using the extended finite element method. *International Journal for Numerical Methods in Engineering 63*, 290—314.

M. Lorentzon, K. E. (2000). A path independent integral for the crack extension force of the circular arc crack. *International Journal of Fracture 66*, 423—439.

Mackerle, J. (2000). Object-oriented techniques in fem and bem, a bibliography (1996-1999). *Finite Elements in Analysis and Design 36*, 189—196.

Melenk, J. M. and I. Babuška (1996). The partition of unity finite element method: Basic theory and applications. *Computer Methods in Applied Mechanics and Engineering 139*, 289–314.

Merle, R. and J. Dolbow (2002). Solving thermal and phase change problems with the extended finite element method. *Computational Mechanics 28*(5), 339–350.

Moës, N., J. Dolbow, and T. Belytschko (1999). A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering 46*, 131–150.

Moës, N., A. Gravouil, and T. Belytschko (2002). Non-planar 3d crack growth by the extended finite element and level sets. part I: Mechanical model. *International Journal for Numerical Methods in Engineering 53*, 2549–2568.

Moran, B. and C. F. Shih (1987). Crack tip and associated domain integrals from momentum and energy balance. *Engineering Fracture Mechanics 127*, 615–642.

Nguyen, P., S. Bordas, C. Dunant, H. Nguyen-Dang, and G. A. (2005). An object-oriented extended finite element library, part ii. numerical applications in fracture mechanics. *IJNME*. submitted.

Nikishkov, G. P. and S. N. Atluri (1987). Calculation of fracture mechanics parameters for an arbitrary three-dimensional crack by the 'equivalent domain integral' method. *International Journ Numer Meth. Engng 24*, 851—867.

Nuismer, R. (1975). An energy release rate criterion for mixed mode fracture. *International Journal of Fracture 11*, 245–250.

Osher, S. and J. A. Sethian (1988, November). Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics 79*(1), 12–49.

Paris, P. and F. Erdogan (December 1963). A critical analysis of crack propagation laws. *Journal of Basic Engineering, Transactions of the American Society of Mechanical Engineers*, 528–534.

Remacle, J.-F. and C. Geuzaine (1998). Gmsh finite element mesh generator. Available at http://www.montefiore.ulg.ac.be/geuzaine/gmsh.html.

Remacle, J.-F., B. K. Karamete, and M. Shephard (2000, October). Algorithm oriented mesh database. In *Ninth International Meshing Roundtable, Newport Beach,California*, pp. 349—359.

Rice, J. R. (1968). A path independent integral and the approximate analysis of strain concentration by notches and cracks. *Journal of Applied Mechanics*, 379—386.

S. Bordas, R. Duddu, B. M. and D. Chopp (2005). Extended finite element method for biofilm growth. *International Journal for Numerical Methods in Engineering*. submitted.

Shih, C. F., B. Moran, and T. Nakamura (1986). Energy release rate along a three-dimensional crack front in a thermally stressed body. *International Journal of Fracture 30*, 79–102.

Sih, G. C. (1973). Energy-density concept in fracture mechanics. *Engineering Fracture Mechanics 5*, 1037–1040.

Sladek, J., V. Sladek, and S. N. Atluri (Jan 2000). Local boundary integral equation (lbie) method for solving problems of elasticity with nonhomogeneous material properties. *Computational Mechanics 24*(6), 456—462. .

Stazi, F., E. Budyn, J. Chessa, and T. Belytschko (2003). An extended finite element method with higher-order elements for crack problems with curvature. *International Journal for Numerical Methods in Engineering*. accepted for publication.

Stolarska, M., D. L. Chopp, N. Moës, and T. Belytschko (2001). Modelling crack growth by level sets and the extended finite element method. *International Journal for Numerical Methods in Engineering 51*(8), 943–960.

Stroustrup, B. (2002). *The C++ Programming Language* (third ed.). Addison Wesley.

Sukumar, N., D. Chopp, N. Moës, and T. Belytschko (2001). Modeling holes and inclusions by level sets in the extended finite element method. *International Journal for Numerical Methods in Engineering 190*(47), 6183—6200.

Sukumar, N., D. L. Chopp, and B. Moran (2003). Extended finite element method and fast marching method for three-dimensional fatigue crack propagation. *Engineering Fracture Mechanics 70*(1), 29—48.

Sukumar, N., Z. Y. Huang, J.-H. Prévost, and Z. Suo (June 2003). Partition of unity enrichment for bimaterial interface cracks. *International Journal for Numerical Methods in Engineering*. accepted for publication.

Sukumar, N., N. Moës, T. Belytschko, and B. Moran (2000). Extended Finite Element Method for three-dimensional crack modelling. *International Journal for Numerical Methods in Engineering 48*(11), 1549–1570.

Sukumar, N. and J.-H. Prévost (August 2003). Modeling quasi-static crack growth with the extended finite element method. part i: Computer implementation. *International Journal of Solids and Structures*. Accepted for publication.

Sumi, Y., C. Yang, and Z. Wang (1995). Morphological aspects of fatigue crack propagation. Part II - effects of stress biaxiality and welding residual stresses. Technical report, Department of Naval Architecture and Ocean Engineering, Yokohama National University, Japan.

Timoshenko and Goodier (1970). *Theory of Elasticity*. McGraw-Hill.

Ventura, G., J. X. Xu, and T. Belytschko (June, 2001). Level set crack propagation modelling in the element-free galerkin method. In *European Conference on Computational Mechanics*.

Zimmermann, T., Y. D. Pelerin, and P. Bomme (1992). Object oriented finite element programming: I. governing principles. *Comp. Meth. in Applied Mech. and Engrg. 93*(8), 291–303.

Appendix A

The class hierarchy

This appendix introduces the class hierarchy of the OpenXFEM++ library. Classes in **bold** font are new ones, while the classes in *italic* font are classes of FEMOBJ which are modified to include the XFEM.

AuxiliaryFields
Dictionary
Dof
Domain
FEInterpolation
    FEInterpolation2d
        FEI2dQuadLin
        FEI2dTriLin
IntegrationRule
    SplitGaussQuadrature
    StandardGaussQuadrature
FEMComponent
    CrackGrowthIncrementLaw
        ParisLaw
        FixedIncrement
    CrackGrowthDirectionLaw
        MaxHoopStress
        MaxEnergyReleaseRate

Element
- QuadU
- **TriU**
- **Tri6**

EnrichmentFunction
- **DiscontinuousFunction**
- **AsymptoticFunction**
  - **CrackAsymptotic**
  - **HomogElastCrackAsymp**
  - **BiMaterialElastCrackAsymp**

EnrichmentItem
- **CrackInterior**
- **CrackTip**
- **Hole**
- **MaterialInterface**

GeometryDescription
- **LevelSetDescription**
- **VectorLevelSetDescription**
- **StandardDescription**

GeometryEntity
- **Circle**
- **PiecewiseLinear**
- **Vertex**

Load
- BodyLoad
  - DeadWeight
- BoundaryCondition
- InitialCondition
- NodalLoad

LoadTimeFunction
- ConstantFunction
- PeakFunction
- PiecewiseLinFunction

Material
- ElasticMaterial
- VonMisesMaterial
- VonMisesMaterialH
- **NullMaterial**

NLSolver
- ConstantStiffness

Appendix B

The data file of OpenXFEM++ and some Matlab routines

This appendix presents the data file of the OpenXFEM++ package. After a short introduction of the format of this file, steps to build automatically this input file for a finite element problem is given. At the end of this appendix, the Matlab routines to be used are given.

B.1 The data file of OpenXFEM++

Below is the typical data file for a finite element problem with discontinuities (cracks).

```
TimeIntegrationScheme
1 class Static   *
**
```

```
Material 1
1 E 3.e7 n 0.25 *
**

Element 234
1 class T3U  mat 1 nodes 1 3 4     *
2 class T3U  mat 1 nodes 10 33 41  *
**

Node 1234
1 coord 2 0.0 0.0 nDofs 2 bcOnDof1 1  bcOnDof2 1   *
2 coord 2 2.0 0.0 nDofs 2 bcOnDof1 1  bcOnDof2 1   *
**

Load 2
1 class BoundaryCondition  loadTimeFunction  1  conditions 1  d  0.    *
2 class NodalLoad          loadTimeFunction  2  components 2 4000.  0. *

**
EnrichmentItem 3
1 class CrackInterior myTips 2 2 3 geometry 1
EnrichmentFunctions 1 1 enrichScheme 3*
2 class CrackTip Type HomoElast Mat 1 geometry 2 EnrichmentFuncs 4 2 3 4 5
enrichScheme 1  domainIntRadius 2.5*
3 class CrackTip TypeHomoElast  Mat 1 geometry 2 EnrichmentFuncs 4 2 3 4 5
enrichScheme 1  domainIntRadius 2.5*
**

GeometryEntity 3
1 class PiecewiseLinear numOfVertices 2 vertices 2 3 geoDescription 1*
2 class Vertex coord 2 0.75 3.0   *
3 class Vertex coord 2 1.25 3.0   *
**

EnrichmentFunction 5
1 class DiscontinuousField       *
2 class HomoElastCrackAsymp1     *
3 class HomoElastCrackAsymp2     *
4 class HomoElastCrackAsymp3     *
5 class HomoElastCrackAsymp4     *
**
```

```
CrackGrowthDirectionLaw
1 class MaxHoopStress *
**

CrackGrowthIncrementLaw
1 class FixedIncrement delta 0.2 *
**
```

For the complete discussion on the sections concerning FEM such as **Element, Node, Material, Load, LoadTimeFunction, TimeStep, TimeIntegrationScheme**, please refer to the documents of FEMOBJ which are available at www.zace.com. Here, only sections on XFEM are presented as follows.

B.1.1 Section EnrichmentItem

- Keyword line :
EnrichmentItem n

n : number of enrichment items

CrackInterior

- Data line :
n class CrackInterior myTips $tips$ tip_1 tip_2 geometry $geoID$ EnrichmentFunctions 1 $funcID$ enrichScheme $enrID$ *

n	:	EnrichmentItem number
$tips$	:	number of crack tips
tip_1	:	the first crack tip number
tip_2	:	the second crack tip number
$geoID$	:	geometry number
$funcID$	:	enrichment function number
$enrID$	:	enrichment detector number (see remark below)

- Example:
1 class CrackInterior myTips 2 2 3 geometry 1 EnrichmentFunctions 1 1 enrichScheme 3 * -Remark: The only value that can be assigned to *enrID* is 3, i.e., all nodes of any element split by the CrackInterior will be enriched by the Heaviside function.

CrackTip

- Data line :
n class CrackTip Type $tipType$ Mat mat geometry $geoID$ EnrichmentFunctions m

f_1 f_2 f_3 f_4 enrichScheme $enrID$ domainIntRadius r *

n	:	EnrichmentItem number
$tipType$	:	type of the crack tip
mat	:	material number
$geoID$	:	geometry number
m	:	total number of enrichment functions
f_i	:	the ith enrichment function number
$enrID$	:	enrichment detector number(see remark below)
r	:	radius of the integration domain used to compute SIFs

- Example:
3 class CrackTip Type HomoElast Mat 1 geometry 2 EnrichmentFuncs 4 2 3 4 5 enrichScheme 1 domainIntRadius 2.000 *
-Remark : We can assign 1 or 2 to *enrID*. If $enrID = 1$, all nodes of a given element containing the crack tip will be enriched by the asymptotic functions. If $enrID = 2$, any node belongs to the circle of radius r centered at the crack tip will be tip-enriched nodes. In this case, the data line of CrackTip becomes:
3 class CrackTip Type HomoElast Mat 1 geometry 2 EnrichmentFuncs 4 2 3 4 5 enrichScheme 3 enrichRadius 2.0 domainIntRadius 2.0 *

B.1.2 Section EnrichmentFunction

- Keyword line :
EnrichmentFunction n
n : number of enrichment functions
DiscontinuousField
- Data line :
n class DiscontinuousField *
HomoElastCrackAsymp1
- Data line :
n class HomoElastCrackAsymp1 *

B.1.3 Section GeometryEntity

- Keyword line :
GeometryEntity n
n : number of geometry entities

PiecewiseLinear

- Data line :
n class PiecewiseLinear numOfVertices m vertices v_1 v_2 geoDescription $geoDesID$ *

n	:	PiecewiseLinear number
m	:	number of vertices
v_i	:	the ith vertex number
$geoDesID$	:	the description of the PiecewiseLinear(see remark below)

- Example:
1 class PiecewiseLinear numOfVertices 2 vertices 2 3 geoDescription 1 *
-Remark: The possible values of *geoDesID* are 1 (standard description), 2 (level set description) and 3 (vector level set description). However, in current implementation, only standard description was coded.

Vertex

- Data line :
n class Vertex coord nsd x y *

n	:	Vertex number
nsd	:	number of space dimension
x	:	x coordinate of the Vertex
y	:	y coordinate of the Vertex

- Example:
2 class Vertex coord 2 0.75 3.0 *

B.1.4 Section CrackGrowthDirectionLaw

MaxHoopStress

It means that the crack growth angle is computed by the maximum hoop stress criterion.

B.1.5 Section CrackGrowthIncrementLaw

FixedIncrement
- Data line :
n class FixedIncrement delta Δ *

n : CrackGrowthIncrementLaw number
Δ : crack increment length

B.2 How to build the input data file

The procedure to have a input file of correct format required by the OpenXFEM++ is as follows

1. Generation of the finite element mesh
   To build the geometry model and then the mesh of problem at hand, the program Gmsh was used. The output file extension is *.msh*.

2. Read the above *.msh* file to get nodes and elements
   This is done by using the Matlab M file *msh2mlab.m*.

3. Finally, make the data file for OpenXFEM++ using the Matlab routine *MakeFemObjDataFile.m*.

B.3 Some Matlab routines used to get the data file

B.3.1 The Matlab M file msh2mlab.m

This useful routine is written by Jack Chessa at Northwestern University, USA.

```
function [node,element,elemType]=msh2mlab(meshFile)
% function [node,connectivities,elemType]=msh2mlab('meshFile')
% This file reads in a mesh file from gmsh and returns the appropirate input
% data structures for a matlab finite element code.
%
%   'meshFile' - is the file name of the *.msh file ( includeing *.msh )
%
%   node - is an nodal coordinate matrix ( Nx3, where N is the number of nodes )
%
```

```
%   connectivities - is a cell array of connetivitiy matricies for each
%                        zone defined in the *.msh ( or *.geo ) file
%
%   elemType - is a vector of strings that tell what type of element is in
%              each zone ( we assume that only one type of element is
%              defined for each zone )
%
% Example:
%       plate.msh is a mesh file with three zones. Zone 1 is the mesh of
%       the domain interior, zone 2 is a mesh of the traction boundary
%       and zone 6 is a mesh of the displacement boundary the code to
%       process this file would be
%
%           [node,conns,elemType]=msh2mlab('plate.msh');
%           domain=conns{1};
%           tracBndy=conns{2};
%           dispBndy=conns{6};
%           dispBndyNodes=unique(dispBndy);

% open the file
meshPath=''; %'/home/jack/Meshes/';
fid=fopen([meshPath,meshFile],'r');
% define mesh data structures
pts=[]; seg={}; zon={}; node=[]; element={};

%* README: The 'msh' file format is the native output file format for
%  Gmsh. The file is divided in several sections (enclosed in $KEY and
%  $ENDKEY pairs). Two fields are important: $NOD/$ENDNOD defines the
%  nodes and $ELM/$ENDELM defines the elements.
%
%  The syntax is as follows:
%
%  $NOD
%  number-of-nodes
%  node-number x-coord y-coord z-coord
%  ...
%  $ENDNOD
%
%  $ELM
%  number-of-elements
%  elm-number elm-type elm-region unused number-of-nodes node-numbers
%  ...
```

```
%  $ENDELM
%
%  All the syntactic variables stand for integers except x-coord,
%  y-coord and z-coord which stand for floating point values.  The
%  elm-type value defines the geometrical type for the element:
%
%  elm-type:
%
%  1 Line (2 nodes, 1 edge).
%  2 Triangle (3 nodes, 3 edges).
%  3 Quadrangle (4 nodes, 4 edges).
%  4 Tetrahedron (4 nodes, 6 edges, 4 facets).
%  5 Hexahedron (8 nodes, 12 edges, 6 facets).
%  6 Prism (6 nodes, 9 edges, 5 facets).
%  7 Pyramid (5 nodes, 8 edges, 5 facets).
%  15 Point (1 node).
%
%  The elm-region value is the number of the physical entity to which
%  the element belongs.
zoneID=0;
% read sections
while 1
  line=fgetl(fid);            % read line
  if ~isstr(line), break, end % check if EOF
  switch line                 % find seciton
  case '$PTS'
    n=str2num(fgetl(fid));
    pts=zeros(n,6);

    for i=1:n
      pts(i,:)=str2num(fgetl(fid));
    end

  case '$SEG'
    n=str2num(fgetl(fid));

    for i=1:n
      seg{i}=str2num(fgetl(fid));
    end

  case '$ZON'
    n=str2num(fgetl(fid));
```

```
  for i=1:n
    zon{i}=str2num(fgetl(fid));
  end

  zoneID=ones(length(zon),1);
  for i=1:length(zon);
    zoneID(i)=zon{i}(1);
    element{zoneID(i)}=[];
    elemType{i}='??';
  end

case '$NOD'
  numnode=str2num(fgetl(fid));
  node=zeros(numnode,3);

  for i=1:numnode
    nodeline=str2num(fgetl(fid));
    node(nodeline(1),:)=nodeline(2:4);
  end

case '$NOE'
  numnode=str2num(fgetl(fid));
  node=zeros(numnode,3);

  for i=1:numnode
    nodeline=str2num(fgetl(fid));
    node(nodeline(1),:)=nodeline(2:4);
  end

case '$ELM'
  n=str2num(fgetl(fid));

  for i=1:n
    temp=str2num(fgetl(fid));  % get element
    inZone=temp(3);            % find zone element is in
    z=find(ismember(zoneID,inZone));

    if ( isempty(z) ) % zone is not yet defined
      zoneID=[zoneID;inZone];
      z=size(zoneID,1);
      elemType{zoneID(z)}='??';
```

```
        element{zoneID(z)}=[];
      end

      if isempty(element{z}) % first element inZone so set type
        switch temp(2)
        case 1
          elemType{zoneID(z)}='L2';
        case 2
          elemType{zoneID(z)}='T3';
        case 3
          elemType{zoneID(z)}='Q4';
        case 4
          elemType{zoneID(z)}='H4';
        case 5
          elemType{zoneID(z)}='B8';
        case 6
          elemType{zoneID(z)}='P6';
        case 15
          elemType{zoneID(z)}='P1';
        otherwise
          elemType{zoneID(z)}='??';
        end
      end
      % add element to that zone
      element{zoneID(z)}=[element{zoneID(z)};temp(6:5+temp(5))];
    end
  otherwise
    % skip line
  end
end

fclose(fid);
```

B.3.2 The Matlab M file MakeFemObjDataFile.m

```
% ========================================================================
% The original version is written by Stephane Bordas
% Modified by Nguyen Vinh Phu 2005.05.26 for XFEM
% EMMC IX, Hochiminh University of Technology, Vietnam
% ========================================================================
```

```
% ************************************************************************
% ******                    GEOMETRY and MESH                          ******
% ************************************************************************
% For Gmsh, see Jean Francois Remacle et al.

% GMSH FILE PARPMETERS
gmshFileName = 'griffith.msh';        % name of gmsh input file
botBoundaryID   = 20;                 % displacement boundary
topBoundaryID   = 21;                 % traction boundary
interiorID = 22;                      % interiorID interior of the plate

% READ GMSH FILE

[node,elements,elemType] = msh2mlab(gmshFileName);
[node,elements] = remove_free_nodes(node,elements);

% GET NODE AND CONNECTIVITY MATRICES FOR INTERIOR
node     = node(:,1:2);         % a nodal coordinate matrix
element  = elements{interiorID};% connectivity matrix

% compute total number of nodes and elements
numelem = size(element,1);
numnode = size(node,1);

% GET NODE AND CONNECTIVITY MATRICES FOR BCs
dispEdge  = elements{botBoundaryID};   % connectivity matrix for bottom edge
topEdge   = elements{topBoundaryID };  % connectivity matrix for top edge

% GET NODES ON DIRICHLET BOUNDARY AND ESSENTIAL BOUNDARY
fixedNodes=unique(dispEdge);
loadedNodes=unique(topEdge);

% CHECK FOR BAD JACOBIANS
element=tricheck(node,element,1);

% Recognize element type automatically
switch size(element,2)
 case 2
  elt_type = 'L2'
 case 3
  elt_type = 'T3'
 case 6
```

```
  elt_type = 'T6'
 case 4
  elt_type = 'Q4'
 otherwise
  error('Unknown element type found in mesh');
end

% **************************************************************************
% *******                    MATERIAL CONSTANT                        *******
% **************************************************************************

E = 3.e7 ;      % Young modulus
nu = 0.25;      %  Poisson ratio

% BOUNDARY CONDITIONS
dispval = 0.0 ; % value of fixed displacement

% LOADING
sigma = 1.0 ;   % the far field stress acting on top edge, along Y direction
nodalLoadValue = sigma * Width ; % nodal load = sigma * width

% --------------------------------------------------------------------------
%     WRITE THE INPUT FILE follows format of FEMOBJ
%     For more info on FEMOBJ, see www.zace.com
% --------------------------------------------------------------------------
outFileName = 'D:\MASTERTHESIS\Inputfiles\HomoCracks\griffith.inp';
outfid = fopen(outFileName,'w');

% ----------------------------------
%               MATERIAL
% ----------------------------------
fprintf(outfid,'Material 1 \n');
fprintf(outfid,'1 class ElasticMaterial E %15.14f n %15.14f t 1.0 *\n',E,nu);
fprintf(outfid,'**\n\n');
% ----------------------------------------
%           TIME INTEGRATION
% ----------------------------------------
fprintf(outfid,'TimeIntegrationScheme *\n');
fprintf(outfid,'1 class Static *\n');
fprintf(outfid,'**\n\n');

% --------------------------------------------------------------------------
```

```
%   LOAD SECTION (includes Dirichlet conditions)
%   Example :
%   Load 2
%   1 class BoundaryCondition loadTimeFunction  1  conditions 1  d  0.   *
%   2 class NodalLoad         loadTimeFunction  2  components 2 4000.  0.*
%   **
% ------------------------------------------------------------------------

fprintf(outfid,'Load %d \n',2);
fprintf(outfid,'%d class BoundaryCondition loadTimeFunction 1 conditions 1
d %15.14f *\n',1,dispval);
fprintf(outfid,'%d class NodalLoad loadTimeFunction 2 components 2
%15.14f %15.14f  * \n',2,0,nodalLoadValue);
fprintf(outfid,'**\n\n');

% ------------------------------------------------------------------------
%           LOADTIME FUNCTIONS
% ------------------------------------------------------------------------

fprintf(outfid,'LoadTimeFunction 1 \n');
fprintf(outfid,'1 class ConstantFunction f(t) %f *\n',1);
fprintf(outfid,'**\n\n');

% ------------------------------------------------------------------------
%                                  NODE SECTION
%   Example :
%   Node 1234
%   1  coord 2 0.0 0.0 nDofs 2 bcOnDof1 1 loads 1 *
%   2  ...
% ------------------------------------------------------------------------
fprintf(outfid,'Node %d \n',numnode);
nodei = 0;
for i=1:numnode
  nodei = nodei +1 ;
  fprintf(outfid,['%d nDofs 2 coord 2 %15.14f %15.14f'],nodei,node(i,1),node(i,2));
  if (ismember(i,fixedNodes))
      fprintf(outfid,' bcOnDof1 %d',1);    % two Dofs are fixed
      fprintf(outfid,' bcOnDof2 %d',1);
  end
  if (ismember(i,loadedNodes))
      fprintf(outfid,' loads 1 2 %d');     % supported nodal loads
  end
```

```
  fprintf(outfid,' *');
  fprintf(outfid,' \n');
end fprintf(outfid,'**\n\n');
% ------------------------------------------------------------------------
%                       ELEMENT SECTION
%    Example :
%    Element 2356
%    1  class T3U   mat   1   nodes  1 3 56    *
%    2  class T3U   mat   1   nodes  12 34 56  *
%    ...
% ------------------------------------------------------------------------

fprintf(outfid,'Element %d \n',numelem);
for i=1:numelem
    nume = nume+1;
    fprintf(outfid,'%d class T3U nodes %d %d %d  mat 1 ',nume,...
        element(i,1),element(i,2),element(i,3));
    fprintf(outfid,' *');
    fprintf(outfid,' \n');
end
fprintf(outfid,'**\n\n');

% ---------------------------------------------------------------------
%                       ENRICHMENT ITEM SECTION
%    Example : for one Griffith crack,i.e.,one CrackInterior and two CrackTip
%    EnrichmentItem 3
%    1 class CrackInterior geometry 1 EnrichmentFunctions 1 1 enrichScheme 3*
%    2 class CrackTip Type HomoElast Mat 1 geometry 2
%          EnrichmentFuncs  4 2 3 4 5  enrichScheme 1 enrichRadius 2.5*
%    3 class CrackTip Type HomoElast Mat 1 geometry 3
%          EnrichmentFuncs  4 2 3 4 5  enrichScheme 1 enrichRadius 2.5*
%    **
% ---------------------------------------------------------------------
numOfEnrItem = 3;
enrichScheme = 1 ; %  standard enrichment => no need enrichRadius
enrichRadius = 3 ; % radius of enrichment
domainIntRadius = 2 ;

fprintf(outfid,'EnrichmentItem %d \n',numOfEnrItem);

fprintf(outfid,'%d class CrackInterior  geometry 1 EnrichmentFunctions 1 1
enrichScheme 3',1);
```

```
fprintf(outfid,' *'); fprintf(outfid,' \n');

if (enrichScheme == 1)
   fprintf(outfid,'%d class CrackTip Type HomoElast Mat 1 geometry 2
   EnrichmentFuncs 4  2 3 4 5 enrichScheme %d domainIntRadius %5.3f',2,
   enrichScheme,domainIntRadius);
   fprintf(outfid,' *'); fprintf(outfid,' \n');
   fprintf(outfid,'%d class CrackTip Type HomoElast Mat 1 geometry 3
   EnrichmentFuncs 4  2 3 4 5 enrichScheme %d domainIntRadius %5.3f',3,
   enrichScheme,domainIntRadius);
   fprintf(outfid,' *'); fprintf(outfid,' \n');
else
    fprintf(outfid,'%d class CrackTip  Type HomoElast  Mat 1  geometry 2
    EnrichmentFuncs  4   2  3  4  5 enrichScheme %d enrichRadius %5.3f
    domainIntRadius %5.3f ',3,enrichScheme,enrichRadius,domainIntRadius);
    fprintf(outfid,' *'); fprintf(outfid,' \n');
    fprintf(outfid,'%d class CrackTip  Type HomoElast  Mat 1  geometry 3
    EnrichmentFuncs  4   2  3  4  5 enrichScheme %d enrichRadius %5.3f
    domainIntRadius %5.3f ',3, enrichScheme,enrichRadius,domainIntRadius);
    fprintf(outfid,' *'); fprintf(outfid,' \n');
end
fprintf(outfid,'**\n\n');
% ------------------------------------------------------------------
%                       GEOMETRY ENTITY SECTION
%    Example :
%
%     GeometryEntity 3
%     1 class PiecewiseLinear numOfVertices 2 vertices 2 3 geoDescription 1 *
%     2 class Vertex coord 0.75 3.0                    *
%     3 class Vertex coord 1.25 3.0                    *
%     **
% ------------------------------------------------------------------
numOfGeoEntity = 3;
fprintf(outfid,'GeometryEntity %d \n',numOfEnrItem);

fprintf(outfid,'%d class PiecewiseLinear numOfVertices 2 2 3 ...
geoDescription 1',1);
fprintf(outfid,' *');fprintf(outfid,' \n');
fprintf(outfid,'%d class Vertex  coord 2 0.75 3.0 geoDescription 1',2);
fprintf(outfid,' *'); fprintf(outfid,' \n');
fprintf(outfid,'%d class Vertex  coord 2 1.25 3.0 geoDescription 1',3);
fprintf(outfid,' *'); fprintf(outfid,' \n');
```

```
fprintf(outfid,'**\n\n');
% ------------------------------------------------------------------------
%                       ENRICHMENT FUNCTION SECTION
%    Example :   enrichment functions for cracks
%              (Heavide function + 4 asymp functions)
%
%    EnrichmentFunction 5
%    1 class DiscontinuousField       *
%    2 class HomoElastCrackAsymp1     *
%    3 class HomoElastCrackAsymp2     *
%    4 class HomoElastCrackAsymp3     *
%    5 clas  HomoElastCrackAsymp4     *
%    **
% ------------------------------------------------------------------------
fprintf(outfid,'EnrichmentFunction %d \n',numOfEnrFunc);

fprintf(outfid,'%d class DiscontinuousField ',1);
fprintf(outfid,'*');fprintf(outfid,'\n');
for i = 1:4
    fprintf(outfid,['%d class HomoElastCrackAsymp' num2str(i) ],1+i);
    fprintf(outfid,' *'); fprintf(outfid,' \n');
end fprintf(outfid,'**\n\n');
% ----------------------------------
%              NLSOLVER
% ----------------------------------
fprintf(outfid,'NLSolver *\n'); fprintf(outfid,'1 class
NewtonRaphson n 100 t 1e-5 c 1 *\n'); fprintf(outfid,'**\n\n');
% ----------------------------------
%              TIME STEP
% ----------------------------------
fprintf(outfid,'TimeStep 1 \n'); fprintf(outfid,'1 dt 1.0  *\n');
fprintf(outfid,'**\n\n');
% ----------------------------------
%            CLOSE FILE
% ----------------------------------
fclose(outfid);
```

It is obvious that the way to get the data file is just semi-automatic. It is a good exercise on C++ to implement these routines directly in OpenXFEM++.

Appendix C

Some details on the X-FEM

C.1 Derivation of the discretized equations

Linear space, Y A set Y is a linear (or vector) space if

$$\forall v_1, v_2 \in Y, v_1 + v_2 \in Y$$

$$\forall \alpha \in R, \forall v \in Y, \alpha v \in Y$$

Linear forms, L(v)

$$L : \underbrace{Y}_{\text{input}} \rightarrow \underbrace{R}_{\text{output}}$$

$$L(\alpha v_1 + v_2) = \alpha L(v_1) + L(v_2)$$

$$\forall \alpha \in R, \quad \forall v_1, v_2 \in Y$$

Bilinear forms, B(u,v) is a bilinear form or operator if

B(u,$\overline{v}$) is a linear form in u for fixed $\overline{v}$,

B($\overline{u}$,v) is a linear form in v for fixed $\overline{u}$

B(u,v) is a symmetric bilinear form if B(u,v) = B(v,u).

The weak formulation of the equilibrium equation is give by (2.2.2)

$$\text{Find } \mathbf{u} \in U \quad | \quad \forall \mathbf{v} \in \mathcal{U}_0,\ B(\mathbf{u}, \mathbf{v}) = L(\mathbf{v}) \tag{C.1}$$

The enriched finite element approximation is reintroduced (2.40)

$$\mathbf{u}^h(\mathbf{x}) = \sum_{I, n_I \in \mathcal{N}_e} N_I(\mathbf{x}) \mathbf{u}_I + \sum_{J, n_J \in \mathcal{N}^{enr}} N_J(\mathbf{x}) \Phi(\mathbf{x}) \mathbf{a}_J \tag{C.2}$$

The equation (2.36) is equivalent to find $(\mathbf{u}_I, \mathbf{a}_J) : \forall (\mathbf{v}_K, \mathbf{b}_L)$, the following equation is satisfied

$$\begin{aligned} B\Big(\sum_{\substack{I \\ n_I \in N_e}} N_I \mathbf{u}_I + \sum_{\substack{J \\ n_J \in N^{enr}}} N_J \Phi \mathbf{a}_J,\ \sum_{\substack{K \\ n_K \in N_e}} N_K \mathbf{v}_K + \sum_{\substack{L \\ n_L \in N^{enr}}} N_L \Phi \mathbf{b}_L \Big) = \\ = L\Big(\sum_{\substack{K \\ n_K \in N_e}} N_K \mathbf{v}_K + \sum_{\substack{L \\ n_L \in N^{enr}}} N_L \Phi \mathbf{b}_L \Big) \end{aligned} \tag{C.3}$$

Due to the linearity of L, we have

$$L\Big(\sum_{\substack{K \\ n_K \in N_e}} N_K \mathbf{v}_K + \sum_{\substack{L \\ n_L \in N^{enr}}} N_L \Phi \mathbf{b}_L \Big) = \sum_{\substack{K \\ n_K \in N_e}} \mathbf{v}_K L(N_K) + \sum_{\substack{L \\ n_L \in N_{enr}}} \mathbf{b}_L L(N_L \Phi) \tag{C.4}$$

Since B(u,v) is a bilinear form

$$
\begin{aligned}
B(u^h, v^h) &= B\Bigg(\sum_{\substack{I \\ n_I \in N_e}} N_I \mathbf{u}_I + \sum_{\substack{J \\ n_J \in N^{enr}}} N_J \Phi \mathbf{a}_J, \sum_{\substack{K \\ n_K \in N_e}} N_K \mathbf{v}_K + \sum_{\substack{L \\ n_L \in N^{enr}}} N_L \Phi \mathbf{b}_L \Bigg) \\
&= B\Bigg(\sum_{\substack{I \\ n_I \in N_e}} N_I \mathbf{u}_I, \sum_{\substack{K \\ n_K \in N_e}} N_K \mathbf{v}_K + \sum_{\substack{L \\ n_L \in N^{enr}}} N_L \Phi \mathbf{b}_L \Bigg) + \\
&+ B\Bigg(\sum_{\substack{J \\ n_J \in N^{enr}}} N_J \Phi \mathbf{a}_J, \sum_{\substack{K \\ n_K \in N_e}} N_K \mathbf{v}_K + \sum_{\substack{L \\ n_L \in N^{enr}}} N_L \Phi \mathbf{b}_L \Bigg) \\
&= B\Bigg(\sum_{\substack{I \\ n_I \in N_e}} N_I \mathbf{u}_I, \sum_{\substack{K \\ n_K \in N_e}} N_K \mathbf{v}_K \Bigg) + B\Bigg(\sum_{\substack{I \\ n_I \in N_e}} N_I \mathbf{u}_I, \sum_{\substack{L \\ n_L \in N^{enr}}} N_L \Phi \mathbf{b}_L \Bigg) + \\
&+ B\Bigg(\sum_{\substack{J \\ n_J \in N^{enr}}} N_J \Phi \mathbf{a}_J, \sum_{\substack{K \\ n_K \in N_e}} N_K \mathbf{v}_K \Bigg) + B\Bigg(\sum_{\substack{J \\ n_J \in N^{enr}}} N_J \Phi \mathbf{a}_J, \sum_{\substack{L \\ n_L \in N^{enr}}} N_L \Phi \mathbf{b}_L \Bigg) \\
&= \sum_{\substack{I \\ n_I \in N_e}} \sum_{\substack{K \\ n_K \in N_e}} \mathbf{u}_I \mathbf{v}_K B(N_I, N_K) + \sum_{\substack{I \\ n_I \in N_e}} \sum_{\substack{L \\ n_L \in N_{enr}}} \mathbf{u}_I \mathbf{b}_L B(N_I, N_L \Phi) + \\
&+ \sum_{\substack{J \\ n_J \in N_{enr}}} \sum_{\substack{K \\ n_K \in N_e}} \mathbf{a}_J \mathbf{v}_K B(N_J, N_K \Phi) + \sum_{\substack{J \\ n_I \in N_{enr}}} \sum_{\substack{L \\ n_K \in N_{enr}}} \mathbf{a}_J \mathbf{b}_L B(N_J \Phi, N_L \Phi)
\end{aligned}
\tag{C.5}
$$

Defining the following matrices

$$K^{uu}_{IK} = B(N_I, N_K) \tag{C.6a}$$

$$K^{ue}_{IL} = B(N_I, N_K \Phi) \tag{C.6b}$$

$$K^{eu}_{JK} = B(N_J, N_K \Phi) \tag{C.6c}$$

$$K^{ee}_{JL} = B(N_J \Phi, N_J \Phi) \tag{C.6d}$$

$B(u^h, v^h)$ can be rewritten

$$\sum_{\substack{I \\ n_I \in N_e}} \sum_{\substack{K \\ n_K \in N_e}} K^{uu}_{IK} u_I v_K + \sum_{\substack{I \\ n_I \in N_e}} \sum_{\substack{L \\ n_L \in N_{enr}}} K^{ue}_{JK} u_I b_L + \sum_{\substack{J \\ n_J \in N_e}} \sum_{\substack{L \\ n_L \in N_{enr}}} K^{eu}_{JK} a_J v_K + \sum_{\substack{J \\ n_J \in N_{enr}}} \sum_{\substack{L \\ n_L \in N_{enr}}} K^{ee}_{JL} a_J b_L \tag{C.7}$$

Also having

$$B(u^h, v^h) = \sum_{\substack{K \\ n_K \in N_e}} \mathbf{v}_K L(N_K) + \sum_{\substack{L \\ n_L \in N_{enr}}} \mathbf{b}_L L(N_L \Phi) \tag{C.8}$$

$$\sum_{\substack{I \\ n_I \in N_e}} K^{uu}_{IK} u_I + \sum_{\substack{J \\ n_J \in N_{enr}}} K^{eu}_{JK} a_J + \sum_{\substack{I \\ n_I \in N_e}} K^{eu}_{LI} u_I + \sum_{\substack{J \\ n_J \in N_{enr}}} K^{ee}_{JL} a_J = L(N_K) + L(N_L \Phi) \tag{C.9}$$

Under matrix form

$$\mathbf{K} \cdot \mathbf{u} = \mathbf{f}^{ext} \Leftrightarrow \begin{bmatrix} \mathrm{K}^{\mathrm{uu}} & \mathrm{K}^{\mathrm{ua}} \\ \mathrm{K}^{\mathrm{au}} & \mathrm{K}^{\mathrm{aa}} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{a} \end{bmatrix} = \begin{bmatrix} \mathbf{f}^{ext}_{\mathrm{u}} \\ \mathbf{f}^{ext}_{\mathrm{a}} \end{bmatrix} \tag{C.10}$$

where

$$\mathbf{f}^{ext}_{\mathrm{u}} = L(N_I) = \int_{\Gamma_t} N_I \bar{\mathbf{t}} \mathrm{d}\Gamma + \int_{\Omega} N_I \mathbf{b} \mathrm{d}\Omega \tag{C.11}$$

$$\mathbf{f}^{ext}_{\mathrm{a}} = L(N_J \Phi) = \int_{\Gamma_t} N_J \Phi \bar{\mathbf{t}} \mathrm{d}\Gamma + \int_{\Omega} N_J \Phi \mathbf{b} \mathrm{d}\Omega \tag{C.12}$$

C.2 Derivatives of near tip enrichment functions

To compute the stiffness matrices for elements enriched by the near tip asymptotic functions Φ_α, the following expression is about to determined

$$(N_i\Phi_\alpha)_{,x} = (N_i)_{,x}\Phi_\alpha + N_i(\Phi_\alpha)_{,x} \tag{C.13a}$$

$$(N_i\Phi_\alpha)_{,y} = (N_i)_{,y}\Phi_\alpha + N_i(\Phi_\alpha)_{,y} \tag{C.13b}$$

where $\Phi_{\alpha,x}$ and $\Phi_{\alpha,y}$ are the derivatives of Φ_α with respect to the global Cartesian coordinate system. These derivatives are, at first, found in the local crack tip coordinate system (x_1, x_2) and a vector transformation is used to obtain them with respect to the global Cartesian coordinate system (x, y).

The derivatives of Φ_α with respect to (w.r.t) the local crack tip coordinate system (x_1, x_2) are given

$$(\Phi_\alpha)_{,x_1} = (\Phi_\alpha)_{,r}r_{,x_1} + (\Phi_\alpha)_{,\theta}\theta_{,x_1} \tag{C.14a}$$

$$(\Phi_\alpha)_{,x_2} = (\Phi_\alpha)_{,r}r_{,x_2} + (\Phi_\alpha)_{,\theta}\theta_{,x_2} \tag{C.14b}$$

The near tip enrichment functions are recalled for ease of reading

$$\Phi_\alpha(r,\theta) = \left\{ \sqrt{r}\sin\frac{\theta}{2}, \sqrt{r}\cos\frac{\theta}{2}, \sqrt{r}\sin\frac{\theta}{2}\sin\theta, \sqrt{r}\cos\frac{\theta}{2}\sin\theta \right\} \tag{C.15}$$

The derivatives of Φ_α with respect to the polar coordinate (r, θ)

$$(\Phi_1)_{,r} = \frac{1}{2\sqrt{r}}\sin\frac{\theta}{2}, \quad (\Phi_1)_{,\theta} = \frac{\sqrt{r}}{2}\cos\frac{\theta}{2} \tag{C.16a}$$

$$(\Phi_2)_{,r} = \frac{1}{2\sqrt{r}}\cos\frac{\theta}{2}, \quad (\Phi_2)_{,\theta} = -\frac{\sqrt{r}}{2}\sin\frac{\theta}{2} \tag{C.16b}$$

$$(\Phi_3)_{,r} = \frac{1}{2\sqrt{r}}\sin\frac{\theta}{2}\sin\theta, \quad (\Phi_3)_{,\theta} = \sqrt{r}(\frac{1}{2}\cos\frac{\theta}{2}\sin\theta + \sin\frac{\theta}{2}\cos\theta) \tag{C.16c}$$

$$(\Phi_4)_{,r} = \frac{1}{2\sqrt{r}}\cos\frac{\theta}{2}\sin\theta, \quad (\Phi_4)_{,\theta} = \sqrt{r}(-\frac{1}{2}\sin\frac{\theta}{2}\sin\theta + \cos\frac{\theta}{2}\cos\theta) \tag{C.16d}$$

The derivatives of r, θ with respect to (x_1, x_2) are as follows

$$r_{,x_1} = \cos(\theta), \quad r_{,x_2} = \sin\theta \tag{C.17a}$$

$$\theta_{,x_1} = -\sin\theta/r, \quad \theta_{,x_2} = \cos\theta/r \tag{C.17b}$$

Finally, we have the derivatives of Φ_α with respect to the local crack tip system

$$(\Phi_1)_{,x_1} = -\frac{1}{2\sqrt{r}}\sin\frac{\theta}{2} \tag{C.18a}$$

$$(\Phi_1)_{,x_2} = \frac{1}{2\sqrt{r}}\cos\frac{\theta}{2} \tag{C.18b}$$

$$(\Phi_2)_{,x_1} = \frac{1}{2\sqrt{r}}\cos\frac{\theta}{2} \tag{C.19a}$$

$$(\Phi_2)_{,x_2} = \frac{1}{2\sqrt{r}}\sin\frac{\theta}{2} \tag{C.19b}$$

$$(\Phi_3)_{,x_1} = -\frac{1}{2\sqrt{r}}\sin\frac{3\theta}{2}\sin\theta \tag{C.20a}$$

$$(\Phi_3)_{,x_2} = \frac{1}{2\sqrt{r}}(\sin\frac{\theta}{2} + \sin\frac{3\theta}{2}\cos\theta) \tag{C.20b}$$

$$(\Phi_4)_{,x_1} = -\frac{1}{2\sqrt{r}}\cos\frac{3\theta}{2}\sin\theta \tag{C.21a}$$

$$(\Phi_4)_{,x_2} = \frac{1}{2\sqrt{r}}(\cos\frac{\theta}{2} + \cos\frac{3\theta}{2}\cos\theta) \tag{C.21b}$$

Using a vector transformation, the derivatives of near tip enrichment functions with respect to the global coordinate system are given by

$$(\Phi_\alpha)_{,x} = (\Phi_\alpha)_{,x_1}\cos(\alpha) - (\Phi_\alpha)_{,x_2}\sin(\alpha) \tag{C.22a}$$

$$(\Phi_\alpha)_{,y} = (\Phi_\alpha)_{,x_1}\sin(\alpha) + (\Phi_\alpha)_{,x_2}\cos(\alpha) \tag{C.22b}$$

where α is the inclination angle of the crack w.r.t the x axe of the global coordinate system (see Figure C.1).

C.3 Stress intensity factors computation using the interaction integral

This section presents, in great details, the computation of the stress intensity factors using the domain form of the interaction integral.
The interaction integral for states 1 and 2 is recalled for convenience

$$I^{(1,2)} = \frac{2}{E^*}(K_I^{(1)}K_I^{(2)} + K_{II}^{(1)}K_{II}^{(2)}) \tag{C.23}$$

To compute K_I, we choose state 2 as the pure Mode I ($K_{II}^{(2)} = 0$) with $K_I^{(2)} = 1$,

hence :

$$K_I = \frac{E^*}{2} I^{(1,ModeI)} \tag{C.24}$$

The domain form of the interaction integral is recalled (equation (2.26))

$$I^{(1,2)} = \int_\Gamma \left[\sigma_{ij}^{(1)} \frac{\partial u_i^{(2)}}{\partial x_1} - \sigma_{ij}^{(2)} \frac{\partial u_i^{(1)}}{\partial x_1} - W^{(1,2)} \delta_{1j} \right] \frac{\partial q}{\partial x_j} \mathrm{d}A \tag{C.25}$$

where $W^{(1,2)}$ is the interaction strain energy

$$W^{(1,2)} = \sigma_{ij}^{(1)} \epsilon_{ij}^{(2)} \tag{C.26}$$

After the solution of the boundary value problem, we obtained the stress and displacement fields of state 1, i.e., $\sigma_{ij}^{(1)}$, $u_i^{(1)}$ and the spatial derivatives of the displacement field in the global Cartesian coordinate system $u_{i,x}^{(1)}$, and $u_{i,y}^{(1)}$. These terms need to be transformed to the local crack tip coordinate system by using an appropriate vector transformation.

The derivatives of the displacement field with respect to x_1, x_2 is given as

$$\begin{pmatrix} u_{1,x_1}^{(1)} & u_{1,x_2}^{(1)} \\ u_{2,x_1}^{(1)} & u_{2,x_2}^{(1)} \end{pmatrix} = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} u_{1,x}^{(1)} & u_{1,y}^{(1)} \\ u_{2,x}^{(1)} & u_{2,y}^{(1)} \end{pmatrix} \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{C.27}$$

where α is the inclination angle between the local crack tip coordinate system and the global coordinate system, see Figure C.1.

The stress of state (1) in the local crack tip coordinate system is as follows

$$\begin{aligned}
\sigma_{x_1x_1} &= \frac{\sigma_{xx}+\sigma_{yy}}{2} + \left(\frac{\sigma_{xx}-\sigma_{yy}}{2}\right)\cos(2\alpha) + \tau_{xy}\sin(2\alpha) \\
\sigma_{y_1y_1} &= \frac{\sigma_{xx}+\sigma_{yy}}{2} - \left(\frac{\sigma_{xx}-\sigma_{yy}}{2}\right)\cos(2\alpha) - \tau_{xy}\sin(2\alpha) \\
\sigma_{x_1y_1} &= \tau_{xy}\cos(2\alpha) - \left(\frac{\sigma_{xx}-\sigma_{yy}}{2}\right)\sin(2\alpha)
\end{aligned} \tag{C.28}$$

With the isoparametric element formulation, the distribution of weighting function within elements can be determined by the usual finite element interpolation

$$q = \sum_{i=1}^{m} N_i q_i \tag{C.29}$$

Hence,

$$\frac{\partial q}{\partial x} = \sum_{i=1}^{m} \frac{\partial N_i}{\partial x} q_i \tag{C.30}$$

A vector transformation is then used to convert $\partial q/\partial x$ to the local crack tip coordinate system $\partial q/\partial x_j$

$$\begin{aligned}
q_{,x_1} &= q_{,x}\cos(\alpha) + q_{,y}\sin(\alpha) \\
q_{,x_2} &= -q_{,x}\sin(\alpha) + q_{,y}\cos(\alpha)
\end{aligned} \tag{C.31}$$

In the following, the displacement, stress and strain field of the auxiliary field are computed. Since we chose the state 2 as the pure mode I with $K_1 = 1$, then we have

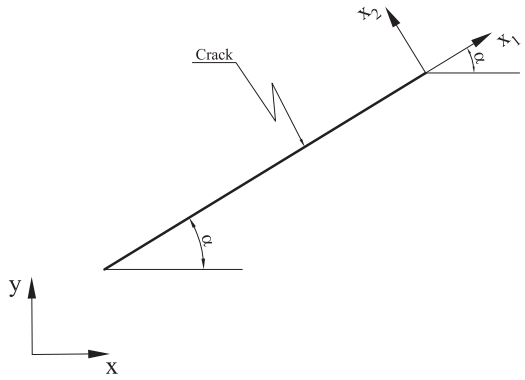


Figure C.1: Global and local coordinate systems

$$
\begin{aligned}
u_i^{(2)} = \begin{Bmatrix} u_x \\ u_y \end{Bmatrix} &= \frac{1}{2\mu}\sqrt{\frac{r}{2\pi}} \begin{Bmatrix} \cos(\theta/2)\left[\kappa - 1 + 2\sin^2(\theta/2)\right] \\ \sin(\theta/2)\left[\kappa + 1 - 2\cos^2(\theta/2)\right] \end{Bmatrix} \\
&= \frac{1}{2\mu}\sqrt{\frac{r}{2\pi}} \begin{Bmatrix} \cos(\theta/2)\left[\kappa - \cos(\theta)\right] \\ \sin(\theta/2)\left[\kappa - \cos(\theta)\right] \end{Bmatrix}
\end{aligned}
\tag{C.32}
$$

where κ and μ are material constants given as follows

$$
\mu = \frac{E}{2(1+\nu)}; \quad \kappa = \begin{cases} 3 - 4\nu & \text{plane strain} \\ \dfrac{3-\nu}{1+\nu} & \text{plane stress} \end{cases}
\tag{C.33}
$$

Letting

$$
\begin{aligned}
A &= \frac{1}{2\mu}; B = \sqrt{\frac{r}{2\pi}} \\
f_1 &= \cos(\theta/2)\left[\kappa - \cos(\theta)\right] \\
f_2 &= \sin(\theta/2)\left[\kappa - \cos(\theta)\right]
\end{aligned}
\tag{C.34}
$$

The strain components of state 2 are

$$\epsilon_{ij}^{(2)} = \frac{1}{2}(u_{i,j}^{(2)} + u_{j,i}^{(2)}) \tag{C.35}$$

The derivatives of the displacement fields are

$$u_{1,1}^{(2)} = A(Bf_{1,1} + \frac{f_1 r_{,1}}{4\pi B}) \tag{C.36a}$$

$$u_{1,2}^{(2)} = A(Bf_{1,2} + \frac{f_1 r_{,2}}{4\pi B}) \tag{C.36b}$$

$$u_{2,1}^{(2)} = A(Bf_{2,1} + \frac{f_2 r_{,1}}{4\pi B}) \tag{C.36c}$$

$$u_{2,2}^{(2)} = A(Bf_{2,2} + \frac{f_2 r_{,2}}{4\pi B}) \tag{C.36d}$$

Since $r_{,1} = \cos(\theta)$, $r_{,2} = \sin(\theta)$, $\theta_{,1} = -\sin(\theta)/r$, and $\theta_{,2} = \cos(\theta)/r$, by using the chain rule, we can write the derivatives of f_1, f_2 as follows

$$f_{1,1} = f_{1,\theta}\theta_{,1}; \quad f_{1,2} = f_{1,\theta}\theta_{,2} \tag{C.37a}$$

$$f_{2,1} = f_{2,\theta}\theta_{,1}; \quad f_{2,2} = f_{2,\theta}\theta_{,2} \tag{C.37b}$$

$$f_{1,\theta} = -\frac{\kappa}{2}\sin\frac{\theta}{2} + \frac{1}{2}\sin\frac{\theta}{2}\cos\theta + \cos\frac{\theta}{2}\sin\theta \tag{C.38a}$$

$$f_{2,\theta} = \frac{\kappa}{2}\cos\frac{\theta}{2} - \frac{1}{2}\cos\frac{\theta}{2}\cos\theta - \sin\frac{\theta}{2}\sin\theta \tag{C.38b}$$

The stress field of Mode I is given by

$$\begin{aligned}
\sigma_{xx}^{(2)} &= \frac{1}{\sqrt{2\pi r}}\cos\frac{\theta}{2}\left(1-\sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right)\\
\sigma_{yy}^{(2)} &= \frac{1}{\sqrt{2\pi r}}\cos\frac{\theta}{2}\left(1+\sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right)\\
\sigma_{xy}^{(2)} &= \frac{1}{\sqrt{2\pi r}}\cos\frac{\theta}{2}\sin\frac{\theta}{2}\cos\frac{3\theta}{2}
\end{aligned} \tag{C.39}$$

All terms in equation (C.25) were calculated, it is ready to compute numerically this interaction integral.

$$I^{(1,2)} = \sum_{\substack{elements\\ inA}}\sum_{p=1}^{P}\left\{\left[\sigma_{ij}^{(1)}\frac{\partial u_i^{(2)}}{\partial x_1} - \sigma_{ij}^{(2)}\frac{\partial u_i^{(1)}}{\partial x_1} - W^{(1,2)}\delta_{1j}\right]\frac{\partial q}{\partial x_j}\right\} w_p \mathrm{detJ} \tag{C.40}$$

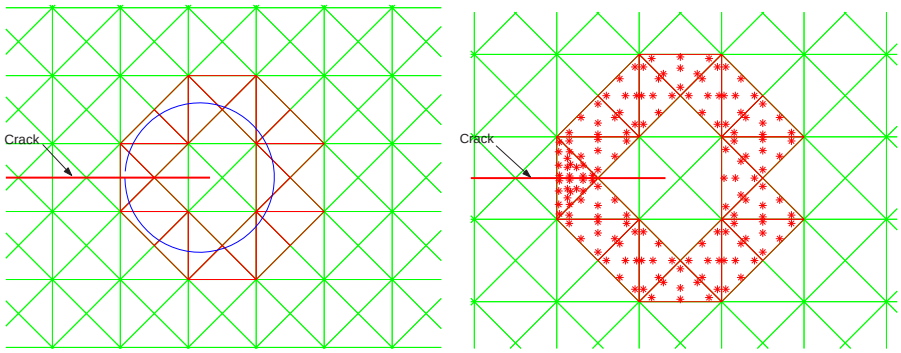
where all terms of the state 2 are functions of variables r, θ. Therefore, it is necessary to compute the coordinates of Gauss points in the local crack tip coordinate system. For a Gauss point with coordinate (r, s), its global coordinates are defined by

$$\begin{aligned}
x &= \sum N_i(r,s)x_i\\
y &= \sum N_i(r,s)y_i
\end{aligned} \tag{C.41}$$

From this global coordinate, it is easy to compute the local coordinate in the local crack tip coordinate system, (x_{loc}, y_{loc}). Then, (r, θ) of this Gauss point is

$$r = \sqrt{x_{loc}^2 + y_{loc}^2}$$
$$\theta = tan^{-1}\frac{y_{loc}}{x_{loc}} \tag{C.42}$$

For the numerical evaluation of the above integral, the domain A is set from the collection of elements which intersect with the circle centered at the crack tip and having predefined radius $r_d = r_k h_{local}$ with r_k is a scalar multiple and h_{local} is the square root of the area of tip element. Figure C.2 shows a typical set of elements for the domain A. The value of weight function q at nodes are also plotted.



(a) Elements selected about the crack tip

(b) Gauss points used to numerically perform the I integral

Figure C.2: Elements used in the interaction integral computation

In the same manner, in order to compute K_{II}, we choose state 2 as the pure Mode II ($K_I^{(2)} = 0$) with $K_{II}^{(2)} = 1$, hence :

$$K_{II} = \frac{E^*}{2} I^{(1,ModeII)} \tag{C.43}$$

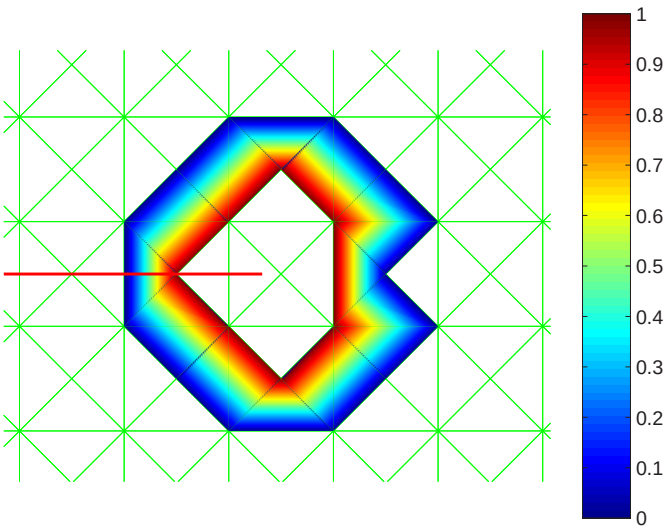


Figure C.3: Weight functions q on the elements

The computation of the interaction integral is implemented as method *CrackTip::computeInteractionIntegral(TimeStep* stepN)*, see file cracttip.cpp, while the terms of the auxiliary fields are implemented in class **AuxiliaryField** (auxiliaryfield.cpp).

C.4 Numerical integration

As indicated in Section 2.2.5, elements split by the crack or contain the crack tip require special treatment for the integration. Instead of doing the numerical integration on the element, say e, we partition this element into sub-triangles and perform the integration on these sub-triangles.

A finite element(parent) is denoted by e_q, whereas e_q^Δ is used for a sub-triangle(child) that belongs to e_q. The coordinates of nodes of e_q^Δ are defined as $x_i^{\Delta_q}$.

For each Gauss point $\xi_q^\Delta \in e_q^\Delta$, the finite interpolation gives its global coordi-

nate

$$\mathbf{x} = \sum_{i=1}^{3} N_i(\xi_q^{\Delta})\mathbf{x}_{\mathrm{i}}^{\Delta_{\mathrm{q}}} \tag{C.44}$$

where N_i is the shape functions of the three-noded triangular elements, which are given here for convenience

$$\begin{aligned} N_1 &= 1 - \xi - \eta \\ N_2 &= \xi \\ N_3 &= \eta \end{aligned} \tag{C.45}$$

Now since the degrees of freedom are defined on the parent element, say the three-noded triangular element, the local coordinate ξ^{Δ} is required. To this end, an inverse map from global to local coordinate system is performed.

$$\mathbf{x} = \sum_{i=1}^{3} N_i(\xi^{\Delta})\mathbf{x}_{\mathrm{i}}^{\Delta} \tag{C.46}$$

where $\mathbf{x}$ is computed from equation (C.44) and $\mathbf{x}_{\mathrm{i}}^{\Delta}$ are coordinates of three nodes of e_q.

In explicit form, equation (C.46) is written as

$$\begin{cases} x = (1 - \xi - \eta)x_1 + \xi x_2 + \eta x_3 \\ y = (1 - \xi - \eta)y_1 + \xi y_2 + \eta y_3 \end{cases} \tag{C.47}$$

or

$$\begin{cases} (x_2 - x_1)\xi + (x_3 - x_1)\eta = x - x_1 \\ (y_2 - y_1)\xi + (y_3 - y_1)\eta = y - y_1 \end{cases} \tag{C.48}$$

From equation (C.48), it is easy to compute the coordinates ξ, η of Gauss point defined in the local coordinate system of e_q. This Gauss point will be used to evaluate the shape functions, enrichment functions.

All of this was implemented in method *Global2Local* of class **FEI2dTriLin**.

If the quadrilateral elements or high order elements are used, then the equation (C.46) is a non-linear system which should be solved using the Newton-Raphson method.

C.5 Maximum hoop stress criterion

In this section, the maximum hoop stress criterion is presented in detail. This criterion states that the crack will propagate from its tip in the direction θ_c where the circumferential stress $\sigma_{\theta\theta}$ is maximum. Under general mixed mode loadings, the asymptotic near tip circumferential and shear stress take the form

$$\begin{aligned} \begin{Bmatrix} \sigma_{\theta\theta} \\ \sigma_{r\theta} \end{Bmatrix} &= \frac{1}{4}\frac{K_I}{\sqrt{2\pi r}} \begin{Bmatrix} 3\cos(\theta/2) + \cos(3\theta/2) \\ \sin(\theta/2) + \sin(3\theta/2) \end{Bmatrix} \\ &+ \frac{1}{4}\frac{K_{II}}{\sqrt{2\pi r}} \begin{Bmatrix} -3\sin(\theta/2) - 3\sin(3\theta/2) \\ \cos(\theta/2) + 3\cos(3\theta/2) \end{Bmatrix} \end{aligned} \tag{C.49}$$

The circumferential stress in the direction of crack growth is a principal stress. Therefore, the critical angle θ_c defining the radial direction of propagation can be

determined by setting the shear stress $\sigma_{r\theta}$ in equation (C.49) to zero. After some trigonometrical manipulations, the following expression is obtained

$$\frac{1}{\sqrt{2\pi r}}\cos(\frac{\theta}{2})\left[\frac{1}{2}K_I\sin(\theta)+\frac{1}{2}K_{II}(3\cos(\theta)-1)\right]=0 \tag{C.50}$$

This leads to the equation defining the angle of crack propagation θ_c in the tip coordinate system

$$K_I\sin(\theta_c)+K_{II}(3\cos(\theta_c)-1)=0 \tag{C.51}$$

By using $t=\tan(\theta_c/2)$, and solving an quadratic equation, we obtain

$$\theta_c=2\arctan\left[\frac{1}{4}\left(\frac{K_I}{K_{II}}\pm\sqrt{\left(\frac{K_I}{K_{II}}\right)^2+8}\right)\right] \tag{C.52}$$

If $K_{II}=0$ then $\theta_c=0$ (pure mode I) and by noting that if $K_{II}>0$, the crack growth angle $\theta_c<0$, and if $K_{II}<0$ then, $\theta_c>0$, a more efficient expression for θ_c is implemented (Suo,2002):

$$\theta_c=2\arctan\left[\frac{-2K_{II}/K_I}{1+\sqrt{1+8(K_{II}/K_I)^2}}\right] \tag{C.53}$$