1-1-2006

Extended finite element simulation of fracture mechanisms in composite materials.

Siamak Tavoosfard
University of Windsor

NOTE TO USERS

This reproduction is the best copy available.

EXTENDED FINITE ELEMENT SIMULATION OF FRACTURE MECHANISMS IN COMPOSITE MATERIALS

by

Siamak Tavoosfard

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through Engineering Materials
In Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

2006

Your file Votre référence
ISBN: 978-0-494-42316-5
Our file Notre référence
ISBN: 978-0-494-42316-5

ABSTRACT

Due to the increasing use of aluminum silicon alloys in automotive industries for reduced weight in vehicles and decreased emission of pollutants for improvements in fuel economy, the fracture characteristics and wear resistance of aluminum alloys have become of great interest. Hypereutectic Al-Si alloys have the potential to eliminate cast iron liners in engine blocks. One of the challenges for casting difficulties of hypereutectic Al-Si Alloys is coarse primary silicon particles size, irregular shape and uneven distribution. Deformation of surface and subsurface layers and also fracture of subsurface layer in this nano-composite structure are the determining factors of the wear resistance properties of the material. The major concern of this work is focused on developing a numerical model of fracture (crack propagation) in aluminum-silicon nano-composites. More particularly, an extended finite element approach couple with the level set method is used.

Dedicated to my wife Haleh

ACKNOWLEDGEMENTS

This thesis is the account of two years of concerted work at University of Windsor, which would not have been possible without the help of many. A few lines are too short to make a complete account of my deep appreciation for my advisor, Vesselin Stoilov. I wish to first thank him for his unfaltering trust and constant encouragements, which have been essential ingredients in the construction of our success throughout the last two years. I wish to thank him for who he is as a professor and manager, an enthusiastic, sagacious, square and honest minded individual with broad interests who provides unwavering support in the most difficult times. But I want to extend my appreciation to him for his understanding and account of life outside work. His trust and honesty, his efforts in understanding a student's personality and tailoring his management accordingly translated for me into very pleasurable two years at University of Windsor. I also wish to thank Vesselin Stoilov for being such an attainable, likable and witty professor and individual, helping a student feel at ease and comfortable. My advisor's approach to research and science inspired, after long internal struggles, my pursuing of an academic career. I wish to thank him for that. It has been a distinct privilege for me to work with Professor Vesselin Stoilov.

I would like to extend my thanks to Professor Ahmet Alpas for accepting me so readily as a new addition to the research team.

I thank all the professors here at University of Windsor that have dispensed wonderful lessons to me at the university. I am thinking especially about Professors Northwood for Advanced Crystallography , Altenhof for the Crashworthiness and Impact Analysis by Finite Element, Hu for Casting Modelling and Simulation, for Introduction to Finite Element Analysis, Zamani.

I would not be sitting in front of my monitor typing these acknowledgment lines without my Mom and Dad. I owe my parents, Ismail and Tahereh much of what I have become. I thank them for their love, their support and their

confidence throughout the past years. My parents have always put education as a first priority in my life, and raised me to set high goals for myself. They taught me to value honesty, courage and humility above all other virtues. I have always needed to work hard to achieve my goals in life and they have always been there for me as an unwavering support. But for my wife, Haleh who has always stood by me and comforted me on the numerous occasions when I felt downcast and dispirited, this work could not have been achieved. I thank her for her love and devotion, which, with her earnest application to helping me see through the mist of discouragement were essential to my emotional and psychological balance and thus to my success in this enterprize. I also thank her for wordlessly bearing with my long hours in the lab, my little availability, recurrent bad temper and irascibility. Haleh has never known me otherwise than as a graduate student and I dedicate this work to her, to honor her love, patience and genuine attention during these years.

This thesis has been financially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and General Motors of Canada (GM) which is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Aluminium alloys have the potential to reduce vehicles weight and decrease emission of pollutants by improvements in fuel economy. However, no aluminum alloy has been developed or identified that provides the required combination of casting, machining and wear resistance properties for manufacturing engine block. Engine manufacturers have tended to select castable and machinable alloys and modify the surfaces of the cylinder walls to obtain the necessary wear resistance. Thus, when current cylinder blocks are cast of alloys such as AA 319 or AA 356 they require cylinder liners (cast iron, metal matrix composite, AA390) or surface treatment (plating, coating) to provide wear resistance during operation. Cast iron liners have been placed like cores in the casting mold or inserted in the machined cylinder bores. Other wear resistant liner compositions have also been used. As an alternative to cylinder liners, wear resistant coatings have been applied to the cylinder walls of the cast and machined block. Each of these modifications to the block increases the cost of the product. In addition, engine blocks with cast iron liners suffer because of the added weight, and relatively high thermal stresses due to different thermal conductivity and thermal expansion of the two materials. Therefore there still remains a need for an aluminum alloy that provides all of the above properties for manufacturing engine block and wear properties. The requirements for an aluminum alloy intended for mass production of an all-aluminum cylinder block for an automotive engine are very demanding. The alloy must require minimal post casting operations, such as heat-treatment, machining and assembly.

Germany is the leading producer of the all-aluminum performance blocks (by Porsche, Daimler-Benz and BMW). Conversely, US automotive manufacturers are still without an all-aluminum block in production. This state of affairs may change because the Japanese automotive manufacturers have a stated goal of decreasing the weight of the automobile by 40%. Thus, the cast iron block and the aluminum

block with cast iron liners, both will be unacceptable in the relatively near future for automakers that are interested in gaining or maintaining market share. Usage of all-aluminum engine blocks provides performance-based benefits including reduced weight, improve thermal conductivity and greater rigidity. The thermal conductivity of hypereutectic Al-Si alloys is nearly 400% higher than that of cast iron. This high thermal conductivity is one of the most useful and important properties. The thermal and physical properties of the copper-free aluminum silicon hypereutectic alloys are closely related to and dependent on the silicon content of the alloy. For example, with increasing silicon content the coefficient of thermal expansion decreases where as the modules of elasticity increases. Finally, wear resistance is provided by the volume fraction of primary silicon particles in the microstructure. There is nearly a doubling of the wear resistance in going from the 16% silicon content to the 20% silicon content because the volume fraction of primary silicon doubles with this composition [12].

One of the challenges for casting difficulties of hypereutectic Al-Si Alloys is coarse primary silicon particles size, irregular shape and uneven distribution. The development of an optimization module integrated with simulation codes will facilitate the fast set up of the most suitable casting parameters while additionally reducing internal defects. By using numerical optimization, expensive and time consuming "trial and error" iterations are reduced to the minimum, or even eliminated.

Computer simulation of fracture processes remains a challenge for many industrial modelling problems. The basic question is how to incorporate the discontinuity of the displacement field into the finite element model. One straightforward method is to enforce mesh lines along the crack, i.e., to create a new mesh at each propagation step as the crack propagates with time. In standard finite element method, the non-smooth displacement near the crack tip is captured by refining the mesh locally. The number of degrees of freedom may drastically increase, especially in three dimensional applications. Moreover the incremental computation

of a crack growth needs frequent remeshings. Reprocessing the solution on the updated mesh is not only a costly operation but also it may have a troublesome impact on the quality of results. Also in standard finite-element analysis the representation of discontinuities in the displacement and/or strain fields requires the alignment of mesh boundaries to the discontinuity line or surface. The extended finite-element method (XFEM) enriches the standard finite-element basis through a local partition of unity (PU), Babuska and Melenk [22]. This was first applied to fracture problems by Belytschko et al. [3], where the asymptotic nearfield for a crack was incorporated by a local PU and the discontinuity in this field was used to represent the crack discontinuity independent of the mesh.

When multiple crack segments are needed to be enriched with the near tip fields, a mapping algorithm introduced by Fleming et al. [14] is used to align the discontinuity with the crack geometry. They also proved that the use of discontinuous displacements along the crack produces a solution with zero traction along the crack faces. Moës et al. [23] introduced a much more elegant and straightforward procedure to introduce a discontinuous field across the crack faces away from the crack tip by adapting the generalized Heaviside function, and developed simple rules for the introduction of the discontinuous and crack tip enrichments. Later, Daux et al. [8] introduced the junction function concept to account for multiple branched cracks and named their method the extended finite element method (XFEM). They have employed this method for modelling complicated geometries such as multiple branched cracks, voids and cracks emanating from holes without the need for the geometric entities to be meshed. Sukumar et al. [32] studied cracks in three dimensions with the XFEM. Dolbow et al. [10] studied 2D crack growth under three different interfacial constitutive laws on the crack faces: perfect contact and unilateral contact with or without friction [11]. In XFEM, a standard finite element mesh for the problem is first created without accounting for the geometric entity. The presence of cracks, voids or inhomogeneities is then represented independently of the mesh by enriching the standard displacement

approximation with additional functions. For crack modelling, both discontinuous displacement fields along the crack faces and the leading singular crack tip asymptotic displacement fields are added to the displacement based finite element approximation through the PU method. The additional coefficients at each enriched node are independent. In addition, XFEM provides a seamless means to use higher order elements or special finite elements without significant changes in the formulation. The XFEM will also improve the accuracy in problems where some aspects of the functional behavior of the solution field is known a priori and relevant enrichment functions can then be used.

In order to model complex crack configurations, more powerful and convenient techniques for representing internal discontinuities are required. The level set method (LSM) developed by Osher and Sethian [25] [29] for modelling the motion of interfaces is very promising. It represents the interface as the zero level set of a function of one higher dimension. With the use of the LSM, the motion of the interface is computed on a fixed mesh. The LSM handles topology changes of the interface naturally, and extending it to higher dimensions is easy. The geometric properties of the interface can also be obtained from the level set function. By coupling the LSM with the XFEM, Stolarska et al. [31] studied the growth of a fatigue crack; Belytschko et al. [5] studied several frictionless contact problems and provided the level set functions of discontinuities in a function, in a specific component of a function as well as in its derivatives.

Iarve [17] replaced the Heaviside step function with a higher order polynomial B-spline shape function approximation. Stazi et al. [30] investigated quadratic background elements for linear elastic fracture mechanics able to represent a crack with curvature. Fan et al. [13] enriched the crack tip node with singular as well as higher order terms of the crack tip asymptotic field. The accuracy of the directly determined stress intensity factors (SIFs) can be significantly improved. However, for general mixed mode cracks, the SIFs are still not very accurate, since

the enrichment approximation adjacent to the crack tip cannot reduce to the actual crack tip field. Xiao and Karihaloo [34], and Liu et al. [21] ensured that the enriched approximation is equivalent to the crack tip asymptotic field. They obtained SIFs directly for homogeneous as well as interfacial cracks. Belytschko et al. [4] developed a new method for handling a discontinuity that ends within an element, and showed how to switch from a continuum description to a discrete discontinuity for rate-independent materials when the governing partial differential equation for momentum loses hyperbolicity (i.e. the differential equations of equilibrium lose ellipticity). The loss of hyperbolicity is tracked enabling both the crack speed and crack direction to be determined for a given material model. Ventura et al. [33] described the level set in two dimensions by the sign of the level set function and the components of the closest point projection to the surface. The update of the level set was constructed by geometric formulas. Chessa and Belytschko [6] introduced arbitrary discontinuities in space-time using a local enrichment to discontinuities along a moving hyper-surface. They showed that by capturing the discontinuity in time as well as space the results are improved in comparison with capturing the discontinuity in space alone. For stationary and growing cracks, Lee et al. [19] modelled the near-tip field by superimposing quarter point elements on an overlaid mesh while the rest of the discontinuity was implicitly described by a step function.

Moës et al. [24] and Gravouil et al. [15] studied non-planar 3D crack growth. Chopp and Sukumar [7] studied propagation of multiple coplanar cracks. Liang et al. [20] studied evolving crack patterns in thin films. Many works have also appeared on the cohesive crack model, as reviewed by de Borst et al. [9].

Rubinstein [28] has shown that relatively small errors in the determination of the crack path deflection angle can lead to a significant cumulative deviation of the crack path over a finite crack length. Therefore, a reliable analysis of crack propagation requires not only a suitable criterion of crack growth but also accurate evaluation of the crack tip stress field.

Recently, Bchet et al. [2] and Laborde et al. [18] introduced singular mappings for numerical integration of the singularity in r- or radial direction.

CHAPTER II

STANDARD FINITE ELEMENT METHOD

2.1 What is the Finite Element Method?

The finite element method is a numerical technique for obtaining approximate solutions to the partial differential equations that arise in scientific and engineering applications.

2.2 How the Finite Element Method works?

In a continuum problem of any dimension, the field variable (whether it is pressure, temperature, displacement, stress, or some other quantity) possesses infinitely many values because it is a function of each generic point in the body or solution region. Consequently, the problem is one with an infinite number of unknowns. The finite element discretization procedures reduce the problem to one of a finite number of unknowns by dividing the solution region into elements and by expressing the unknown field variable in terms of assumed approximating functions within each element. The approximating functions (sometimes called interpolation functions) are defined in terms of the values of the field variables at specified points called nodes or nodal points. Nodes usually lie on the element boundaries where adjacent elements are connected. In addition to boundary nodes, an element may also have a few interior nodes. The nodal values of the field variable and the interpolation functions for the elements completely define the behavior of the field variable within the elements. For the finite element representation of a problem the nodal values of the field variable become the unknowns. Once these unknowns are found, the interpolation functions define the field variable throughout the assemblage of elements. Clearly, the nature of the solution and the degree of approximation depend not only on the size and number of the elements used but also on the interpolation functions selected. We cannot choose functions arbitrarily, because certain compatibility conditions should be satisfied. Often functions

are chosen so that the field variable or its derivatives are continuous across adjoining element boundaries. The important feature of the finite element method is the ability to formulate solutions for individual elements before putting them together to represent the entire problem. This means, for example, that if we are treating a problem in stress analysis, we find the force-displacement or stiffness characteristics of each individual element and then assemble the elements to find the stiffness of the whole structure. In essence, a complex problem reduces to considering a series of greatly simplified problems.

Another advantage of the finite element method is the variety of ways in which one can formulate the properties of individual elements. There are basically three different approaches. The first approach to obtaining element properties is called the direct approach because its origin is traceable to the direct stiffness method of structural analysis. However the direct approach can be used only for relatively simple problems.

Element properties obtained by the direct approach can also be determined by the variational approach. The variational approach relies on the calculus of variations and involves extremizing a functional. For problems in solid mechanics the functional turns out to be the potential energy, the complementary energy, or some variant of these. Knowledge of the variational approach is necessary to work beyond the introductory level and to extend the finite element method to a wide variety of engineering problems.

A third and even more versatile approach to deriving element properties is the weighted residuals approach. The weighted residuals approach begins with the governing equations of the problem and proceeds without relying on a variational statement. This approach is advantageous because it thereby becomes possible to extend the finite element method to problems where no functional is available. The method of weighted residuals is widely used to derive element properties for nonstructural applications such as heat transfer and fluid mechanics.

Regardless of the approach used to find the element properties, the solution of

a continuum problem by the finite element method always follows an orderly step-by-step process. To summarize in general terms how the finite element method works we will succinctly list these steps now.

1. Discretize the Continuum: The first step is to divide the continuum or solution region into elements. A variety of element shapes may be used, and different element shapes may be employed in the same solution region. Indeed, when analyzing an elastic structure that has different types of components such as plates and beams, it is not only desirable sometimes but also necessary to use different elements in the same solution. Although the number and the type of elements in a given problem are matters of engineering judgment, the analyst can rely on the experience of others for guidelines.

2. Select Interpolation Functions: The next step is to assign nodes to each element and then choose the interpolation function to represent the variation of the field variable over the element. The field variable may be a scalar, a vector, or a higher-order tensor. Often, polynomials are selected as interpolation functions for the field variable because they are easy to integrate and differentiate. The degree of the polynomial chosen depends on degree of approximation, the nature and number of unknowns at each node, and certain continuity requirements imposed at the nodes and along the element boundaries. The magnitude of the field variable as well as the magnitude of its derivatives may be the unknowns at the nodes.

3. Find the Element Properties: Once the finite element model has been established (that is, once the elements and their interpolation functions have been selected), we are ready to determine the matrix equations expressing the properties of the individual elements. For this task we may use one of the three approaches just mentioned: the direct approach, the variational approach, or the weighted residuals approach.

4. Assemble the Element Properties to Obtain the System Equations. To find the properties of the overall system modeled by the network of elements we must assemble all the element properties. In other words, we combine the

matrix equations expressing the behavior of the elements and form the matrix equations expressing the behavior of the entire system. The matrix equations for the system have the same form as the equations for an individual element except that they contain many more terms because they include all nodes. The basis for the assembly procedure stems from the fact that at a node, where elements are interconnected, the value of the field variable is the same for each element sharing that node. A unique feature of the finite element method is that the system equations are generated by assembly of the individual element equations.

5. Impose the Boundary Conditions: Before the system equations are ready for solution they must be modified to account for the boundary conditions of the problem. At this stage we impose known nodal values of the dependent variables or nodal loads.

6. Solve the System Equations: The assembly process gives a set of simultaneous equations that we solve to obtain the unknown nodal values of the problem. If the problem describes steady or equilibrium behavior, then we must solve a set of linear or nonlinear algebraic equations. If the problem is unsteady, the nodal unknowns are a function of time, and we must solve a set of linear or nonlinear ordinary differential equations.

7. Make Additional Computations If Desired: Many times we use the solution of the system equations to calculate other important parameters. For example, in a structural problem usually the nodal unknowns are displacement components. From these displacements we calculate element strains and stresses. Similarly, in a heat-conduction problem the nodal unknowns are temperatures, and from these we calculate element heat fluxes.

2.3 Basic Steps for Solving a Problem

There are several basic steps to solving a problem by the FEM method. These steps are:

1. Formulation of the problem into a differential or integral equation.
2. Development of the weak formulation of the problem.

3. Finite element approximation of the domain and variables.
4. Assembling the stiffness matrix and load vector and solving the problem.
5. Post processing (analysis of the results).

2.3.1 Steady State Heat Conduction in a Bar

We will proceed to describe the steps in the finite element method by the aid of an example. Consider the problem of heat conduction through a bar.

2.3.2 Step 1: Formulation of the problem

The problem under consideration is that of the steady state flow of heat through a bar. Figure 2.1 shows how heat flows through an element of this bar. Balancing the heat entering and exiting this element gives

$$q(x) + f(x)\Delta(x) = q(x + \Delta x), \tag{2.1}$$

where q is the heat flux along the bar and $f(x)$ is the heat flow through the lateral surfaces per unit length of the bar. Reorganizing and taking the limit as Δx goes to zero yields

$$\frac{dq}{dx} = f(x) \tag{2.2}$$

Introducing Fouriers law of heat conduction ($q = -k\frac{d\theta}{dx}$) into this equation results

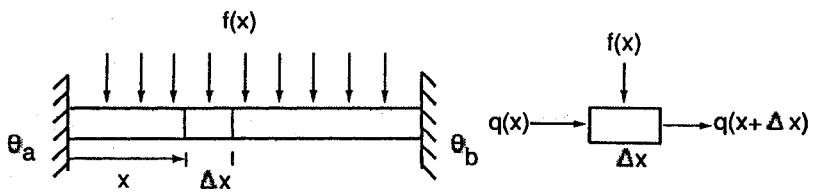


Figure 2.1: Steady state thermal conduction in a bar

in

$$-k\frac{d^2\theta}{dx^2} = f(x), \tag{2.3}$$

where θ is the temperature and k is the coefficient of thermal conduction.

Since the problem is a second order ordinary differential equation one can prescribe two conditions. We will look at solving the above differential equation subject to any one of the following boundary conditions:

$\theta = \theta_a$ at $x = a$ and $\theta = \theta_b$ at $x = b$,

$\theta = \theta_a$ at $x = a$ and $q = -k\frac{d\theta}{dx} = q_b$ at $x = b$,

$q = -k\frac{d\theta}{dx} = q_a$ at $x = a$ and $\theta = \theta_b$ at $x = b$, or

$q = -k\frac{d\theta}{dx} = q_a$ at $x = a$ and $q = -k\frac{d\theta}{dx} = q_b$ at $x = b$.

2.3.3 Step 2: Formulation of the weak form

The solution to this problem is a temperature field which satisfies the differential equation

$$-k\frac{d^2\theta}{dx^2} = f(x) \tag{2.4}$$

at every point in the bar. We will relax this requirement by replacing this equation by

$$-\int_R k\bar{\theta}\frac{d^2\theta}{dx^2}dx = \int_R \bar{\theta}f(x)dx, \tag{2.5}$$

where R is any region of the bar and $\bar{\theta}$ is a test function. If one requires that this equation hold for all possible test functions $\bar{\theta}$, it will be possible to get the original differential equation from this integral equation. Any point in the bar can be isolated by selecting a test function which is only positive and is nonzero only in a small region around that point. The aim is not to require the integral equation to hold for all possible $\bar{\theta}$ functions, but to select a set of test functions and require the integral equality to hold for this set of functions. Hence, we do not require the differential equation to be exactly satisfied at all points, but require the temperature field to only satisfy the integral equation for a select number of test functions (we have weakened the original equation).

We will now use integration by parts to get the weak form of the problem as

$$-[k\bar{\theta}\frac{d\theta}{dx}]_{x=a}^{b} + \int_a^b k\frac{d\bar{\theta}}{dx}\frac{d\theta}{dx} = \int_a^b \bar{\theta}f(x)dx, \tag{2.6}$$

subject to the appropriate boundary conditions. The application of integration by parts also introduces a weakening of the problem since we are no longer required to have temperature fields which have a second derivative. It is now only required that the temperature field have a first derivative as can be seen from equation.

2.3.4 Step 3: Finite element approximation

We will use two node line elements to approximate both the temperature field and the test functions. The length of the bar will be partitioned into ne elements as shown in Figure 2.2. Each element has an element number and one node at each end. Nodes are given two numbers. The global node number is one which is unique for each node and which distinguishes it from all other nodes. For each element there is also a local numbering system. For a two node element the local nodes are distinguished in the element by designating a local node number of 1 to one node and 2 to the other node.

The temperature and the test function will be approximated over each element by

$$\theta = \theta_1^e N_1(x) + \theta_2^e N_2(x) = \begin{bmatrix} N_1(x) & N_2(x) \end{bmatrix} \begin{bmatrix} \theta_1^e \\ \theta_2^e \end{bmatrix} \tag{2.7}$$

and

$$\bar{\theta} = \bar{\theta}_1^e N_1(x) + \bar{\theta}_2^e N_2(x) = \begin{bmatrix} \bar{\theta}_1^e & \bar{\theta}_2^e \end{bmatrix} \begin{bmatrix} N_1(x) \\ N_2(x) \end{bmatrix} \tag{2.8}$$

where $\bar{\theta}_i^e$ is the value of θ at local node i, $\bar{\theta}_i^e$ is the value of θ at local node i, and

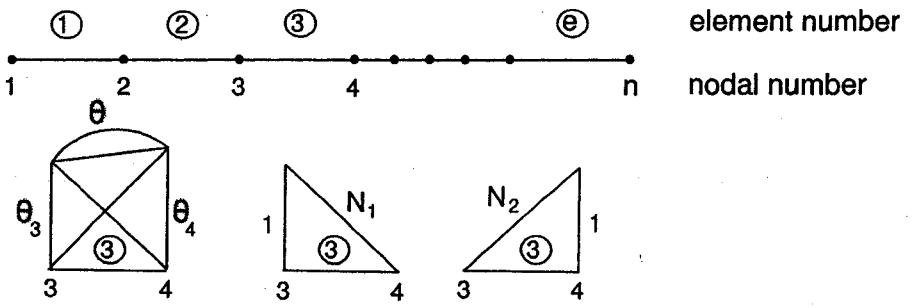


Figure 2.2: Approximation of θ and $\bar{\theta}$ by the use of shape functions.

$N_i(x)$ are known functions of x. The N_i are known as shape functions. The shape functions are selected such that $N_1 = 1$ at local node 1, $N_1 = 0$ at local node 2, $N_2 = 0$ at local node 1, $N_2 = 1$ at local node 2. For a two node line element of length l_e and local coordinate s as shown in Figure 2.3, the shape functions will

be

$$N_1 = 1 - \frac{s}{l_e} \tag{2.9}$$

and

$$N_2 = \frac{s}{l_e} \tag{2.10}$$

The shape functions are selected such that the sum of the shape functions at each point will add up to unity. It must be noted that the shape functions are different for different elements, even though this is not made explicit in the notation.

The two integrals in the weak formulation of the problem, given in equation (2.6), can each be written as a sum of integrals, each integral being over one element. This can be written as

$$\int_a^b k\frac{d\bar{\theta}}{dx}\frac{d\theta}{dx}dx = \sum_{e=1}^{ne}\int_{\Omega_e} k\frac{d\bar{\theta}}{dx}\frac{d\theta}{dx}dx, \tag{2.11}$$

and

$$\int_a^b \bar{\theta}f(x)dx = \sum_{e=1}^{ne}\int_{\Omega_e} \bar{\theta}f(x)dx, \tag{2.12}$$

where Ω is the domain of element number e.

Over each element one can use the above approximations for the temperature and test function to get

$$\int_\Omega k\frac{d\bar{\theta}}{dx}\frac{d\theta}{dx}dx = \int_0^{l_e} k\frac{d[\bar{\theta}_1^e N_1 + \bar{\theta}_2^e N_2]}{ds}\frac{d[\theta_1^e N_1 + \theta_2^e N_2]}{ds}ds \tag{2.13}$$

and

$$\int_\Omega \bar{\theta}f(x)dx = \int_0^{l_e} [\bar{\theta}_1^e N_1 + \bar{\theta}_2^e N_2]f(x)ds, \tag{2.14}$$

since $x = x_1 + s$, where x_1 is the location of the first node in the element. Since θ_1^e, θ_2^e , $\bar{\theta}_1^e$, and $\bar{\theta}_2^e$ are constants, one can write

$$\frac{d\bar{\theta}}{dx} = \frac{d[\bar{\theta}_1^e N_1 + \bar{\theta}_2^e N_2]}{ds} = \begin{bmatrix}\bar{\theta}_1^e & \bar{\theta}_2^e\end{bmatrix}\begin{bmatrix}\frac{dN_1}{ds} \\ \frac{dN_2}{ds}\end{bmatrix} \tag{2.15}$$

and

$$\frac{d\theta}{dx} = \frac{d[\theta_1^e N_1 + \theta_2^e N_2]}{ds} = \begin{bmatrix}\frac{dN_1}{ds} & \frac{dN_2}{ds}\end{bmatrix}\begin{bmatrix}\theta_1^e \\ \theta_2^e\end{bmatrix} \tag{2.16}$$

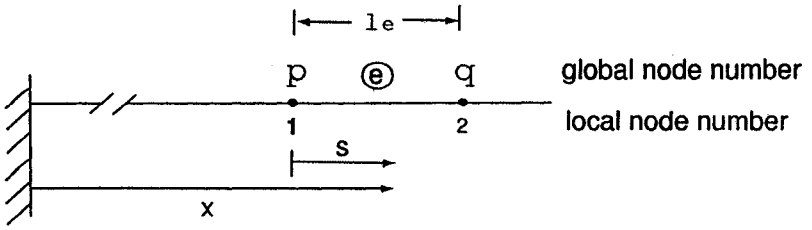


Figure 2.3: Relation between local and global coordinates.

Substitution of these relations into the integrals of equations (2.13) and (2.14) gives

$$\int_\Omega k\frac{d\bar{\theta}}{dx}\frac{d\theta}{dx}dx = k\begin{bmatrix}\bar{\theta}_1^e & \bar{\theta}_2^e\end{bmatrix}\int_0^{l_e}\begin{bmatrix}\frac{dN_1}{ds}\\ \frac{dN_2}{ds}\end{bmatrix}\begin{bmatrix}\frac{dN_1}{ds} & \frac{dN_2}{ds}\end{bmatrix}ds\begin{bmatrix}\theta_1^e\\ \theta_2^e\end{bmatrix} \tag{2.17}$$

and

$$\int_\Omega \bar{\theta}f(x)dx = \begin{bmatrix}\bar{\theta}_1^e & \bar{\theta}_2^e\end{bmatrix}\int_0^{l_e}\begin{bmatrix}N_1\\ N_2\end{bmatrix}f(x)ds. \tag{2.18}$$

These equations can be written as

$$\int_\Omega k\frac{d\bar{\theta}}{dx}\frac{d\theta}{dx}dx = [\bar{\theta}^e]^T[K^e][\theta^e], \tag{2.19}$$

and

$$\int_\Omega \bar{\theta}f(x)dx = [\bar{\theta}^e]^T[f^e], \tag{2.20}$$

where

$$[\bar{\theta}^e] = \begin{bmatrix}\bar{\theta}_1^e\\ \bar{\theta}_2^e\end{bmatrix} \tag{2.21}$$

$$[\theta^e] = \begin{bmatrix}\theta_1^e\\ \theta_2^e\end{bmatrix} \tag{2.22}$$

$$[K^e] = \begin{bmatrix}K_{11}^e & K_{12}^e\\ K_{21}^e & K_{22}^e\end{bmatrix} = k\int_0^{l_e}\begin{bmatrix}(\frac{dN_1}{ds})^2 & \frac{dN_1}{ds}\frac{dN_2}{ds}\\ \frac{dN_2}{ds}\frac{dN_1}{ds} & (\frac{dN_2}{ds})^2\end{bmatrix}ds, \tag{2.23}$$

$$[f^e] = \begin{bmatrix}f_1^e\\ f_2^e\end{bmatrix} = \int_0^{l_e}\begin{bmatrix}N_1\\ N_2\end{bmatrix}f(x)ds. \tag{2.24}$$

The matrix $[K^e]$ is known as the element stiffness matrix and the matrix $[f^e]$ is known as the element load vector. For the shape functions in (2.9) and (2.10) we

have $\frac{dN_1}{ds} = -\frac{1}{l_e}$ and $\frac{dN_2}{ds} = \frac{1}{l_e}$. Therefore, the element stiffness matrix can be written as

$$[K^e] = k\int_0^{l_e} \begin{bmatrix} \frac{1}{l_e^2} & -\frac{1}{l_e^2} \\ -\frac{1}{l_e^2} & \frac{1}{l_e^2} \end{bmatrix} ds = \begin{bmatrix} \frac{k}{l_e} & -\frac{k}{l_e} \\ -\frac{k}{l_e} & \frac{k}{l_e} \end{bmatrix}. \tag{2.25}$$

To evaluate the element load vector one needs to know $f(x)$. We will later calculate this for a particular example. The weak form of the problem can now be written as

$$[\bar{\theta}_q]_{x=a}^b + \sum_{e=1}^{ne} [\bar{\theta}^e]^T [K^e][\theta^e] = \sum_{e=1}^{ne} [\bar{\theta}^e]^T [f^e], \tag{2.26}$$

where Fouriers law is used to replace the derivative of temperature in the first term.

2.3.5 Step 4: Assembling the stiffness matrix and load vector

The next step is to organize the problem in the form of a set of algebraic equations. Consider the example of a problem with two elements and three nodes as shown in Figure 2.4. The two summations in equation (2.26) can be written as

$$\begin{aligned} \sum_{e=1}^{2} [\bar{\theta}^e]^T [K^e][\theta^e] &= \begin{bmatrix} \bar{\theta}_1^1 & \bar{\theta}_2^1 \end{bmatrix} \begin{bmatrix} K_{11}^1 & K_{12}^1 \\ K_{21}^1 & K_{22}^1 \end{bmatrix} \begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} + \begin{bmatrix} \bar{\theta}_1^2 & \bar{\theta}_2^2 \end{bmatrix} \begin{bmatrix} K_{11}^2 & K_{12}^2 \\ K_{21}^2 & K_{22}^2 \end{bmatrix} \begin{bmatrix} \theta_1^2 \\ \theta_2^2 \end{bmatrix} \\ &= \begin{bmatrix} \bar{\theta}_1 & \bar{\theta}_2 & \bar{\theta}_3 \end{bmatrix} \begin{bmatrix} K_{11}^1 & K_{12}^1 & 0 \\ K_{21}^1 & K_{22}^1 + K_{11}^2 & K_{12}^2 \\ 0 & K_{21}^2 & K_{22}^2 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \end{aligned} \tag{2.27}$$

and

$$\begin{aligned} \sum_{e=1}^{2} [\bar{\theta}^e]^T [f^e] &= \begin{bmatrix} \bar{\theta}_1^1 & \bar{\theta}_2^1 \end{bmatrix} \begin{bmatrix} f_1^1 \\ f_2^1 \end{bmatrix} + \begin{bmatrix} \bar{\theta}_1^2 & \bar{\theta}_2^2 \end{bmatrix} \begin{bmatrix} f_1^2 \\ f_2^2 \end{bmatrix} \\ &= \begin{bmatrix} \bar{\theta}_1 & \bar{\theta}_2 & \bar{\theta}_3 \end{bmatrix} \begin{bmatrix} f_1^1 \\ f_2^1 + f_1^2 \\ f_2^2 \end{bmatrix} \end{aligned} \tag{2.28}$$

Therefore, for this problem the weak formulation given in equation (2.26) can be put in the form

$$[\bar{\theta}_{3q3} - \bar{\theta}_{1q1}] + [\bar{\theta}]^T [K][\theta] = [\bar{\theta}][f], \tag{2.29}$$

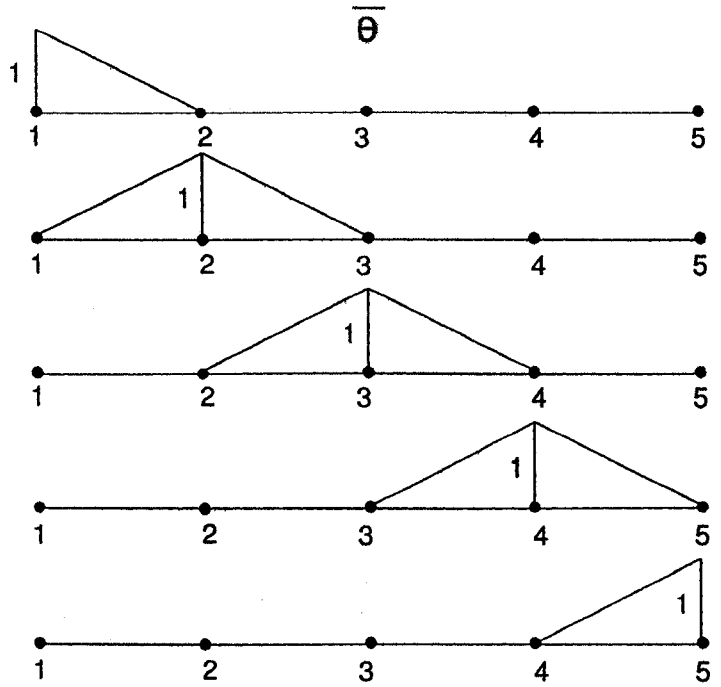


Figure 2.4: Test functions used to get the final form of the equation.

where $[K]$ is known as the global stiffness matrix, $[f]$ is known as the global load vector, and

$$[\bar{\theta}] = \begin{bmatrix} \bar{\theta}_1 \\ \bar{\theta}_2 \\ \bar{\theta}_3 \end{bmatrix}, \tag{2.30}$$

$$[\theta] = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}, \tag{2.31}$$

$$[K] = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} = \begin{bmatrix} K_{11}^1 & K_{12}^1 & 0 \\ K_{21}^1 & K_{22}^1 + K_{11}^2 & K_{12}^2 \\ 0 & K_{21}^2 & K_{22}^2 \end{bmatrix}, \tag{2.32}$$

$$[f] = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} f_1^1 \\ f_2^1 + f_1^2 \\ f_2^2 \end{bmatrix}, \tag{2.33}$$

To illustrate the way in which one arrives at the final set of equations we will write the expanded form of equation (2.29) in the form

$$[\bar{\theta}_{3q3} - \bar{\theta}_{1q1}] + \begin{bmatrix} \bar{\theta}_1 & \bar{\theta}_2 & \bar{\theta}_3 \end{bmatrix} \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} \bar{\theta}_1 & \bar{\theta}_2 & \bar{\theta}_3 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \quad (2.34)$$

The test functions $\bar{\theta}$ are arbitrary functions which we will select in the manner shown in Figure 2.4. Three test functions are selected since we will have three unknowns in this problem (θ_1, θ_2, θ_3; q_1, θ_2, θ_3; θ_1, θ_2, q_2; or q_1, θ_2, q_2). Each test function provides one algebraic equation for the unknowns. As can be seen from Figure 2.4, the first test function has $\bar{\theta}_1 = 1$, $\bar{\theta}_2 = 0$, and $\bar{\theta}_3 = 0$. The second test function has $\bar{\theta}_1 = 0$, $\bar{\theta}_2 = 1$, and $\bar{\theta}_3 = 0$. The third test function has $\bar{\theta}_1 = 0$, $\bar{\theta}_2 = 0$, and $\bar{\theta}_3 = 1$. The following three equations are obtained from the substitution of the selected test functions into equation (2.34).

$$-q_1 + K_{11}\theta_1 + K_{12}\theta_2 + K_{13}\theta_3 = f_1,$$

$$K_{21}\theta_1 + K_{22}\theta_2 + K_{23}\theta_3 = f_2,$$

$$q_3 + K_{31}\theta_1 + K_{32}\theta_2 + K_{33}\theta_3 = f_3, \quad (2.35)$$

These equations can now be solved for the three unknowns. If q_1 and q_3 are specified, then this system can be written as

$$\begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} f_1 + q_1 \\ f_2 \\ f_3 - q_3 \end{bmatrix} \quad (2.36)$$

This would be the problem which one would solve if this was a well posed problem. It turns out that this particular set of boundary values are not proper. Since the temperature is not fixed at any point in the bar, the temperature profile can slide up and down. An examination of the stiffness matrix shows that its determinant is zero (i.e., the system is singular).

If θ_1 and θ_3 are specified, then the middle equation can be used to find θ_2, and the first and last equation are used to find the heat flux at the two ends. The, mixed type boundary conditions can also be solved in a similar manner.

Before we proceed it is important to state that in practice the boundary conditions are not imposed as was presented above. Theoretically, if we have known values of temperature at the boundary we should eliminated these known values from the set of unknowns and introduce the unknown heat fluxes in the boundary term into the list of unknowns. To avoid this complex procedure, the penalty method is used for imposing the temperature boundary. This method modifies the stiffness matrix and load vector to fix the value of temperature to a given value at a boundary node. The method is based on the fact that for any boundary node i there will be an equation

$$K_{ii}\theta_i + otherterms = f_i. \tag{2.37}$$

To impose the condition $\theta_i = \hat{\theta}$. one can add a term $pK_{ii}\theta_i$ to the left hand side of the equation and the term $pK_{ii}\hat{\theta}$ to the right hand side of the equation to get

$$K_{ii}\theta_i + pK_{ii}\theta_i + otherterms = f_i + pK_{ii}\hat{\theta} \tag{2.38}$$

where p is a large number. After dividing by pK_{ii} one gets

$$\frac{\theta_i}{p} + \theta_i + \frac{otherterms}{pK_{ii}} = \frac{f_i}{pK_{ii}} + \hat{\theta}$$

All term in this equation become small accept the two newly added ones. Therefore, the equation will become dominated by the equation $\theta_i = \hat{\theta}$ and the other terms become unimportant. Since the method eliminates the importance of any other terms entering this equation, it will not be necessary to introduce the unknown heat flux into the list of unknowns.

In short, to impose a temperature boundary condition at global node i one can multiply K_{ii} in the stiffness matrix by $(1+p)$ (i.e., add pK_{ii} to K_{ii}), and add $pK_{ii}\hat{\theta}$ to f_i. To impose a heat flux boundary condition at global node j one adds q to f_j, where q is a heat flux directed into the bar.

CHAPTER III

XFEM AND LEVEL SET

3.1 Extended Finite Element Method

This section reviews the basics of the eXtended Finite Element Method applied to crack growth problems. The eXtended Finite Element Method is a Partition of Unity Method. Melenk and Babuška [22] and Babuška and Melenk [1] treat in detail the mathematical foundations of Partition of Unity Finite Element Methods.

The difficulty in modelling cracks using the standard finite element method is that the discretization (mesh) has to conform to the crack faces in order to model the discontinuity associated with the presence of the crack. Here again, the notion of local approximation can be exploited. By adding a discontinuous function across the crack faces to the displacement approximation space, it enable the finite element method to model the discontinuity without needing to conform the discretization to the discontinuity. This incorporation of the asymptotic fields of linear elastic fracture mechanics as well as a discontinuous function in a standard finite element approximation in this fashion was first used in Moes et al. [23] and the associated finite element method was coined the eXtended Finite Element Method (X-FEM). Belytschko et al. [5] explain how the X-FEM may be used to introduce arbitrary discontinuities in finite elements. Since its introduction, the eXtended Finite Element Method has been used for a variety of fracture mechanics problems including non-planar crack growth in three dimensions, cohesive crack growth, and dynamic crack growth.

3.1.1 Extended finite element approximation

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

assumes the form

$$u^h(x) = \sum_{I, n_I \in \mathcal{N}_e} N_I(x) u_I + \sum_{J, n_J \in \mathcal{N}^g} N_J(x) \mathcal{E}(x) a_J \tag{3.1}$$

where the nodal set $\mathcal{N}^g$ is the set of nodes whose support is intersected by the domain Ω_g associated with a geometric entity such as a hole, crack surface, or crack front and $\mathcal{N}$ is the set of nodes that are not enriched. Mathematically,

$$\mathcal{N}^g = \{n_J : n_J \in \mathcal{N}_e | \omega_J \cap \Omega_g \neq 0\} \tag{3.2}$$

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

As an alternative to the approximation (3.1) above, the approximation

$$u^h(x) = \sum_{I, n_I \in \mathcal{N}} N_I(x) u_I + \sum_{J, n_J \in \mathcal{N}^g} N_J(x)(\mathcal{E}(x) - \mathcal{E}(x_J)) a_J \tag{3.3}$$

provides the enriched displacement function u^h with the nice property that u_I are the nodal displacements.

3.1.2 Discretized equilibrium equations

In the eXtended Finite Element Method, the additional unknowns a_J associated with the enrichment functions simply augment the conventional unknown displacement vector u and are solved for in the same manner:

$$K.u = f^{ext} \Longleftrightarrow \begin{bmatrix} K_{uu} & K_{ua} \\ K_{au} & K_{aa} \end{bmatrix} \begin{bmatrix} u \\ a \end{bmatrix} = \begin{bmatrix} f_u^{ext} \\ f_a^{ext} \end{bmatrix} \tag{3.4}$$

Note that the extended finite element equations are identical to the standard finite element equations with the additional unknowns a and the additional enrichment arrays K_{aa} and $K_{ua} = K_{au}$. The size of the stiffness matrix is $N_{total} = N_e + N_{enr}$, it is symmetric, positive definite, sparse and banded - since enrichment is local.

The implementation of the eXtended Finite Element Method is thus very closely related to that of the finite element method. Using the definition of the bilinear form for linear elasticity,

$$\begin{cases} \mathcal{B} : \mathcal{H} \times \mathcal{Y} \to \Re \\ (u, v) \mapsto \mathcal{B}(u, v) = \int_\Omega (\nabla_s u) C (\nabla_s v) d\Omega \end{cases} \tag{3.5}$$

where $\mathcal{H}$ and $\mathcal{Y}$ are Hilbert spaces of square integrable functions and $\nabla_s = \frac{1}{2}(\nabla + \nabla^T)$ is the symmetric gradient operator, C is the fourth order elasticity tensor. The stiffness matrix K can also be written

$$\forall I, J \in \|1, \cdots, N_{total}] : K_{IJ} = \mathcal{B}(\bar{N}_I, \bar{N}_J) = \int_\Omega (\nabla \bar{N}_I) C (\nabla \bar{N}_J) d\Omega = \int_\Omega \bar{B}_I C \bar{B}_J d\Omega \tag{3.6}$$

where, for an element e, with N_e standard degrees of freedom and total number of degrees of freedom $N_{total} = N_e + N_{enr}$, the shape function vector $\bar{N}$ is the enriched version of its finite element counterpart $N_{FEM} = N_I$. In the following, the discussion is restricted to three-node triangular elements to fix ideas but is easily generalized to other linear and higher-order elements. Assuming all the three nodes in the element are enriched with the local enrichment function E, and letting N_I be their associated shape functions, $\bar{N}$ can be written

$$\bar{N} = [N_{FEM}, N_{ENRICHED}] = [[N_1, N_2, N_3], [\mathcal{E}N_1, \mathcal{E}N_2, \mathcal{E}N_3]] \tag{3.7}$$

where the shape functions N_I and the enrichment function, $\mathcal{E}$, are functions of the point $x = (x, y) \in \Re^2$ in the plane at which they are evaluated. The discretized gradient of the extended shape functions is written

$$\bar{B} = [[B_{FEM}], [B_{ENRICHED}]]$$

$$\bar{B} = \begin{bmatrix} & (\mathcal{E}N_1)_{,x} & 0 & (\mathcal{E}N_2)_{,x} & 0 & (\mathcal{E}N_3)_{,x} & 0 \\ B_{FEM} & 0 & (\mathcal{E}N_1)_{,y} & 0 & (\mathcal{E}N_2)_{,y} & 0 & (\mathcal{E}N_3)_{,y} \\ & (\mathcal{E}N_1)_{,y} & (\mathcal{E}N_1)_{,x} & (\mathcal{E}N_2)_{,y} & (\mathcal{E}N_2)_{,x} & (\mathcal{E}N_3)_{,y} & (\mathcal{E}N_3)_{,x} \end{bmatrix} \tag{3.8}$$

The mentioned before definition of the discretized gradient operator $\bar{B}$ is easily extended to the case where more than one enrichment function is used for each

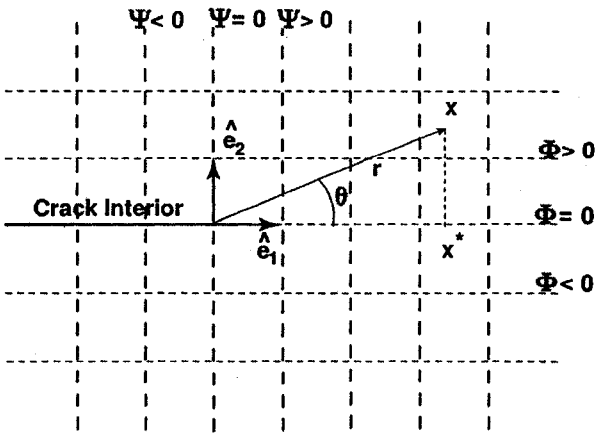
node in the element. Also note that for an element which is not enriched, $\bar{B}$ degenerates to the standard finite element discretized gradient operator B_{FEM}.

3.1.3 Level set representation of a crack in 2D

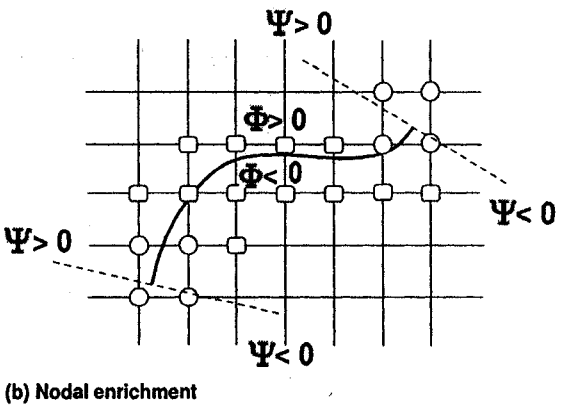
For more information about Level Set, see next section.

Consider a single crack in two dimensions, and let Γ_c be the crack interior (crack faces) and $(x_i)_{1\leq i\leq 2}$ the crack tips. First, the definition of the signed-distance function ϕ to the curve Γ_c

$$\phi(x) = \begin{cases} \min_{x_c\in\Gamma_c} \|x - x_c\| = d(x, x^*), & if (x - x^*).\hat{e}_2 \geq 0 \\ -\min_{x_c\in\Gamma_c} \|x - x_c\| = -d(x, x^*), & otherwise \end{cases} \tag{3.9}$$



(a) Definition of the two level set functions representing the crack in 2D



(b) Nodal enrichment

Figure 3.1: Nodal enrichment for a crack in 2D defined by two level set functions

where $d(x, x^*)$ is the usual Euclidian distance on $\Re^2$, x^* the closest point to x on the crack Γ_c and x_c a point on Γ_c. The function ϕ is schematically represented

in Figure 3.1.

While the function ϕ suffices to describe a closed or unbounded contour in a two-dimensional space, additional information is required to describe the geometry of an open segment in a two-dimensional space (Stolarska et al., [31]). To describe the location of each crack tip i additional functions ψ_i are introduced. These functions are defined by the signed distance function to the line going through the tip and normal to the crack, as shown in Figure 3.1. Mathematically:

$$\psi_i(x) = (x - x_i).\hat{e}_1 \tag{3.10}$$

To avoid using a separate function ψ_i for each tip, it is convenient to use the function ψ defined by $\psi = max_i\psi_i$ (Stolarska et al., [31]). Figure 3.1 illustrates the use of both level set functions ϕ and ψ to define the geometry of a line crack in 2D.

The crack is the part of the zero level set of the signed-distance to the crack, ϕ, for which $\psi \leq 0$. The crack therefore is the set of points

$$\Gamma_c = \{x \in \Re^d | \phi(x) = 0 \text{ and } \psi(x) \leq 0\} \tag{3.11}$$

3.1.4 Definition and selection of the enriched nodes

The crack is modelled by enriching the nodes whose nodal shape function support intersects the interior of the crack by the discontinuous function H. The nodes whose nodal shape function support contains the crack tips are enriched by the two-dimensional asymptotic crack-tip fields. The level set description of cracks permits a natural selection of the enriched nodes. Also, the values of the functions ϕ and ψ are computed at the nodes of the fixed mesh (the same throughout the crack growth) used to solve the elliptic problem for the crack evolution. To determine the location of a point x relative to the crack, it is sufficient to know the value of ϕ at that point. If $\phi(x) < 0$, x is below the crack, if $\phi(x) > 0$, x is above the crack.

Similarly, due to the orthogonal nature of the zero level sets of ϕ and ψ at the crack tips, the computation of the branch functions, present in the asymptotic

enrichment, in the domain is simplified. As shown in Figure 3.1, the two level set functions ϕ and ψ allow the construction of a natural local coordinate system, centered at the crack tip. Consequently, the values of r and θ needed for the computation of the near-tip fields can be simply obtained by

$$r(x) = \sqrt{\phi^2(x) + \psi^2(x)}, \theta(x) = arctan\frac{\phi(x)}{\psi(x)} \tag{3.12}$$

The selection of enriched nodes is also simplified by the use of the level set functions ϕ and ψ. Let ϕ_{min} and ϕ_{max} (resp. ψ_{min} and ψ_{max}) be the minimum and maximum values of ϕ (resp ψ) at the nodes of a given element. For an element to contain the crack tip, it is necessary and sufficient that $\psi_{min} \times \psi_{max} \leq 0$ and $\phi_{min} \times \phi_{max} \leq 0$. Also, an element is completely slit by a crack if and only if the function ϕ takes both positive and negative values at its nodes ($\phi_{min} \times \phi_{max} \leq 0$) and the function ψ is negative ($\psi \leq 0$) at all of its nodes. This is illustrated in Figure 3.1.

3.1.5 Enrichment functions: Discontinuous interior enrichment

The displacement discontinuities in the interior of a line crack are modelled by an enrichment function based on the function H, which will be referred to as a generalized Heaviside function. The function H takes the value +1 above the crack and -1 below the crack. More precisely, let x^* be the closest point to x on the crack Γ_c, and $\hat{e}_2$ be the normal to the crack at x^* as shown in Figure 3.1. First, recall the definition of the signed-distance function ϕ to a curve Γ_c

$$\phi(x) = \begin{cases} \min_{x_c \in \Gamma_c} \|x - x_c\| = d(x, x^*) & if (x - x^*).\hat{e}_2 \geq 0 \\ -\min_{x_c \in \Gamma_c} \|x - x_c\| = -d(x, x^*) & otherwise \end{cases} \tag{3.13}$$

where $d(x, x^*)$ is the usual Euclidian distance on $\Re^2$, x^* the closest point to x on the crack Γ_c and x_c a point on Γ_c. The function ϕ is schematically represented in Figure 3.1. The function H is then given by +1 if $(x - x^*).\hat{e}_2 \geq 0$, -1 otherwise, i.e.,

$$H(x) = sign(\phi(x)) = \begin{cases} 1 & if (x - x^*).n(x^*) \geq 0 \\ -1 & otherwise \end{cases} \tag{3.14}$$

Therefore, the computation of the discontinuous enrichment function H at a point x in the domain reduces to

$$H(x) = sign(\phi(x)) = \begin{cases} 1 & if \phi(x) \geq 0 \\ -1 & otherwise \end{cases} \tag{3.15}$$

3.1.6 Enrichment functions: Asymptotic near-tip enrichment

To model the crack front and also to improve the representation of crack-tip fields in three-dimensional computations, crack-tip enrichment functions are used in elements which contain a crack tip. The enrichment consists of functions which incorporate the radial and angular behavior of the two-dimensional asymptotic crack-tip displacement field

$$B(x) \equiv \{B_1, B_2, B_3, B_4\}(x) = [\sqrt{r}cos\frac{\theta}{2}, \sqrt{r}sin\frac{\theta}{2}, \sqrt{r}sin\frac{\theta}{2}sin\theta, \sqrt{r}cos\frac{\theta}{2}sin\theta] \tag{3.16}$$

where r and θ are polar coordinates in the plane. Note that the second function $B_2(r,\theta) = \sqrt{r}sin\frac{\theta}{2}$ is discontinuous at the crack interior, which induces the required discontinuity in the displacement field in the tip elements.

Remark : Enrichment may be seen as adding extra degrees of freedom to the enriched nodes. An enriched node gains one extra degree of freedom per enrichment function per dimension. In 2D, a node for which the displacement fields are enriched with the four crack tip asymptotic enrichment functions, for instance, has two conventional degrees of freedom (its displacements in both directions) and $2 \times 4 = 8$ enrichment degrees of freedom.

3.1.7 Summary of the algorithm

Let the crack be described at each increment in crack advance n by the two level set functions $\phi^{(n)}$ and $\psi^{(n)}$ presented above and let $x_i^{(n)}$ be the set containing the tips of this crack at the increment n. The crack growth rate may be interpreted as a speed function F. This speed function may then be used to update the level sets $\phi^{(n)}$ and $\psi^{(n)}$ to $\phi^{(n+1)}$ and $\psi^{(n+1)}$ respectively. Note that those level set functions need only be updated ahead of the tips of the crack and remain unchanged in the rest

of the domain. The crack advance direction θ_c and the crack growth increment Δa are used to update $\psi_i^{(n)}$ to $\psi_i^{(n+1)}$ for each crack tip $x_i^{(n)}$ according to the hyperbolic conservation law of the same type as (3.20). Note that an hypothesis made above to derive this hyperbolic conservation law for the movement of the hypersurface was that the front (here the level set $\psi^{(n)}$) was propagating perpendicularly to itself so that it is not $\psi_i^{(n)}$ that obeys this conservation law but rather the adequately rotated level set function $\tilde{\psi}_i^{(n)} = R(\psi_i^{(n)})$, such that $\tilde{\psi}_i^{(n)} \perp F$. By construction of ψ, $\|\nabla\psi\| \equiv 1$ and $\|F\| = \Delta a$. Once all the $\psi_i^{(n+1)}$ are computed for all tips, the value of $\psi^{(n+1)}$ at all nodes in the level set update domain may be deduced using the relation $\psi^{(n+1)} = \max_i \psi_i^{(n+1)}$.

The crack propagation direction θ_c is also used to update the level set function $\theta^{(n)}$. This "update" of $\theta^{(n)}$ is really a re-initialization, which ensures that $\theta^{(n+1)}$ is a signed-distance function for the updated interface $\Gamma(n+1)$ as $\theta^{(n)}$ was for $\Gamma(n)$. Once the crack propagation direction is known, $\theta^{(n+1)}$ may be computed exactly at any point x in the level set update region, by computing the signed distance to the new interface:

$$\phi^{(n+1)}_{updateregion}(x) = \pm\|(x - x_i^{(n)}) \times \frac{F}{\|F\|}\| \tag{3.17}$$

The new position of the crack tip is then obtained by finding the intersection of the updated level sets $\phi^{(n+1)}$ and $\psi^{(n+1)}$. Figure 3.2 summarizes the algorithm for both the 2D case and the 3D case.

3.2 Level Set

The level set and fast marching methods were first developed by Sethian [29] to track interfaces moving with curvature-dependent speed. In this method, the interface is represented as the zero level set of a function ϕ of one higher dimension than the dimension of the interface, i.e., ϕ is a function of position and time. Therefore, the interface is a hypersurface of the space in which the function ϕ takes its values. As the interface evolves, it always coincides with the zero-level set of the function ϕ, whose values are determined by solving an initial value

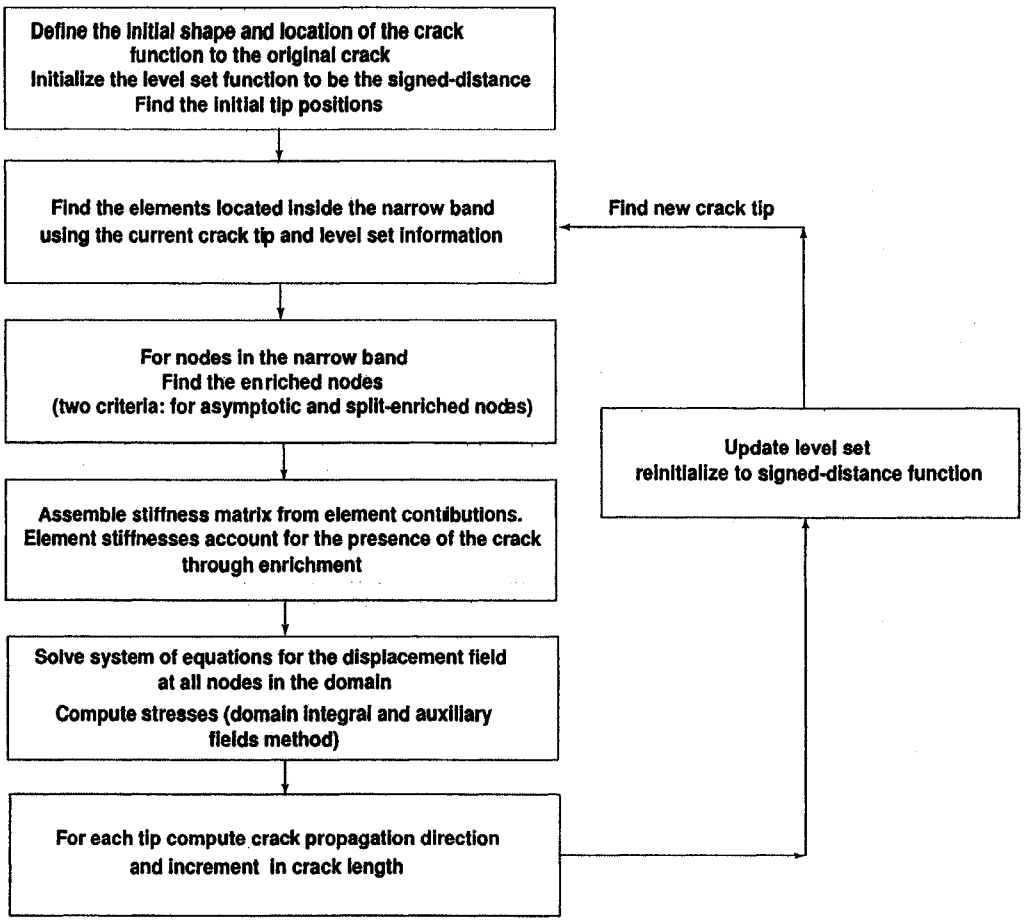


Figure 3.2: X-FEM/LS crack growth algorithm in 2D

partial differential equation in one higher dimension than the interface dimension. This allows sharp corners, merging and changes in topology in the interface (the zero-level set of ϕ) to be naturally and seamlessly accounted for.

The level set function is typically approximated on a fixed mesh, which avoids the additional weight of mesh regeneration or mesh motion tracking. In addition, this allows coupling the level set method with finite element and eXtended Finite Element Methods naturally since the level set update may then use the same mesh that is used for the finite element calculations. Basic geometric properties of the interface (normal vector, curvature) are easily determined from the level set function.

3.3 Definition of Level Set Method

To set the stage for the description of level sets in the fracture problems, the basics of the level set method for the particular case of a curve evolving in 2D are now recalled. Let $\Gamma(t = 0)$ be a closed, non-intersecting curve in $\Re^2$ moving at constant speed F perpendicularly to itself.

Let ϕ be a scalar function such that $\Gamma(t)$ is the zero-level set of ϕ at all positive times t, that is:

$$\forall t \in \Re_+, \Gamma(t) = \{x \in \Re^2 | \phi(x(t), t) = 0\} \tag{3.18}$$

At the initial time $t = 0$, the scalar function ϕ is set to be the signed-distance function to the curve $\Gamma(0)$. It is also assumed that each level set of ϕ flows along its gradient field with the constant speed F. Consider the motion of the level set of ϕ such that $\phi(x(t), t) = C$. Note that $x(t)$ is the route of any particle located on this level set curve. By definition of F (normal velocity), the particle speed $\frac{\partial x(t)}{\partial t}$ in the direction of the outward normal $n_\phi = -\frac{\nabla_\phi}{\|\nabla_\phi\|}$ is equal to the speed function F:

$$(\frac{\partial x(t)}{\partial t})(-\frac{\nabla_\phi}{\|\nabla_\phi\|}) = F \tag{3.19}$$

Using the chain rule on the term $\nabla\phi$, the initial-value partial differential equation governing the evolution of the scalar function ϕ becomes:

$$\begin{cases} \frac{\partial \phi}{\partial t} + F\|\nabla\phi\| = 0, \\ \phi(x(0), 0) = \pm \min_{x_\Gamma \in \Gamma} \|x - x_\Gamma\|, \end{cases} \tag{3.20}$$

The curve $\Gamma(t)$ at time t is then given by solving equation (3.20) for the unknown function ϕ.

3.4 Numerical Integration

There are two main reasons that make numerical integration of X-FEM approximations more involved than the Standard Finite Element Method integration procedures. Due to the presence of a discontinuous function in the approximation, it is important to be considerate the numerical integration of the extended

finite element equations. For elements in which discontinuous enrichment is used, a modified Gaussian quadrature scheme based on sub-elements aligns with the discontinuity is employed. This avoids inaccuracies and ill-conditioned systems associated with Gaussian Integration of discontinuous functions. This principle is summarized, for the 2D case of a rectangular element in Figure 3.3 [16]. The dashed line is the line of discontinuity. Note that no additional degree of freedom is added to the system. The partitioning is only used for numerical integration.

Remark : The triangular elements in Figure 3.3 have no degree of freedom associated with them. They exist only for the purpose of numerical integration.

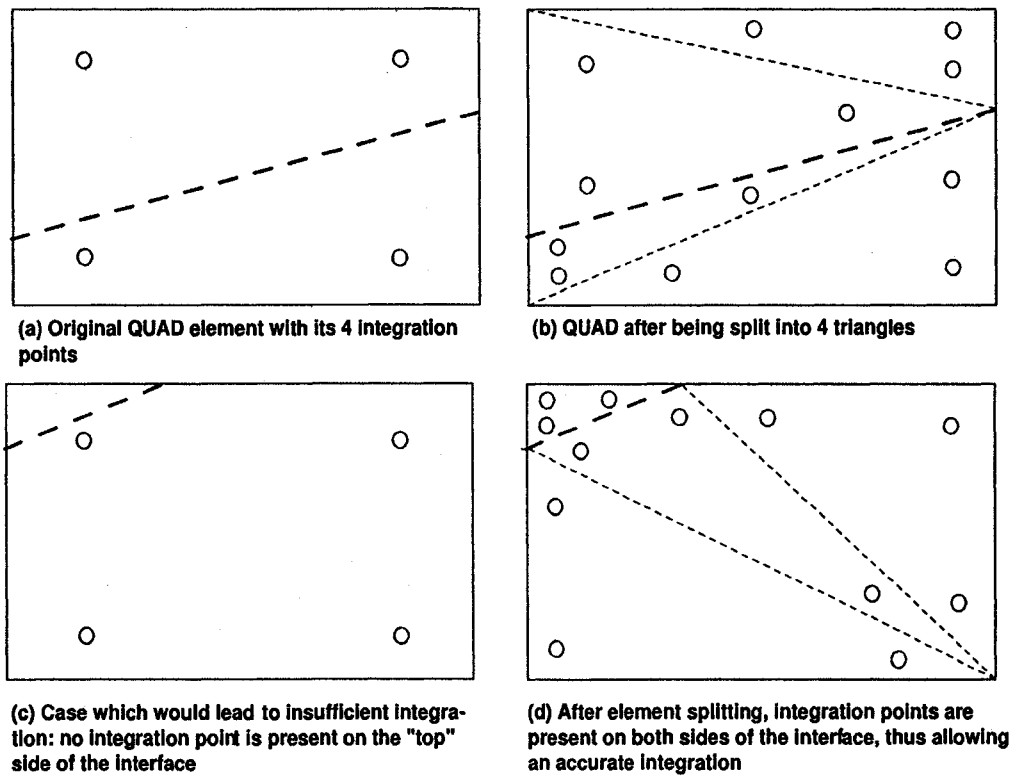


Figure 3.3: Element partitioning for X-FEM integration. The interface is denoted by a heavy dashed line, quadrature points by circles and the triangular elements resulting from the split in light dotted lines. Note that the subelements are only used for integration purposes and that no additional dof is associated with them [16].

3.5 C++ Implementation

C++ is an example of an object-oriented programming language and is the language in which the X-FEM code is written. As opposed to procedural programming, the object-oriented programming philosophy segments the problem at hand into objects and defines the necessary data that those objects should hold and the functions they should be able to perform, on themselves and on other objects. This produces a set of application-specific data types used in writing the code. The rest of the programming resides in precisely designing and implementing, for each of the data types (or objects), the operations that can be carried out with each object type. Then, the logic of the program in terms of those objects and the kinds of operations they allow should be set up. These concepts should make a well-written object-oriented program easier to understand and easier to evolve. In C++, as in any other object oriented programming language, a program is written in terms of objects in the domain of the problem that is to be solved. A large part of the programming process resides in deciding which objects are useful, what data they should hold, which actions they should perform, and how they should interact with each other. For example, in the extended finite element framework, there are objects, which would have to "know" its type (triangular, quadrilateral), its dimension, the coordinates of its nodes, the type of integration used and the coordinates of the integration points, the type of interpolation functions defined on the element, which of its nodes are enriched, etc.. A finite element object should also be able to perform some operations; it would therefore contain functions to operate on it such as: compute its area or volume, its Jacobian matrix, get its stiffness matrix or mass matrix, etc.. This packaging of data values and functions within an object is referred to as encapsulation and is one of the corner stones of object-oriented programming.

We used the "getfem++" library to modify the C++ codes to solve the problem. "getfem++" is a free Generic Finite Element library in C++ language [26]. We modified C++ codes for using crack propagation with XFEM/LS method.

The documentation of final program in C++ language is shown in Appendix B.

3.6 Mesh Generation

For mesh generation we used the GiD software. GiD is an interactive graphical user interface used for the definition, preparation and visualization of all the data related to a numerical simulation. This data includes the definition of the geometry, materials, conditions, solution information and other parameters. The program can also generate a mesh for finite element, finite volume or finite difference analysis and write the information for a numerical simulation program in its desired format [27].

CHAPTER IV

NUMERICAL RESULTS

In this section three groups of numerical examples in two-dimensional and three dimensional elastostatics are presented to illustrate the accuracy of the XFEM for study fracture mechanisms in Aluminium-Silicon composite material. The first example we study crack growth in two dimensional plate with a different shape of inclusions under tension, and then study example of a delamination in bimaterial in two-dimensional and three dimensional. The last example is about crack growth in Aluminium-Silicon polycrystalline structure.

Remark: The materials models are elasticity and the criterion for crack opening is the fracture toughness of the Silicon or Aluminium. The fracture toughness for Silicon is $Kc = 0.95\ MPa.m^{1/2}$ and for Aluminium is $Kc = 0.28\ MPa.m^{1/2}$.

Note: We assume perfect interface boundary between Silicon particles and Aluminium matrix.

4.1 Configuration

The configuration for all tests includes:

- Geometry: 1x1 Square two dimensional geometry with nuclei crack in Y axes in point of Y=0 and X=0. Except for section 4.2.3 which is used three dimensional geometry. See Figure 4.1 and Figure 4.2.

- Mesh: Linear triangle elements with different element size from 0.05 to 0.025 (Size is given by the average side of the corresponding triangle or quadrilateral). Except for section 4.2.3 which is used quadrilateral elements. The numbers of the nodes are between 138 to 1832 and the numbers of the elements are between 222 to 3502. The program doesn't have capability to run with more than 4000 element. We should select the mesh densities, which are acceptable for the program. Mesh density is automatically created by GiD mesh generator software [27].

- Force: Traction force is used for all cases.
- Material: Aluminium - Silicon alloy, AA319 with Young's modulus 72.4 GPa and Poisson's ratio of 0.33. Young's modulus for Silicon is 107 GPa and Poisson's ratio is 0.278. Fracture toughness for Silicon particles is $Kc = 0.95\ MPa.m^{1/2}$ and for Aluminium is $Kc = 0.28\ MPa.m^{1/2}$.
- FEM Method: Lagrangian
- Boundary conditions: Top and bottom edges by uniform traction. One node on the bottom edge is pinned, the other is on a horizontal roller.
- Void and inclusion shape: circular, elliptical, square and triangle shapes are used for voids and only circular and elliptical are used for inclusion.
- Elliptical shape: With a/b=1.5, a/b=2 and a/b=2.5. The theta angle selected for elliptical shape are zero degree (orthogonal to the force), 45 degree (Figure 4.1) and 90 degree (parallel to the force). The angle is measured relative to bottom edge and "a", major axis radius in elliptical inclusion.

This configuration has been selected because we want to find the best shape and orientation of silicon particles in Aluminium matrix to find a better way for casting of these alloys. Therefore we select the simple geometry and simple mesh to apply force in Aluminium Silicon alloy. We used the mechanical properties of Aluminium Silicon alloy, such as Young modules and Poisson ratio and also fracture toughness for Silicon particles.

The tests are categorized in three groups.

1. Inclusions
2. Bimaterial
3. Polycrystal

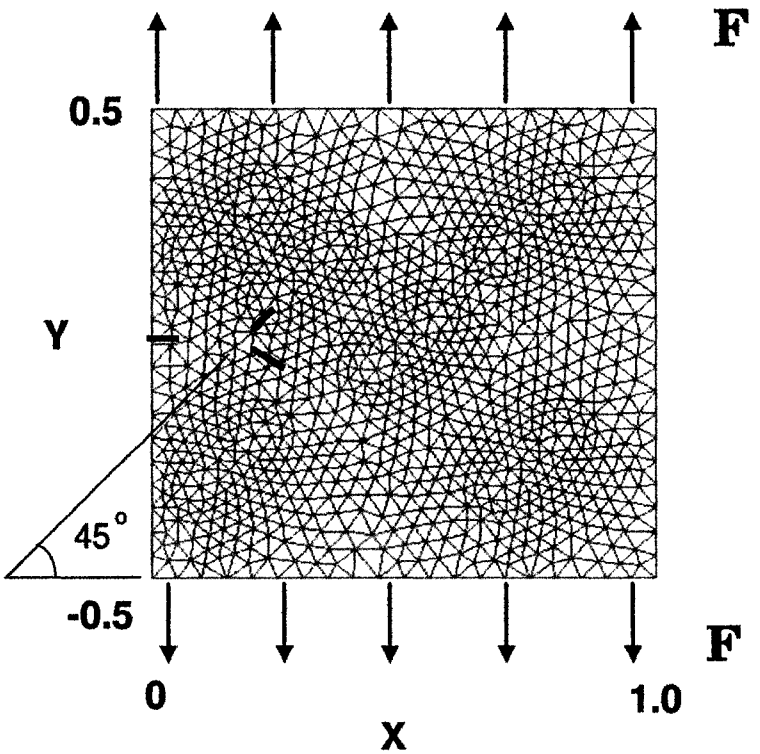


Figure 4.1: Geometry, force and crack nuclei

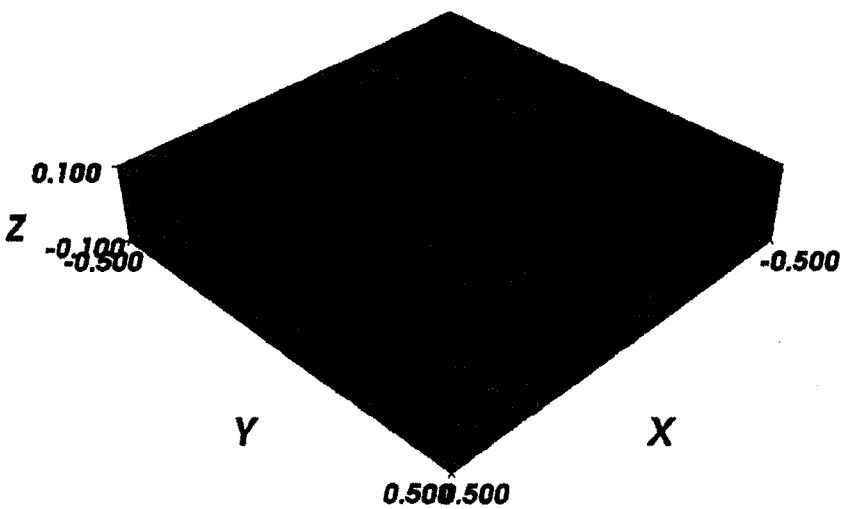


Figure 4.2: Geometry for delaminated crack

4.2 Aluminium Silicon Composite Materials

The XFEM model is applied to AA319 Aluminium Silicon alloy with a different number of Silicon inclusions and different orientations. This part has most important results, because we consider Silicon particles as inclusion in Aluminium matrix and it is close to reality. The initial crack nuclei are present in Figure 4.1. In all simulations, we consider a material with properties similar to AA319 alloy with a Young's modulus of 72.4 GPa and Poisson's ratio of 0.33. Young's modulus for Silicon is 107 GPa and Poisson's ratio is 0.278. We consider fracture toughness of Silicon particles, $K_c = 0.95MPa.m^{1/2}$. All calculations are done in plane strain considering a unit thickness and isothermal conditions. The AA319 plate is subjected to traction at its top and bottom edges. The displacement and the nominal traction are used to characterize the overall response of the body. In all of the simulations, we report nominal strain and nominal stress. Nominal strain defined by:

$$\varepsilon = (U - G)/L. \tag{1}$$

where U averaged edge displacement, and G is the gap (if exists). Nominal stress is magnitude of the traction vector.

Remark: We assume perfect interface boundary between Aluminium and Silicon particles.

We study two shapes of inclusions, circular and elliptical, because with statistical methods of these two shapes, we can find the solution for all shapes of inclusions in reality.

4.2.1 Single inclusion

Figure 4.3 shows a single edge crack nucleus in Aluminium matrix with a single circular inclusion. The monotonic loading facilitated the crack propagation within the homogenous Aluminium matrix and eventually in the Silicon circular inclusion (Figure 4.3a). The characteristic strain stress curve is shown in Figure 4.3b. It can be seen that after a linear response to the point where the crack reaches the

critical stress, the crack starts to grow and snapback occurs. Critical stress is maximum nominal stress when the crack starts to propagate.

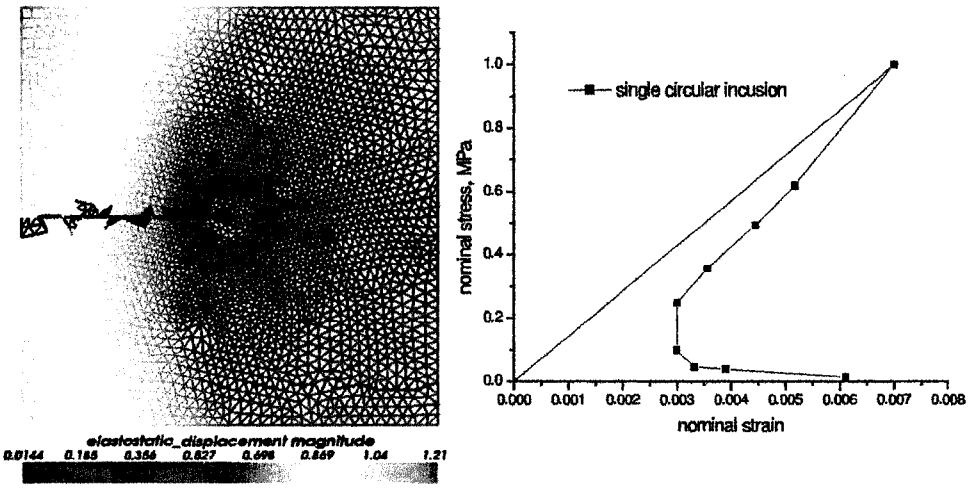


Figure 4.3: Single circular inclusion: a) crack propagation pattern b) stress-strain response.

Figure 4.4a shows crack propagation pattern in Aluminium matrix with a single elliptical inclusion with b=0.07 and a=0.14 oriented along x axis (a/b=2), and the characteristic strain stress curve is shown in Figure 4.4b. We can compare the two stress-strain respond diagrams for circular and elliptical inclusions as shown in Figure 4.5. As we can see the critical stresses between two diagrams almost the same, but the slope is different. The slope for elliptical inclusion is more than circular inclusion, and the elliptical inclusion has more nominal strain. We should note that the angle of elliptical inclusion is zero or it is orthogonal to the force in the stress strain diagram. If we change the angle of elliptical inclusion critical stress and slope will be change. Critical stress as function of the orientation of single elliptical inclusion is shown in Figure 4.6. As we can see we have the maximum critical stress when the elliptical inclusion is parallel to the force or has a 90 degree. If the a/b in ellipse is increased or it like needle, we even have better results. Figure 4.7 shows data for critical stresses as function of the shape of the inclusions. We choose different angles and different a/b factor to find which angle and which a/b factor has the best results. Then we can optimize casting process

to get proper shape and angle of Silicon particles.

We can conclude that the elliptical inclusion with long semi-axes parallel to the external force has a distinct advantage over any other orientations and shapes.

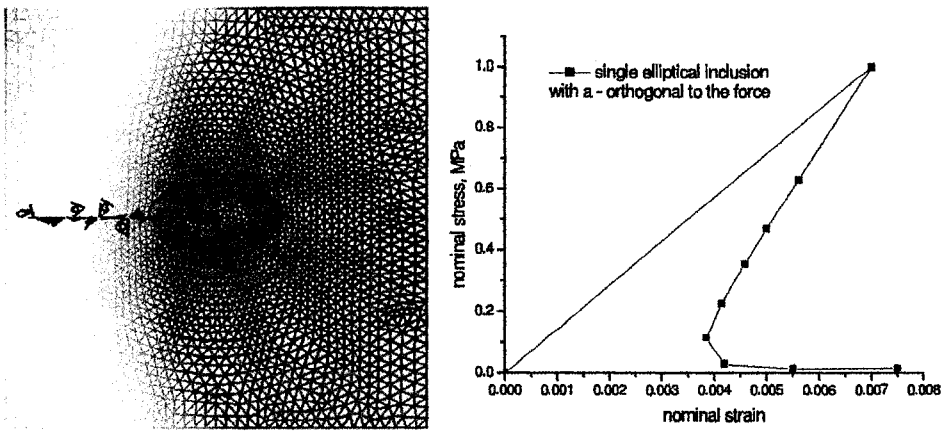


Figure 4.4: Single elliptical inclusion: a) crack propagation pattern b) stress-strain response.

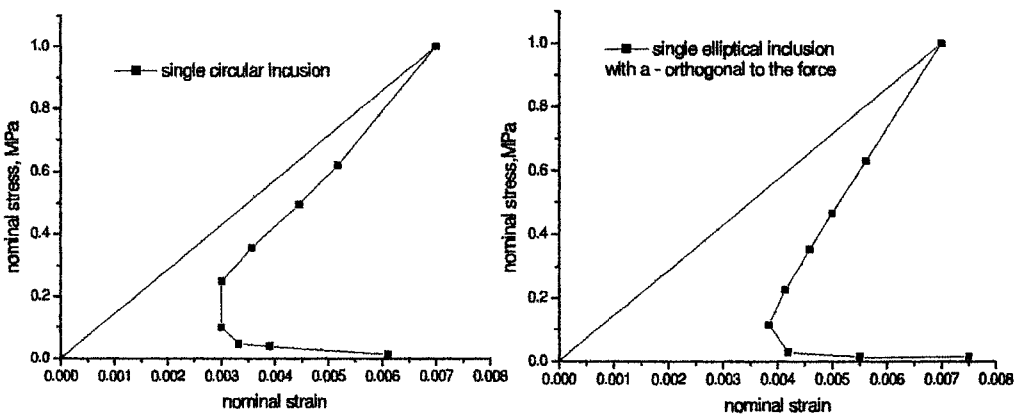


Figure 4.5: Stress-strain diagrams for single circular and elliptical inclusion

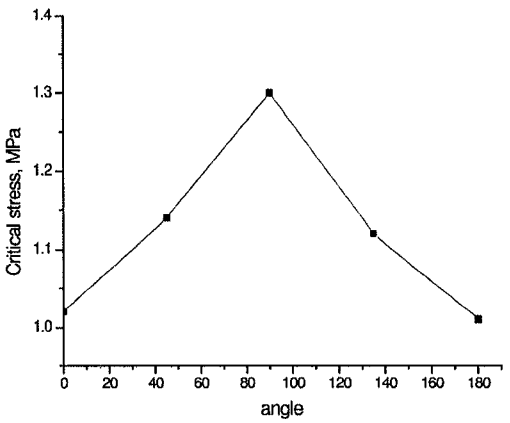


Figure 4.6: Critical stress as function of the orientation of single elliptical inclusion

shape	critical stress (MPa)
Circular	1.00
Elliptical (zero degree)	1.03
Elliptical (45 degree)	1.14
Elliptical (90 degree)	1.30

Figure 4.7: Critical stress as function of the shape of the inclusions

4.2.2 Multiple inclusions

The coordinate for multiple elliptical inclusion with parallel to the force shown in Figure 4.8. Multiple elliptical inclusion with a orthogonal to the force, parallel to the force and 45 degree to the force with stress-strain response shown in Figures 4.9, 4.10 and 4.11. The 45 angle is measured relative to bottom edge and "a", major axis radius in elliptical inclusion. All of elliptical inclusions has a/b=2 with b=0.07 and a=0.14. The reason for why we select these configuration is mention on last section.

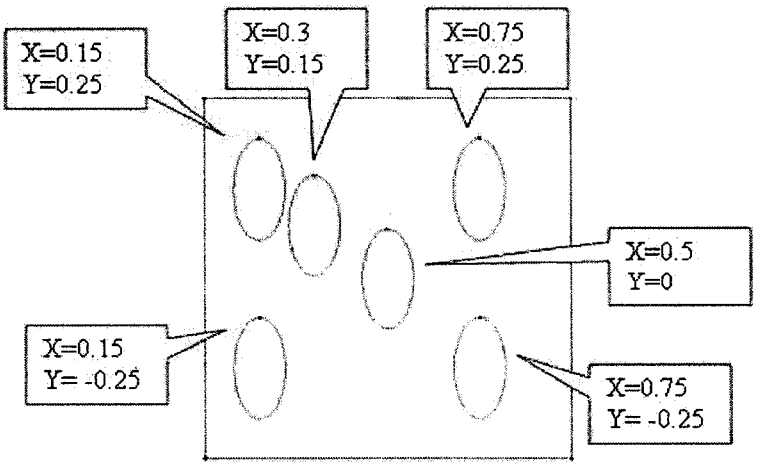


Figure 4.8: Coordinate of multiple elliptical inclusion parallel to the force

Figure 4.12a shows crack propagation pattern in multi-inclusion system with three crack nuclei. Similarly to the single inclusion case after a linear response a critical stress is reached and the edge crack starts to grow. However, at this moment the secondary cracks within the volume do not start their growth. Clearly, the advance of the edge crack initiates the growth of the secondary cracks. The instances of initiation of the secondary cracks can be seen in Figure 4.12b as specific kinks in the snapback portion of the strain stress curve.

We can compare the three stress-strain respond diagrams for elliptical inclusions with different orientation as shown in Figure 4.13. As we can see the critical

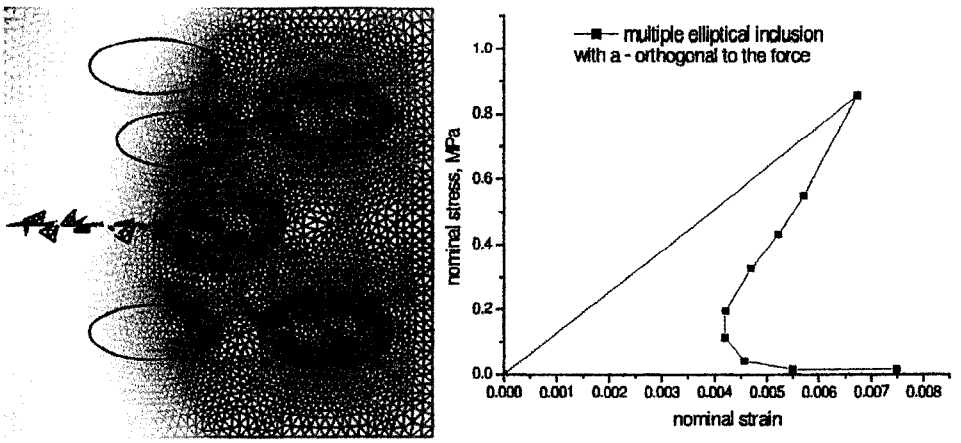


Figure 4.9: Multiple elliptical inclusions: a) crack propagation pattern b) stress-strain response.

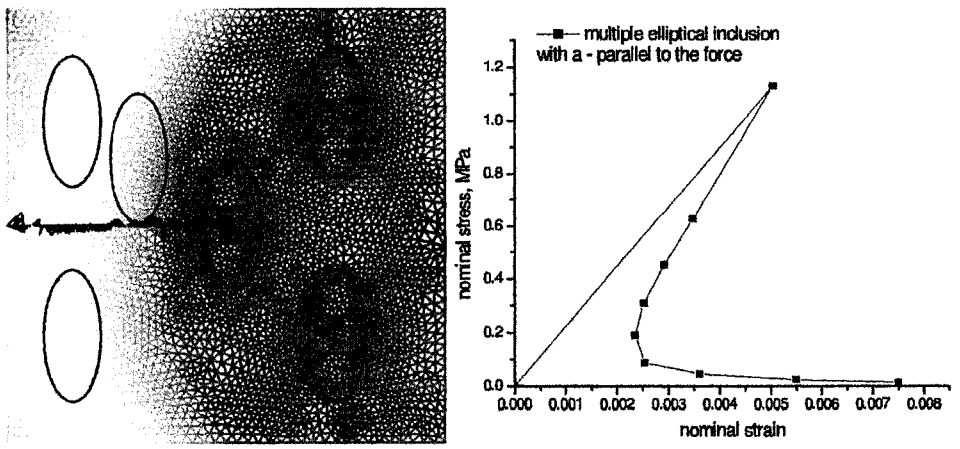


Figure 4.10: Multiple elliptical inclusions: a) crack propagation pattern b) stress-strain response.

stresses of multiple elliptical inclusion parallel to the force is highest amount. Critical stress as function of the orientation of multiple elliptical inclusion is shown in Figure 4.14. As we can see we have the maximum critical stress when the elliptical inclusion is parallel to the force or has a 90 degree. If the a/b in ellipse is increased or it like needle, we even have better results. Figure 4.15 has data for critical stresses as function of the orientation of the inclusions.

We can conclude that the multiple elliptical inclusions with long semi-axes parallel to the external force have a distinct advantage over any other orientations and shapes.

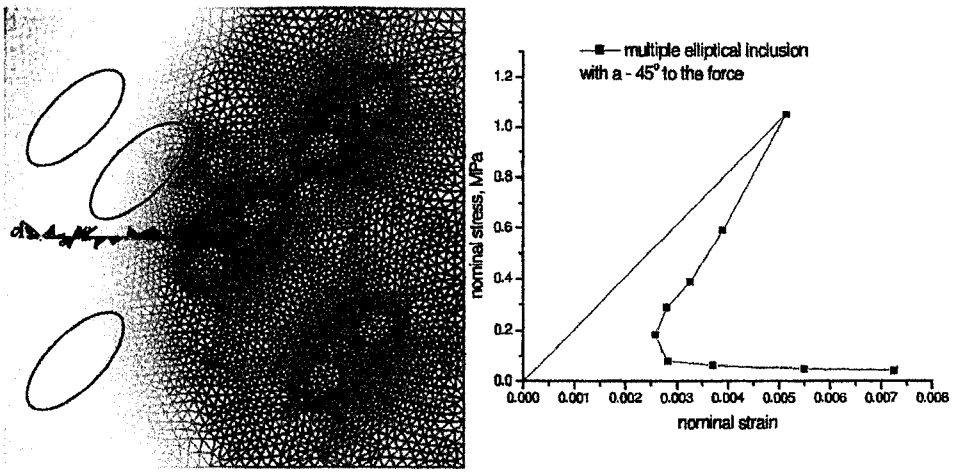


Figure 4.11: Multiple elliptical inclusions: a) crack propagation pattern b) stress-strain response.

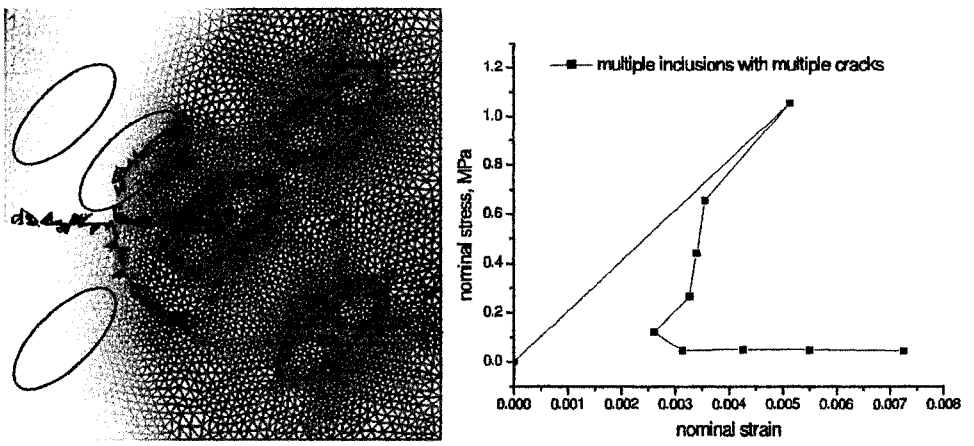


Figure 4.12: Multiple elliptical inclusions: a) multiple cracks propagation pattern b) stress-strain response.

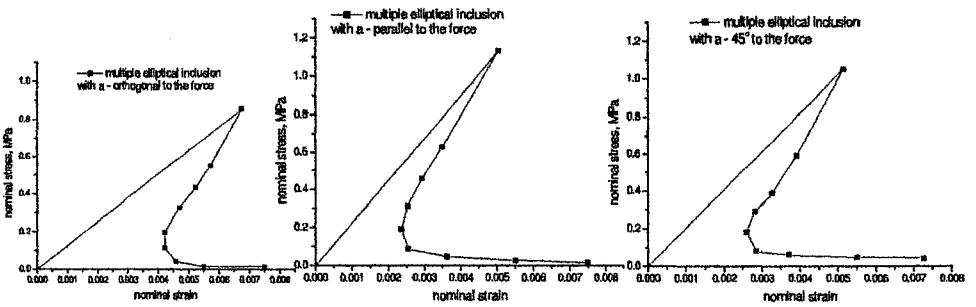


Figure 4.13: Compare critical stress diagrams to orientation of inclusions

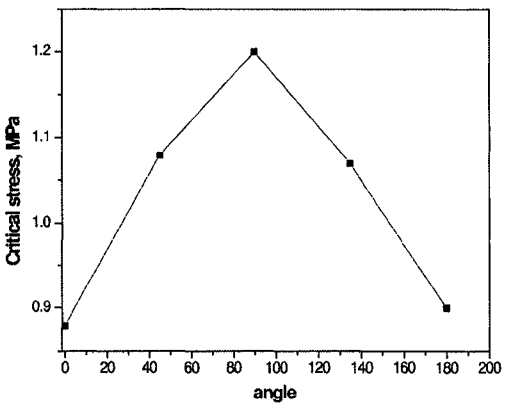


Figure 4.14: Critical stress as function of the orientation of multiple inclusion

Orientation (elliptical)	critical stress (MPa)
Orthogonal (zero degree)	0.86
Parallel (90 degree)	1.14
45 degree	1.05

Figure 4.15: Critical stress as function of the orientation of inclusions

4.2.3 Bimaterial-two dimensional delamination

Bimaterial has two different material that has own properties connected together. In our case one of them is Aluminium and another one is Silicon. We modified program parameters to meet the requirements for Al-Si bimaterial. The geometry, the nuclei of the crack and force is the same as other tests. Figure 4.16 shows the geometry and displacement for Al-Si bimaterial. In larger magnification we can see the meshes and the crack (Figure 4.17). Von Misses Stress magnitude for Al-Si bimaterial with single crack shown in Figure 4.18 and Figure 4.19 shows displacement for Al-Si bimaterial with multiple cracks.

Remarks: The units for displacement is "cm" and for stress is "MPa" in the following Figures.

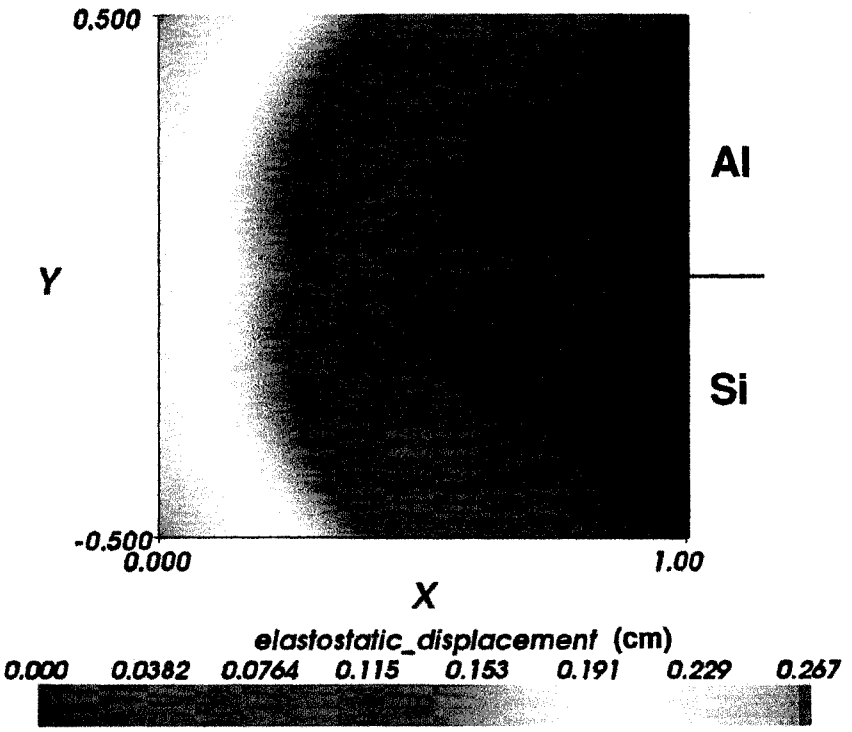


Figure 4.16: Al-Si Bimaterial displacement

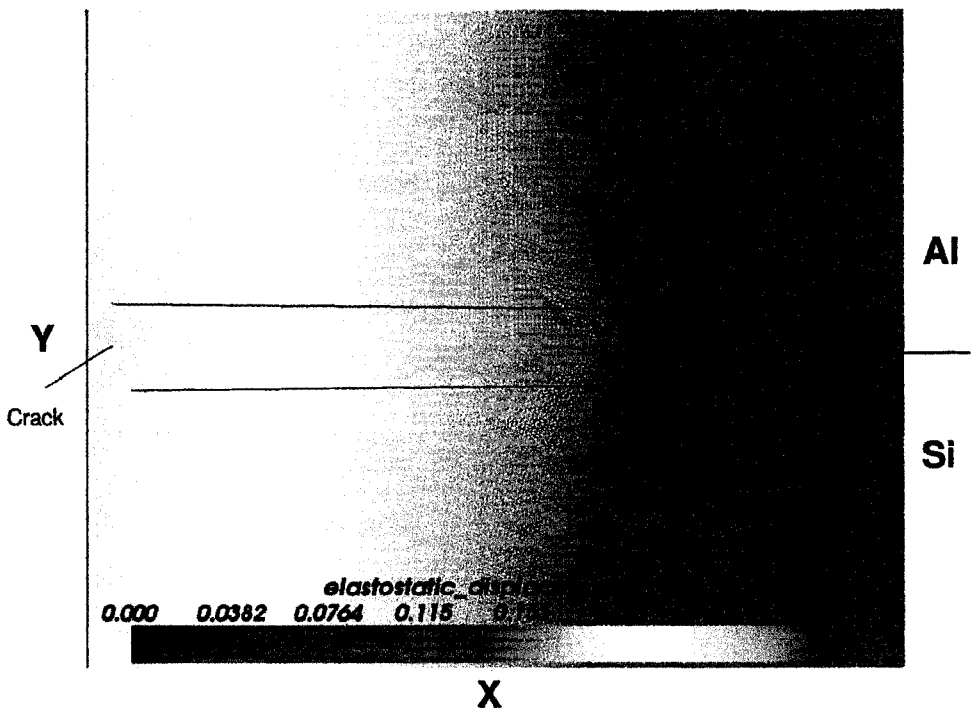


Figure 4.17: Mesh and crack in Al-Si Bimaterial

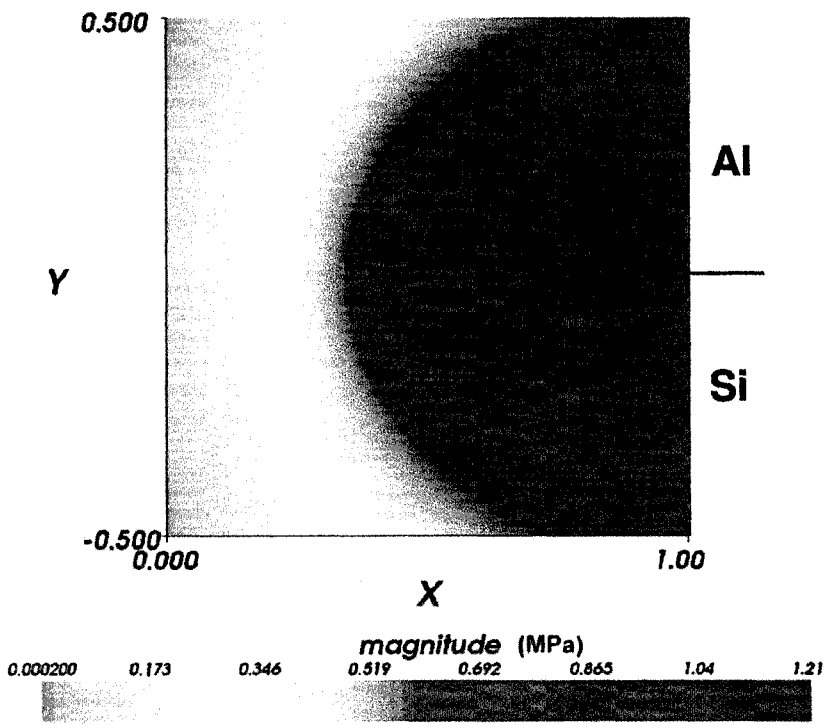


Figure 4.18: Von Misses Stress in Al-Si Bimaterial

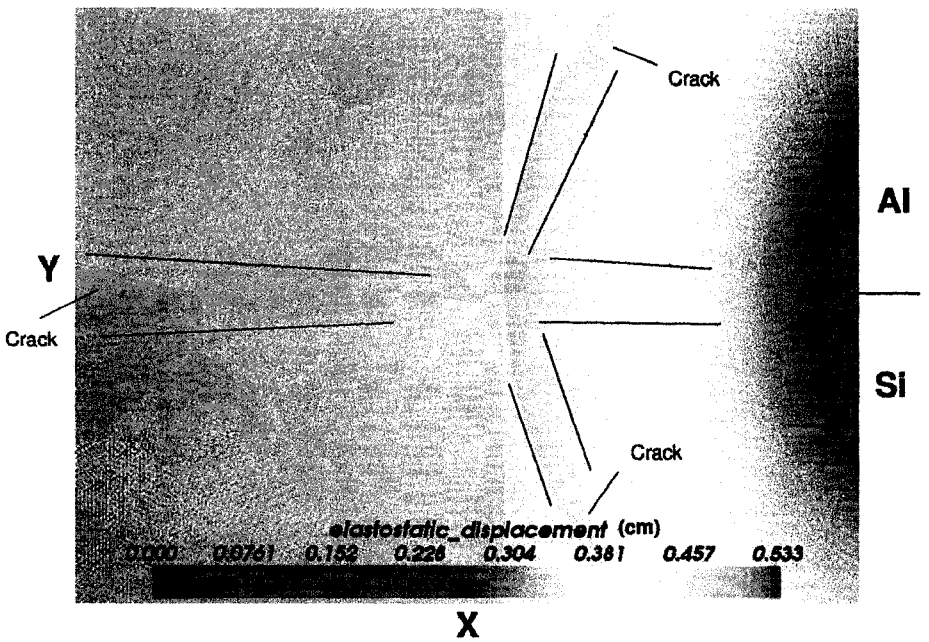


Figure 4.19: Displacement in Al-Si Bimaterial with multiple cracks

4.2.4 Bimaterial - three dimensional delamination

In this case we need 3D mesh to shows crack delaminate the plate. Due to geometry symmetry the quadrilateral elements were selected. Figure 4.20 shows 3D geometry for this test with mesh and crack and Figure 4.21 shows the mesh and crack in Y-Z plane. The displacement for delaminated crack test shows in Figure 4.22.

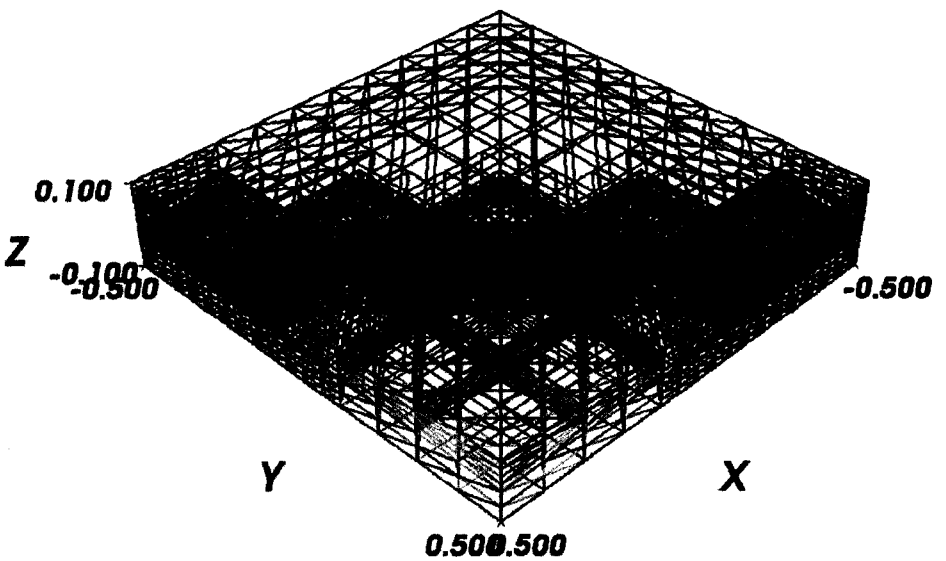


Figure 4.20: 3D geometry for delaminated crack test shows mesh and crack plane

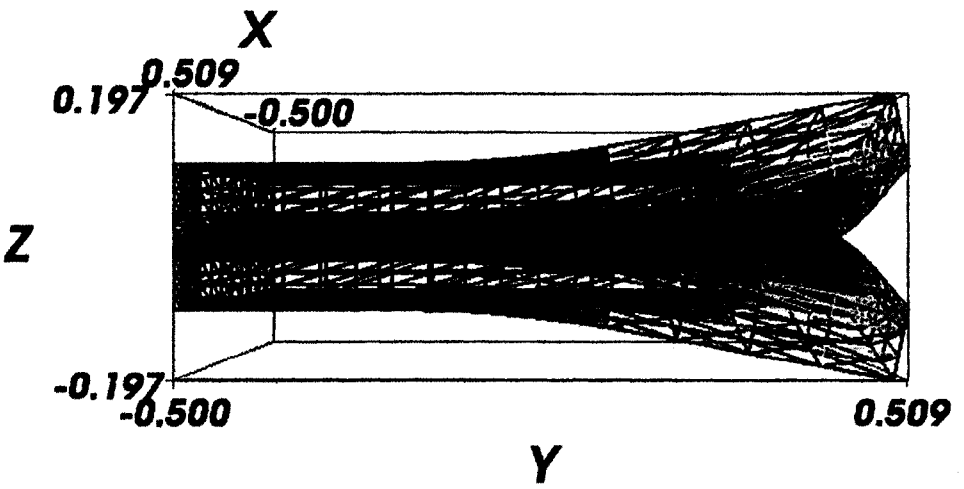


Figure 4.21: Mesh and crack in Y-Z plane

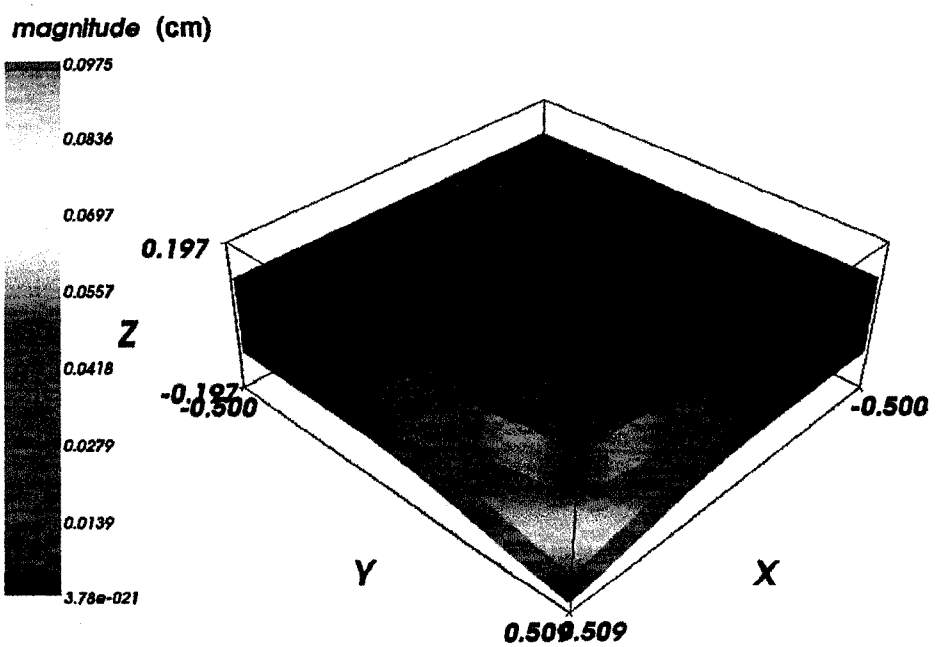


Figure 4.22: Displacement for delaminated crack test

4.2.5 Polycrystal

The methodology used to carry out crack propagation simulations through a polycrystalline microstructure consisted of main ingredients:

- Unlike in many existing simulations of brittle fracture, a continuum description of a polycrystal was employed. This was based on a realistic microstructure.
- The polycrystal is assumed to be elastically homogeneousall grains and grain boundaries have the same elastic constants (E and P).

We presented a numerical model for crack propagation through a realistic polycrystalline material microstructure. In the X-FEM, a discontinuous function and the two-dimensional asymptotic crack-tip displacement fields are added to the finite element approximation to account for the crack using the notion of partition of unity.

Figure 4.23 shows the geometry of polycrystal with grain boundaries. The mesh is 2D with linear triangle elements as shown in Figure 4.24. The size of element in Figure 4.24 is 0.025. We used two size of elements, 0.025 and 0.035. The Figure 4.25 shows the displacement for polycrystal when we add multiple cracks to the program parameters.

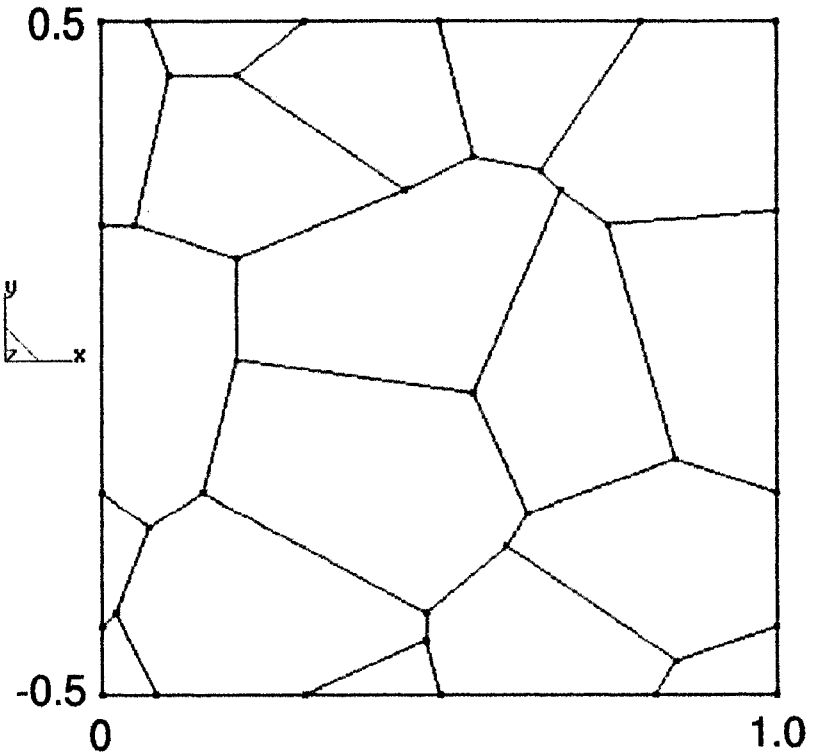


Figure 4.23: Polycrystal geometry

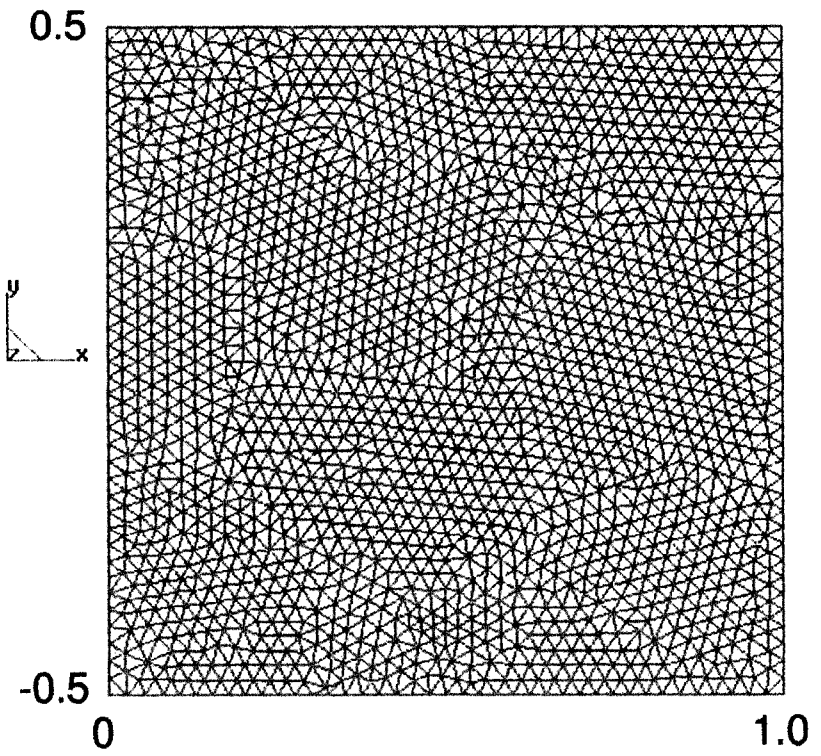


Figure 4.24: Polycrystal mesh

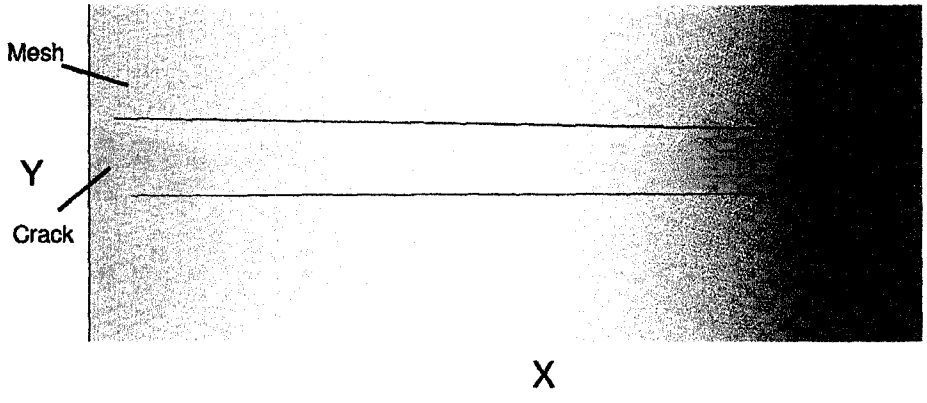


Figure 4.25: Mesh and crack in Polycrystal with a crack

CHAPTER V

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

A methodology to model arbitrary voids, inclusions, bimaterial, delamination in 2D and 3D in single crystal and polycrystalline composite materials has been suggested. The following items are summarized as conclusions:

- The shape of the inclusions along with the type of loading strongly influences the fracture of composite material.
- The needle like inclusions with long semi-axes parallel to the external force showed distinct advantage over any other orientations.
- In the case of multiple cracks the evolution of every single one affects the behavior of the composite material.
- No delamination along the matrix-inclusion interface has been observed, mainly due to the assumption of the continuous displacements at the matrix-inclusion interface.

5.2 Future Work

The approaches for the modelling of 2D fracture within Extended Finite Element are presented and discussed. However, there are several of additional subjects worth addressing in the future.

Perhaps the most important issue is comparison with experimental results. Academic examples are nice in the sense that particular effects may be clarified. Nevertheless, to actually develop models of any use outside the academic world, of course these models have to agree with real-world observations. An experimental comparison may lead to one of two things, either a validation of the proposed models or an indication of how to future refine the same, both equally valuable.

A challenging task and the ultimate goal is a generation to three dimensional analysis in order to simulate more complex geometries in real-world application.

So far, only tension forces is considered. Thus other forces such as shear force or combined forces would be of interest.

Scanning electron microscope (SEM) images of the fracture surface of a cast Aluminium Silicon alloy subjected to cyclic loading demonstrate that some of the pure silicon particles debonded. Therefore the program should be modified to consider matrix-interface delamination.

This program has potential of the XFEM as a computational fracture tool to study complex failure mechanisms in polycrystalline materials, but we should modify program to meet the requirements, such as add some grains as Silicon inclusions to reach the best results. Also we should consider that the toughness (critical energy) of the grain boundary and the grain interior should be different.

APPENDIX A

STEADY STATE HEAT CONDUCTION IN A BAR

A.1 An Example

As an example consider the problem of the bar shown in Figure A.1. The bar is of length $l = 1$, is connected to a constant temperature thermal bath of temperature $\hat{\theta} = 2$ from the left and heat is being drawn from it at a constant rate $\hat{q} = 1$ from the right. The heat flow into the bar per unit length from the lateral surfaces is given by $f(x) = x$. The coefficient of thermal conduction $k = 1$. Three two node

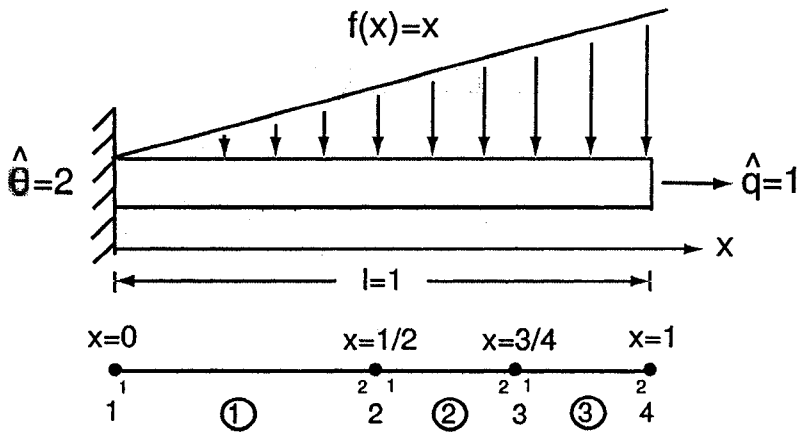


Figure A.1: Example problem.

line elements are selected as shown in Figure 2.5. The local stiffness matrices can be calculated using equation (2.25) and will be

$$[K^1] = \begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix},$$

$$[K^2] = \begin{bmatrix} 4 & -4 \\ -4 & 4 \end{bmatrix},$$

$$[K^3] = \begin{bmatrix} 4 & -4 \\ -4 & 4 \end{bmatrix}.$$

The global stiffness matrix will, therefore, be

$$[K] = \begin{bmatrix} 2 & -2 & 0 & 0 \\ -2 & (2+4) & -4 & 0 \\ 0 & -4 & (4+4) & -4 \\ 0 & 0 & -4 & 4 \end{bmatrix}.$$

The load vectors can be calculated using equation (2.24). Using equation (2.9) and (2.10), the shape functions for the first element are

$$N_1 = 1 - 2x, N_2 = 2x.$$

Therefore,

$$f_1^1 = \int_0^{\frac{1}{2}} (1-2x)x dx = \frac{1}{24},$$
$$f_2^1 = \int_0^{\frac{1}{2}} 2x^2 dx = \frac{1}{12}.$$

The shape functions for the second element are

$$N_1 = 1 - 4(x - \frac{1}{2}), N_2 = 4(x - \frac{1}{2}).$$

Therefore,

$$f_1^2 = \int_{\frac{1}{2}}^{\frac{3}{4}} [(1 - 4(x - \frac{1}{2})]x dx = \frac{7}{96},$$
$$f_2^2 = \int_{\frac{1}{2}}^{\frac{3}{4}} 4(x - \frac{1}{2})x dx = \frac{1}{12}.$$

The shape functions for the third element are

$$N_1 = 1 - 4(x - \frac{3}{4}), N_2 = 4(x - \frac{3}{4}).$$

Therefore,

$$f_1^3 = \int_{\frac{3}{4}}^{1} [(1 - 4(x - \frac{3}{4})]x dx = \frac{5}{48},$$
$$f_2^3 = \int_{\frac{3}{4}}^{1} 4(x - \frac{3}{4})x dx = \frac{11}{96}.$$

The global load vector will become

$$[f] = \begin{bmatrix} \frac{1}{24} \\ \frac{1}{12} + \frac{7}{96} \\ \frac{1}{12} + \frac{5}{48} \\ \frac{11}{96} \end{bmatrix}$$

The problem that we must solve is

$$\begin{bmatrix} 2(1+p) & -2 & 0 & 0 \\ -2 & 6 & -4 & 0 \\ 0 & -4 & 8 & -4 \\ 0 & 0 & -4 & 4 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{24} + 4p \\ \frac{5}{32} \\ \frac{3}{16} \\ \frac{11}{96} - 1 \end{bmatrix},$$

where p is a large number and the stiffness matrix and load vectors have been modified to impose the boundary conditions.

For $p = 1$ the solution will be

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} \frac{7}{4} \\ \frac{71}{48} \\ \frac{167}{128} \\ \frac{13}{12} \end{bmatrix} = \begin{bmatrix} 1.75 \\ 1.47916 \\ 1.30468 \\ 1.08333 \end{bmatrix}.$$

For $p = 10$ the solution is

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} \frac{79}{40} \\ \frac{409}{240} \\ \frac{979}{640} \\ \frac{157}{120} \end{bmatrix} = \begin{bmatrix} 1.975 \\ 1.70416 \\ 1.52968 \\ 1.30833 \end{bmatrix}.$$

For $p = 100$ the solution is

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} \frac{799}{400} \\ \frac{259}{150} \\ \frac{4967}{3200} \\ \frac{1597}{1200} \end{bmatrix} = \begin{bmatrix} 1.9975 \\ 1.72666 \\ 1.55218 \\ 1.33083 \end{bmatrix}.$$

The exact solution is

$$\theta = 2 - \frac{x^3}{6} - \frac{x}{2},$$

which yields

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} 2 \\ \frac{83}{48} \\ \frac{199}{128} \\ \frac{4}{3} \end{bmatrix} = \begin{bmatrix} 2 \\ 1.72916 \\ 1.55468 \\ 1.33333 \end{bmatrix}.$$

Comparison of the exact solution and the finite element method solution shows that for $p = 100$ there is less than 1 error in the finite element solution.

A.2 Organize the Finite Element Method Program

Finite element method programs are normally organized in the following manner. This allows each part of the program to be a separate module which can be replaced from one application to another.

1. Division of the domain into elements, numbering the elements, numbering the nodes, providing the connection between local node numbers and global node numbers, providing the coordinate for each node.

2. Numbering boundary elements and connection between local node numbers and global node numbers.

3. Assembling global stiffness matrix, $[K]$, and load vector, $[f]$, element by element.

(a) Calculate the element stiffness matrix, $[K^e]$, for each element and assemble it into the global stiffness matrix.

(b) Calculate the element load vector, $[f^e]$, for each element and assemble it into the global load vector.

4. Modifying global stiffness matrix and global load vector to enforce boundary conditions.

5. Solving the resulting system of equations (i.e., $[K][\theta] = [f]$) for the value of the unknown function at each nodal point, $[\theta]$.

6. Presenting the results in a suitable format (post-processing).

This layout of the program provides a certain degree of flexibility. For example, the same problem can be solved using several different equation solvers, or the same program can solve different problems just by changing the method for calculating the element stiffness matrix, load vector, and boundary conditions.

APPENDIX B

DOCUMENTATION OF FINAL PROGRAM

B.1 Crack Propagation Program Using XFEM/LS Method

The following programs is a part of getfem++ for crack propagation using XFEM/LS method. The parameters file is shown all parameters that used for the main program.

B.1.1 Crack propagation program

```
// -*- c++ -*- (enables emacs c++ mode)
//===========================================================================
//
// Copyright (C) 2002-2006 Yves Renard, Julien Pommier.
//
// This file is a part of GETFEM++
//
// Getfem++ is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; version 2.1 of the License.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
// You should have received a copy of the GNU Lesser General Public
// License along with this program; if not, write to the Free Software
// Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301,
// USA.
//===========================================================================

/**
 * Linear Elastostatic problem with a crack.
 *
 * This program is used to check that getfem++ is working. This is also
 * a good example of use of Getfem++.
*/

#include <getfem_assembling.h> /* import assembly methods (and norms comp.) */
#include <getfem_export.h>   /* export functions (save solution in a file)  */
#include <getfem_import.h>
```

```
#include <getfem_derivatives.h>
#include <getfem_regular_meshes.h>
#include <getfem_model_solvers.h>
#include <getfem_mesh_im_level_set.h>
#include <getfem_mesh_fem_level_set.h>
#include <getfem_mesh_fem_product.h>
#include <getfem_mesh_fem_global_function.h>
#include <getfem_spider_fem.h>
#include <getfem_mesh_fem_sum.h>
#include <gmm.h>

/* some Getfem++ types that we will be using */
using bgeot::base_small_vector; /* special class for small (dim<16) vectors */
using bgeot::base_node;  /* geometrical nodes(derived from base_small_vector)*/
using bgeot::scalar_type; /* = double */
using bgeot::size_type;   /* = unsigned long */
using bgeot::base_matrix; /* small dense matrix. */
/* definition of some matrix/vector types. These ones are built
 * using the predefined types in Gmm++ */
typedef getfem::modeling_standard_sparse_vector sparse_vector;
typedef getfem::modeling_standard_sparse_matrix sparse_matrix;
typedef getfem::modeling_standard_plain_vector  plain_vector;

/******************************************************************************/
/*  Exact solution. */
/******************************************************************************/

#define VALIDATE_XFEM
#ifdef VALIDATE_XFEM

/* returns sin(theta/2) where theta is the angle
   of 0-(x,y) with the axis 0x */
scalar_type sint2(scalar_type x, scalar_type y) {
  scalar_type r = sqrt(x*x+y*y);
  if (r == 0) return 0;
  else return (y<0 ? -1:1) * sqrt(gmm::abs(r-x)/(2*r));
  // sometimes (gcc3.3.2 -03), r-x < 0 ....
} scalar_type cost2(scalar_type x, scalar_type y) {
  scalar_type r = sqrt(x*x+y*y);
  if (r == 0) return 0;
  else return sqrt(gmm::abs(r+x)/(2*r));
} /* analytical solution for a semi-infinite crack [-inf,a] in an
   infinite plane submitted to +sigma above the crack
   and -sigma under the crack. (The crack is directed along the x axis).
```

```
   nu and E are the poisson ratio and young modulus

   solution taken from "an extended finite element method with high order
   elements for curved cracks", Stazi, Budyn,Chessa, Belytschko */

void elasticite2lame(const scalar_type young_modulus,
             const scalar_type poisson_ratio,
             scalar_type& lambda, scalar_type& mu) {
  mu = young_modulus/(2*(1+poisson_ratio));
  lambda = 2*mu*poisson_ratio/(1-poisson_ratio);
}
void sol_ref_infinite_plane(scalar_type nu, scalar_type E, scalar_type sigma,
                scalar_type a, scalar_type xx, scalar_type y,
                base_small_vector& U, int mode,
                base_matrix *pgrad) {
  scalar_type x  = xx-a; /* the eq are given relatively to the crack tip */
  //scalar_type KI = sigma*sqrt(M_PI*a);
  scalar_type r = std::max(sqrt(x*x+y*y),1e-16);
  scalar_type sqrtr = sqrt(r), sqrtr3 = sqrtr*sqrtr*sqrtr;
  scalar_type cost = x/r, sint = y/r;
  scalar_type theta = atan2(y,x);
  scalar_type s2 = sin(theta/2); //sint2(x,y);
  scalar_type c2 = cos(theta/2); //cost2(x,y);
  // scalar_type c3 = cos(3*theta/2); //4*c2*c2*c2-3*c2; /* cos(3*theta/2) */
  // scalar_type s3 = sin(3*theta/2); //4*s2*c2*c2-s2;  /* sin(3*theta/2) */
  scalar_type lambda, mu;
  elasticite2lame(E,nu,lambda,mu);

  U.resize(2);
  if (pgrad) (*pgrad).resize(2,2);
  scalar_type C= 1./E * (mode == 1 ? 1. : (1+nu));
  if (mode == 1) {
    scalar_type A=2+2*mu/(lambda+2*mu);
    scalar_type B=-2*(lambda+mu)/(lambda+2*mu);
    U[0] = sqrtr/sqrt(2*M_PI) * C * c2 * (A + B*cost);
    U[1] = sqrtr/sqrt(2*M_PI) * C * s2 * (A + B*cost);
    if (pgrad) {
      (*pgrad)(0,0) = C/(2.*sqrt(2*M_PI)*sqrtr)
    * (cost*c2*A-cost*cost*c2*B+sint*s2*A+sint*s2*B*cost+2*c2*B);
      (*pgrad)(1,0) = -C/(2*sqrt(2*M_PI)*sqrtr)
    * (-sint*c2*A+sint*c2*B*cost+cost*s2*A+cost*cost*s2*B);
      (*pgrad)(0,1) = C/(2.*sqrt(2*M_PI)*sqrtr)
    * (cost*s2*A-cost*cost*s2*B-sint*c2*A-sint*c2*B*cost+2*s2*B);
```

```
    (*pgrad)(1,1) = C/(2.*sqrt(2*M_PI)*sqrtr)
  * (sint*s2*A-sint*s2*B*cost+cost*c2*A+cost*cost*c2*B);
  }
} else if (mode == 2) {
  scalar_type C1 = (lambda+3*mu)/(lambda+mu);
  U[0] = sqrtr/sqrt(2*M_PI) * C * s2 * (C1 + 2 + cost);
  U[1] = sqrtr/sqrt(2*M_PI) * C * c2 * (C1 - 2 + cost) * (-1.);
  if (pgrad) {
    (*pgrad)(0,0) = C/(2.*sqrt(2*M_PI)*sqrtr)
  * (cost*s2*C1+2*cost*s2-cost*cost*s2-sint*c2*C1
     -2*sint*c2-sint*cost*c2+2*s2);
    (*pgrad)(1,0) = C/(2.*sqrt(2*M_PI)*sqrtr)
  * (sint*s2*C1+2*sint*s2-sint*s2*cost+cost*c2*C1
     +2*cost*c2+cost*cost*c2);
    (*pgrad)(0,1) = -C/(2.*sqrt(2*M_PI)*sqrtr)
  * (cost*c2*C1-2*cost*c2-cost*cost*c2+sint*s2*C1
     -2*sint*s2+sint*s2*cost+2*c2);
    (*pgrad)(1,1) =  C/(2.*sqrt(2*M_PI)*sqrtr)
  * (-sint*c2*C1+2*sint*c2+sint*cost*c2+cost*s2*C1
     -2*cost*s2+cost*cost*s2);
  }
} else if (mode == 100) {
  U[0] = - sqrtr3 * (c2 + 4./3 *(7*mu+3*lambda)/(lambda+mu)*c2*s2*s2
            -1./3*(7*mu+3*lambda)/(lambda+mu)*c2);
  U[1] = - sqrtr3 * (s2+4./3*(lambda+5*mu)/(lambda+mu)*s2*s2*s2
            -(lambda+5*mu)/(lambda+mu)*s2);
  if (pgrad) {
    (*pgrad)(0,0) = 2*sqrtr*(-6*cost*c2*mu+7*cost*c2*c2*c2*mu
                -3*cost*c2*lambda+3*cost*c2*c2*c2*lambda
                -2*sint*s2*mu
                +7*sint*s2*c2*c2*mu-sint*s2*lambda
                +3*sint*s2*c2*c2*lambda)/(lambda+mu);
    (*pgrad)(1,0) = -2*sqrtr*(6*sint*c2*mu-7*sint*c2*c2*c2*mu
              +3*sint*c2*lambda-3*sint*c2*c2*c2*lambda
              -2*cost*s2*mu
              +7*cost*s2*c2*c2*mu-cost*s2*lambda
              +3*cost*s2*c2*c2*lambda)/(lambda+mu);
    (*pgrad)(0,1) = 2*sqrtr*(-2*cost*s2*mu-cost*s2*lambda
                +cost*s2*c2*c2*lambda+5*cost*s2*c2*c2*mu
                +4*sint*c2*mu
                +sint*c2*lambda-sint*c2*c2*c2*lambda
                -5*sint*c2*c2*c2*mu)/(lambda+mu);
    (*pgrad)(1,1) = 2*sqrtr*(-2*sint*s2*mu-sint*s2*lambda
                +sint*s2*c2*c2*lambda+5*sint*s2*c2*c2*mu
```

```
                -4*cost*c2*mu
                -cost*c2*lambda+cost*c2*c2*c2*lambda
                +5*cost*c2*c2*c2*mu)/(lambda+mu);
  }
} else if (mode == 101) {
  U[0] = -4*sqrtr3*s2*(-lambda-2*mu+7*lambda*c2*c2
          +11*mu*c2*c2)/(3*lambda-mu);
  U[1] = -4*sqrtr3*c2*(-3*lambda+3*lambda*c2*c2-mu*c2*c2)/(3*lambda-mu);
  if (pgrad) {
    (*pgrad)(0,0) = -6*sqrtr*(-cost*s2*lambda-2*cost*s2*mu
              +7*cost*s2*lambda*c2*c2
              +11*cost*s2*mu*c2*c2+5*sint*c2*lambda
              +8*sint*c2*mu-7*sint*c2*c2*c2*lambda
              -11*sint*c2*c2*c2*mu)/(3*lambda-mu);
    (*pgrad)(1,0) = -6*sqrtr*(-sint*s2*lambda-2*sint*s2*mu
              +7*sint*s2*lambda*c2*c2
              +11*sint*s2*mu*c2*c2-5*cost*c2*lambda
              -8*cost*c2*mu+7*cost*c2*c2*c2*lambda
              +11*cost*c2*c2*c2*mu)/(3*lambda-mu);
    (*pgrad)(0,1) = -6*sqrtr*(-3*cost*c2*lambda+3*cost*c2*c2*c2*lambda
              -cost*c2*c2*c2*mu-sint*s2*lambda
              +3*sint*s2*lambda*c2*c2
              -sint*s2*mu*c2*c2)/(3*lambda-mu);
    (*pgrad)(1,1) = 6*sqrtr*(3*sint*c2*lambda
                -3*sint*c2*c2*c2*lambda+sint*c2*c2*c2*mu
                -cost*s2*lambda+3*cost*s2*lambda*c2*c2
                -cost*s2*mu*c2*c2)/(3*lambda-mu);
  }
} else if (mode == 10166666) {

  U[0] = 4*sqrtr3*s2*(-lambda+lambda*c2*c2-3*mu*c2*c2)/(lambda-3*mu);
  U[1] = 4*sqrtr3*c2*(-3*lambda-6*mu+5*lambda*c2*c2+9*mu*c2*c2)/(lambda-3*mu);
  if (pgrad) {
    (*pgrad)(0,0) = 6*sqrtr*(-cost*s2*lambda+cost*s2*lambda*c2*c2-
                3*cost*s2*mu*c2*c2-2*sint*c2*mu+sint*c2*lambda-
                sint*c2*c2*c2*lambda
                +3*sint*c2*c2*c2*mu)/(lambda-3*mu);
    (*pgrad)(1,0) = 6*sqrtr*(-sint*s2*lambda+sint*s2*lambda*c2*c2-
                3*sint*s2*mu*c2*c2+2*cost*c2*mu-cost*c2*lambda+
                cost*c2*c2*c2*lambda
                -3*cost*c2*c2*c2*mu)/(lambda-3*mu);
    (*pgrad)(0,1) = 6*sqrtr*(-3*cost*c2*lambda-6*cost*c2*mu
                +5*cost*c2*c2*c2*lambda+
                9*cost*c2*c2*c2*mu-sint*s2*lambda-2*sint*s2*mu+
```

```
                5*sint*s2*lambda*c2*c2
                +9*sint*s2*mu*c2*c2)/(lambda-3*mu);
    (*pgrad)(1,1) = -6*sqrtr*(3*sint*c2*lambda+6*sint*c2*mu
                -5*sint*c2*c2*c2*lambda-
                9*sint*c2*c2*c2*mu-cost*s2*lambda-2*cost*s2*mu+
                5*cost*s2*lambda*c2*c2
                +9*cost*s2*mu*c2*c2)/(lambda-3*mu);
  }
 } else assert(0);
 if (std::isnan(U[0]))
  cerr << "raaah not a number ... nu=" << nu << ", E=" << E << ", sig="
   << sigma << ", a=" << a << ", xx=" << xx << ", y=" << y << ", r="
   << r << ", sqrtr=" << sqrtr << ", cost=" << cost << ", U=" << U[0]
   << "," << U[1] << endl;
   assert(!std::isnan(U[0]));
   assert(!std::isnan(U[1]));
}
struct exact_solution {
 getfem::mesh_fem_global_function mf;
 getfem::base_vector U;
 exact_solution(getfem::mesh &me) : mf(me) {}
 void init(int mode, scalar_type lambda, scalar_type mu,
       getfem::level_set &ls) {
  std::vector<getfem::pglobal_function> cfun(4);
  for (unsigned j=0; j < 4; ++j)
    cfun[j] = getfem::isotropic_crack_singular_2D(j, ls);
  mf.set_functions(cfun);
  mf.set_qdim(1);

  U.resize(8); assert(mf.nb_dof() == 4);
  getfem::base_vector::iterator it = U.begin();
  scalar_type coeff=0.;
  switch(mode) {
    case 1: {
  scalar_type A=2+2*mu/(lambda+2*mu), B=-2*(lambda+mu)/(lambda+2*mu);
  /* "colonne" 1: ux, colonne 2: uy */
  *it++ = 0;       *it++ = A-B; /* sin(theta/2) */
  *it++ = A+B;     *it++ = 0;   /* cos(theta/2) */
  *it++ = -B;      *it++ = 0;   /* sin(theta/2)*sin(theta) */
  *it++ = 0;       *it++ = B;   /* cos(theta/2)*cos(theta) */
  coeff = 1/sqrt(2*M_PI);
    } break;
    case 2: {
  scalar_type C1 = (lambda+3*mu)/(lambda+mu);
```

```
    *it++ = C1+2-1;   *it++ = 0;
    *it++ = 0;       *it++ = -(C1-2+1);
    *it++ = 0;       *it++ = 1;
    *it++ = 1;       *it++ = 0;
    coeff = 2*(mu+lambda)/(lambda+2*mu)/sqrt(2*M_PI);
      } break;
      default:
    assert(0);
    break;
    }
    gmm::scale(U, coeff);
  }
};

base_small_vector sol_f(const base_node &x) {
  int N = x.size();
  base_small_vector res(N);
  return res;
}
#else base_small_vector sol_f(const base_node &x) {
  int N = x.size();
  base_small_vector res(N); res[N-1] = x[N-1];
  return res;
}
#endif

/**************************************************************************/
/*  Structure for the crack problem. */
/**************************************************************************/

struct crack_problem {

  enum { DIRICHLET_BOUNDARY_NUM = 0, NEUMANN_BOUNDARY_NUM = 1, NEUMANN_BOUNDARY_NUM1=2,
         NEUMANN_HOMOGENE_BOUNDARY_NUM=3};
  getfem::mesh mesh;  /* the mesh */
  getfem::mesh_level_set mls;       /* the integration methods.              */
  getfem::mesh_im_level_set mim;    /* the integration methods.              */
  getfem::mesh_fem mf_pre_u;
  getfem::mesh_fem mf_mult;
  getfem::mesh_fem_level_set mfls_u;
  getfem::mesh_fem_global_function mf_sing_u;
  getfem::mesh_fem mf_partition_of_unity;
  getfem::mesh_fem_product mf_product;
  getfem::mesh_fem_sum mf_u_sum;
```

```
  getfem::spider_fem *spider;
  getfem::mesh_fem mf_us;

  getfem::mesh_fem& mf_u() { return mf_u_sum; }
  // getfem::mesh_fem& mf_u() { return mf_us; }

  scalar_type lambda, mu;    /* Lame coefficients.                */
  getfem::mesh_fem mf_rhs;   /* mesh_fem for the right hand side (f(x),..)   */
  getfem::mesh_fem mf_p;     /* mesh_fem for the pressure for mixed form     */
#ifdef VALIDATE_XFEM
  exact_solution exact_sol;
#endif
  int bimaterial;           /* For bimaterial interface fracture */
  double lambda_up, lambda_down;  /*Lame coeff for bimaterial case*/
  // scalar_type lambda_inc, mu_inc;    // Lam coefficients. of the inclusions
  getfem::level_set ls;      /* The two level sets defining the crack.       */
  getfem::level_set ls2, ls3; /* The two level-sets defining the add. cracks.*/
  base_small_vector translation;
  scalar_type theta0;
  scalar_type spider_radius;
  unsigned spider_Nr;
  unsigned spider_Ntheta;
  int spider_K;
  scalar_type residual;      /* max residual for the iterative solvers      */
  bool mixed_pressure, add_crack;
  unsigned dir_with_mult;
  scalar_type cutoff_radius, cutoff_radius1, cutoff_radius0, enr_area_radius;
  int enrichment_option;
  size_type cutoff_func;
  std::string datafilename;
  ftool::md_param PARAM;

  bool solve(plain_vector &U);
  void init(void);
  crack_problem(void) : mls(mesh), mim(mls), mf_pre_u(mesh), mf_mult(mesh),
            mfls_u(mls, mf_pre_u), mf_sing_u(mesh),
            mf_partition_of_unity(mesh),
            mf_product(mf_partition_of_unity, mf_sing_u),

            mf_u_sum(mesh), mf_us(mesh), mf_rhs(mesh), mf_p(mesh),
#ifdef VALIDATE_XFEM
            exact_sol(mesh),
#endif
            ls(mesh, 1, true), ls2(mesh, 1, true),
```

```
            ls3(mesh, 1, true) {}
};

/* Read parameters from the .param file, build the mesh, set finite element
 * and integration methods and selects the boundaries. */
void crack_problem::init(void) {
  std::string MESH_TYPE = PARAM.string_value("MESH_TYPE","Mesh type ");
  std::string FEM_TYPE  = PARAM.string_value("FEM_TYPE","FEM name");
  std::string INTEGRATION = PARAM.string_value("INTEGRATION",
                            "Name of integration method");
  std::string SIMPLEX_INTEGRATION = PARAM.string_value("SIMPLEX_INTEGRATION",
                    "Name of simplex integration method");
  std::string SINGULAR_INTEGRATION = PARAM.string_value("SINGULAR_INTEGRATION");

  add_crack = (PARAM.int_value("ADDITIONAL_CRACK", "An additional crack ?") != 0);
  enrichment_option = PARAM.int_value("ENRICHMENT_OPTION",
                      "Enrichment option");
  cout << "MESH_TYPE=" << MESH_TYPE << "\n";
  cout << "FEM_TYPE="  << FEM_TYPE << "\n";
  cout << "INTEGRATION=" << INTEGRATION << "\n";

  spider_radius = PARAM.real_value("SPIDER_RADIUS","spider_radius");
  spider_Nr = PARAM.int_value("SPIDER_NR","Spider_Nr ");
  spider_Ntheta = PARAM.int_value("SPIDER_NTHETA","Ntheta ");
  spider_K = PARAM.int_value("SPIDER_K","K ");

  translation.resize(2);
  translation[0] =0.5;
  translation[1] =0.;
  theta0 =0;

  std::string meshname
    (PARAM.string_value("MESHNAME", "Nom du fichier de maillage"));
  /* First step : build the mesh */
  bgeot::pgeometric_trans pgt =
    bgeot::geometric_trans_descriptor(MESH_TYPE);
  getfem::import_mesh(meshname,"gid", mesh); /* read the mesh from external gid file */
   mesh.optimize_structure();

  size_type N = pgt->dim();
  /* std::vector<size_type> nsubdiv(N);
  std::fill(nsubdiv.begin(),nsubdiv.end(),
        PARAM.int_value("NX", "Nomber of space steps "));
  getfem::regular_unit_mesh(mesh, nsubdiv, pgt,
```

```
                PARAM.int_value("MESH_NOISED") != 0);
base_small_vector tt(N); tt[1] = -0.5;
mesh.translation(tt); */

datafilename = PARAM.string_value("ROOTFILENAME","Base name of data files.");
residual = PARAM.real_value("RESIDUAL"); if (residual == 0.) residual = 1e-10;
enr_area_radius = PARAM.real_value("RADIUS_ENR_AREA",
                   "radius of the enrichment area");

bimaterial = PARAM.int_value("BIMATERIAL", "bimaterial interface crack");

if (bimaterial == 1){
  mu = PARAM.real_value("MU", "Lame coefficient mu");
  lambda_up = PARAM.int_value("LAMBDA_UP", "Lame Coef");
  lambda_down = PARAM.int_value("LAMBDA_DOWN", "Lame Coef");
  lambda = PARAM.real_value("LAMBDA", "Lame coefficient lambda");
  // mu_inc = PARAM.real_value("MUINC", "Lam coefficient mu_inc");
  // lambda_inc = PARAM.real_value("LAMBDAINC", "Lam coefficient lambda_inc");
}
else{

  mu = PARAM.real_value("MU", "Lame coefficient mu");
  lambda = PARAM.real_value("LAMBDA", "Lame coefficient lambda");
}

cutoff_func = PARAM.int_value("CUTOFF_FUNC", "cutoff function");
cutoff_radius = PARAM.real_value("CUTOFF", "Cutoff");
cutoff_radius1 = PARAM.real_value("CUTOFF1", "Cutoff1");
cutoff_radius0 = PARAM.real_value("CUTOFF0", "Cutoff0");
mf_u().set_qdim(N);

/* set the finite element on the mf_u */
getfem::pfem pf_u =
  getfem::fem_descriptor(FEM_TYPE);
getfem::pintegration_method ppi =
  getfem::int_method_descriptor(INTEGRATION);
getfem::pintegration_method simp_ppi =
  getfem::int_method_descriptor(SIMPLEX_INTEGRATION);
getfem::pintegration_method sing_ppi = (SINGULAR_INTEGRATION.size() ?
      getfem::int_method_descriptor(SINGULAR_INTEGRATION) : 0);

mim.set_integration_method(mesh.convex_index(), ppi);
mls.add_level_set(ls);
if (add_crack) { mls.add_level_set(ls2); mls.add_level_set(ls3); }
```

```
  mim.set_simplex_im(simp_ppi, sing_ppi);
  mf_pre_u.set_finite_element(mesh.convex_index(), pf_u);
  mf_mult.set_finite_element(mesh.convex_index(), pf_u);
  mf_mult.set_qdim(N);
  mf_partition_of_unity.set_classical_finite_element(1);

//   if (enrichment_option == 3 || enrichment_option == 4) {
//     spider = new getfem::spider_fem(spider_radius, mim, spider_Nr,
//                 spider_Ntheta, spider_K, translation,
//                 theta0);
//     mf_us.set_finite_element(mesh.convex_index(),spider->get_pfem());
//     for (dal::bv_visitor_c i(mf_us.convex_index()); !i.finished(); ++i) {
//       if (mf_us.fem_of_element(i)->nb_dof(i) == 0) {
//  mf_us.set_finite_element(i,0);
//       }
//     }
//   }

  mixed_pressure =
    (PARAM.int_value("MIXED_PRESSURE","Mixed version or not.") != 0);
  dir_with_mult = PARAM.int_value("DIRICHLET_VERSINO");
  if (mixed_pressure) {
    std::string FEM_TYPE_P  = PARAM.string_value("FEM_TYPE_P","FEM name P");
    mf_p.set_finite_element(mesh.convex_index(),
              getfem::fem_descriptor(FEM_TYPE_P));
  }
  /* set the finite element on mf_rhs (same as mf_u is DATA_FEM_TYPE is
     not used in the .param file */
  std::string data_fem_name = PARAM.string_value("DATA_FEM_TYPE");
  if (data_fem_name.size() == 0) {
    if (!pf_u->is_lagrange()) {
      DAL_THROW(dal::failure_error, "You are using a non-lagrange FEM. "
        << "In that case you need to set "
        << "DATA_FEM_TYPE in the .param file");
    }
    mf_rhs.set_finite_element(mesh.convex_index(), pf_u);
  } else {
    mf_rhs.set_finite_element(mesh.convex_index(),
                getfem::fem_descriptor(data_fem_name));
  }
  /* set boundary conditions
   * (Neuman on the upper face, Dirichlet elsewhere) */
  cout << "Selecting Neumann and Dirichlet boundaries\n";
```

```
  getfem::mesh_region border_faces;
  getfem::outer_faces_of_mesh(mesh, border_faces);
  for (getfem::mr_visitor i(border_faces); !i.finished(); ++i) {

    base_node un = mesh.normal_of_face_of_convex(i.cv(), i.f());
    un /= gmm::vect_norm2(un);
    if(bimaterial == 1) {

      if (un[0]  > 1.0E-7 ) { // new Neumann face
    mesh.region(DIRICHLET_BOUNDARY_NUM).add(i.cv(), i.f());
      } else {
    if (un[1]  > 1.0E-7 ) {
      cout << "normal = " << un << endl;
      mesh.region(NEUMANN_BOUNDARY_NUM1).add(i.cv(), i.f());
    }
    else {
      if (un[1]  < -1.0E-7 ) {
        cout << "normal = " << un << endl;
        mesh.region(NEUMANN_BOUNDARY_NUM).add(i.cv(), i.f());
      }
      else {
        if (un[0]  < -1.0E-7 ) {
          cout << "normal = " << un << endl;
          mesh.region(NEUMANN_HOMOGENE_BOUNDARY_NUM).add(i.cv(), i.f());
        }
      }
    }
      }
    }
    else {

#ifdef VALIDATE_XFEM
      mesh.region(DIRICHLET_BOUNDARY_NUM).add(i.cv(), i.f());
      #else
    base_node un = mesh.normal_of_face_of_convex(i.cv(), i.f());
    un /= gmm::vect_norm2(un);
    if (un[0] - 1.0 < -1.0E-7) { // new Neumann face
      mesh.region(NEUMANN_BOUNDARY_NUM).add(i.cv(), i.f());
    } else {
      cout << "normal = " << un << endl;
      mesh.region(DIRICHLET_BOUNDARY_NUM).add(i.cv(), i.f());
      }
#endif
    }
```

```
  }

#ifdef VALIDATE_XFEM
  exact_sol.init(1, lambda, mu, ls);
#endif }

base_small_vector ls_function(const base_node P, int num = 0) {
  scalar_type x = P[0], y = P[1];
  base_small_vector res(2);
  switch (num) {
    case 0: {
      res[0] = y;
      res[1] = -.5 + x;
    } break;
    case 1: {
      res[0] = gmm::vect_dist2(P, base_node(0.5, 0.)) - .25;
      res[1] = gmm::vect_dist2(P, base_node(0.25, 0.0)) - 0.27;
    } break;
    case 2: {
      res[0] = x - 0.25;
      res[1] = gmm::vect_dist2(P, base_node(0.25, 0.0)) - 0.35;
    } break;
    default: assert(0);
  }
  return res;
}

bool crack_problem::solve(plain_vector &U) {
  size_type nb_dof_rhs = mf_rhs.nb_dof();
  size_type N = mesh.dim();
  ls.reinit();
  cout << "ls.get_mesh_fem().nb_dof() = " << ls.get_mesh_fem().nb_dof() << "\n";
  for (size_type d = 0; d < ls.get_mesh_fem().nb_dof(); ++d) {
    ls.values(0)[d] = ls_function(ls.get_mesh_fem().point_of_dof(d), 0)[0];
    ls.values(1)[d] = ls_function(ls.get_mesh_fem().point_of_dof(d), 0)[1];
  }
  ls.touch();

  if (add_crack) {
    ls2.reinit();
    for (size_type d = 0; d < ls2.get_mesh_fem().nb_dof(); ++d) {
      ls2.values(0)[d] = ls_function(ls2.get_mesh_fem().point_of_dof(d), 1)[0];
      ls2.values(1)[d] = ls_function(ls2.get_mesh_fem().point_of_dof(d), 1)[1];
    }
```

```
  ls2.touch();

  ls3.reinit();
  for (size_type d = 0; d < ls3.get_mesh_fem().nb_dof(); ++d) {
    ls3.values(0)[d] = ls_function(ls2.get_mesh_fem().point_of_dof(d), 2)[0];
    ls3.values(1)[d] = ls_function(ls2.get_mesh_fem().point_of_dof(d), 2)[1];
  }
  ls3.touch();
}

mls.adapt();
mim.adapt();
mfls_u.adapt();
std::vector<getfem::pglobal_function> vfunc(4);
for (size_type i = 0; i < 4; ++i)
  vfunc[i] = isotropic_crack_singular_2D(i, ls,
                    (enrichment_option == 2) ? 0.0 : cutoff_radius,
                    (enrichment_option == 2) ? 0.0 : cutoff_radius1,
                    (enrichment_option == 2) ? 0.0 : cutoff_radius0,
                    cutoff_func);

mf_sing_u.set_functions(vfunc);

if (enrichment_option == 3 || enrichment_option == 4) {
  spider = new getfem::spider_fem(spider_radius, mim, spider_Nr,
                 spider_Ntheta, spider_K, translation,
                 theta0);
  mf_us.set_finite_element(mesh.convex_index(),spider->get_pfem());
  for (dal::bv_visitor_c i(mf_us.convex_index()); !i.finished(); ++i) {
    if (mf_us.fem_of_element(i)->nb_dof(i) == 0) {
  mf_us.set_finite_element(i,0);
    }
  }
  spider->check();
}

switch (enrichment_option) {
case 1 :{
  if(cutoff_func == 0)
    cout<<"Using exponential Cutoff..."<<endl;
  else
 cout<<"Using Polynomial Cutoff..."<<endl;
  mf_u_sum.set_mesh_fems(mf_sing_u, mfls_u); break;
}
```

```
case 2 :
  {
    dal::bit_vector enriched_dofs;
    plain_vector X(mf_partition_of_unity.nb_dof());
    plain_vector Y(mf_partition_of_unity.nb_dof());
    getfem::interpolation(ls.get_mesh_fem(), mf_partition_of_unity,
              ls.values(1), X);
    getfem::interpolation(ls.get_mesh_fem(), mf_partition_of_unity,
              ls.values(0), Y);
    for (size_type j = 0; j < mf_partition_of_unity.nb_dof(); ++j) {
  if (gmm::sqr(X[j]) + gmm::sqr(Y[j]) <= gmm::sqr(enr_area_radius))
    enriched_dofs.add(j);
    }
    if (enriched_dofs.card() < 3)
  DAL_WARNINGO("There is " << enriched_dofs.card() <<
         " enriched dofs for the crack tip");
    mf_product.set_enrichment(enriched_dofs);
    mf_u_sum.set_mesh_fems(mf_product, mfls_u);
  }
  break;
  case 3 : mf_u_sum.set_mesh_fems(mf_us); break;
case 4 :
  mf_u_sum.set_mesh_fems(mf_us, mfls_u);
  break;
default : mf_u_sum.set_mesh_fems(mfls_u); break;
}
U.resize(mf_u().nb_dof());
if (mixed_pressure) cout << "Number of dof for P: " << mf_p.nb_dof() << endl;
cout << "Number of dof for u: " << mf_u().nb_dof() << endl;

// Linearized elasticity brick.

getfem::mdbrick_isotropic_linearized_elasticity<>
  ELAS(mim, mf_u(), mixed_pressure ? 0.0 : lambda, mu);

if(bimaterial == 1){
  cout<<"_____________________________________________________________________"<<endl;
  cout<<"CASE OF BIMATERIAL CRACK  with lambda_up = "<<lambda_up<<";
  cout<<"and lambda_down = "<<lambda_down<<endl;
  cout<<"_____________________________________________________________________"<<endl;
  std::vector<float> bi_lambda(ELAS.lambda().mf().nb_dof());

  cout<<"ELAS.lambda().mf().nb_dof()==="<<ELAS.lambda().mf().nb_dof()<<endl;
```

```
  for (size_type ite = 0; ite < ELAS.lambda().mf().nb_dof();ite++) {
    if (ELAS.lambda().mf().point_of_dof(ite)[1] > 0)
  bi_lambda[ite] = lambda_up;
  else
    bi_lambda[ite] = lambda_down;
  }
  //cout<<"bi_lambda.size() = "<<bi_lambda.size()<<endl;
  // cout<<"ELAS.lambda().mf().nb_dof()==="<<ELAS.lambda().mf().nb_dof()<<endl;

  ELAS.lambda().set(bi_lambda);
}

  getfem::mdbrick_abstract<> *pINCOMP;
if (mixed_pressure) {
  getfem::mdbrick_linear_incomp<> *incomp
    = new getfem::mdbrick_linear_incomp<>(ELAS, mf_p);
  incomp->penalization_coeff().set(1.0/lambda);
  pINCOMP = incomp;
} else pINCOMP = &ELAS;

// Defining the volumic source term.
plain_vector F(nb_dof_rhs * N);
for (size_type i = 0; i < nb_dof_rhs; ++i)
    gmm::copy(sol_f(mf_rhs.point_of_dof(i)),
      gmm::sub_vector(F, gmm::sub_interval(i*N, N)));

// Volumic source term brick.
getfem::mdbrick_source_term<> VOL_F(*pINCOMP, mf_rhs, F);

// Defining the Neumann condition right hand side.
gmm::clear(F);

// Neumann condition brick.

getfem::mdbrick_abstract<> *pNEUMANN;

if(bimaterial ==  1){
for(size_type i = 1; i<F.size(); i=i+2)
  F[i]=-0.2;
}
getfem::mdbrick_source_term<>  NEUMANN(VOL_F, mf_rhs, F,NEUMANN_BOUNDARY_NUM);

 gmm::clear(F);
 getfem::mdbrick_source_term<> NEUMANN_HOM(NEUMANN, mf_rhs, F,NEUMANN_HOMOGENE_BOUNDARY_NUM);
```

```
  gmm::clear(F);
  for(size_type i = 1; i<F.size(); i=i+2)
    F[i]=0.2;

  getfem::mdbrick_source_term<> NEUMANN1(NEUMANN_HOM, mf_rhs, F,NEUMANN_BOUNDARY_NUM1);

  if (bimaterial ==1)
    pNEUMANN = & NEUMANN1;
  else
    pNEUMANN = & NEUMANN;

 //toto_solution toto(mf_rhs.linked_mesh()); toto.init();
 //assert(toto.mf.nb_dof() == 1);
 // Dirichlet condition brick.
   getfem::mdbrick_Dirichlet<> final_model(*pNEUMANN, DIRICHLET_BOUNDARY_NUM, mf_mult);

   if (bimaterial == 1)
     final_model.rhs().set(exact_sol.mf,0);
   else {
#ifdef VALIDATE_XFEM
     final_model.rhs().set(exact_sol.mf,exact_sol.U);
#endif
   }
   final_model.set_constraints_type(getfem::constraints_type(dir_with_mult));

   // Generic solve.
 cout << "Total number of variables : " << final_model.nb_dof() << endl;
 getfem::standard_model_state MS(final_model);
 gmm::iteration iter(residual, 1, 40000);
 getfem::standard_solve(MS, final_model, iter);

 // Solution extraction
 gmm::copy(ELAS.get_solution(MS), U);

 return (iter.converged());
}

/**************************************************************************/
/*  main program. */
/**************************************************************************/

int main(int argc, char *argv[]) {
```

```
DAL_SET_EXCEPTION_DEBUG; // Exceptions make a memory fault, to debug.
FE_ENABLE_EXCEPT;        // Enable floating point exception for Nan.

//getfem::getfem_mesh_level_set_noisy();

try {
  crack_problem p;
  p.PARAM.read_command_line(argc, argv);
  p.init();
  p.mesh.write_to_file(p.datafilename + ".mesh");
  plain_vector U(p.mf_u().nb_dof());
  if (!p.solve(U)) DAL_THROW(dal::failure_error,"Solve has failed");

  {
    getfem::mesh mcut;
    p.mls.global_cut_mesh(mcut);
    unsigned Q = p.mf_u().get_qdim();
    getfem::mesh_fem mf(mcut, Q);
    mf.set_classical_discontinuous_finite_element(2, 0.001);
    // mf.set_finite_element
    //    (getfem::fem_descriptor("FEM_PK_DISCONTINUOUS(2, 2, 0.0001)"));
    plain_vector V(mf.nb_dof());

    getfem::interpolation(p.mf_u(), mf, U, V);

    getfem::stored_mesh_slice sl;
    getfem::mesh mcut_refined;

    unsigned NX = p.PARAM.int_value("NX"), nn;
    if (NX < 6) nn = 24;
    else if (NX < 12) nn = 8;
    else if (NX < 30) nn = 3;
    else nn = 1;

    /* choose an adequate slice refinement based on the distance to the crack tip */
    std::vector<bgeot::short_type> nrefine(mcut.convex_index().last_true()+1);
    for (dal::bv_visitor cv(mcut.convex_index()); !cv.finished(); ++cv) {
  scalar_type dmin=0, d;
  base_node Pmin,P;
  for (unsigned i=0; i < mcut.nb_points_of_convex(cv); ++i) {
    P = mcut.points_of_convex(cv)[i];
    d = gmm::vect_norm2(ls_function(P));
    if (d < dmin || i == 0) { dmin = d; Pmin = P; }
  }
```

```
    if (dmin < 1e-5)
      nrefine[cv] = nn*8;
    else if (dmin < .1)
      nrefine[cv] = nn*2;
    else nrefine[cv] = nn;
    if (dmin < .01)
      cout << "cv: "<< cv << ", dmin = " << dmin << "Pmin=" << Pmin << " " << nrefine[cv] << "\n";
      }
      {
    getfem::mesh_slicer slicer(mcut);
    getfem::slicer_build_mesh bmesh(mcut_refined);
    slicer.push_back_action(bmesh);
    slicer.exec(nrefine, getfem::mesh_region::all_convexes());
      }
      /*
      sl.build(mcut,
      getfem::slicer_build_mesh(mcut_refined), nrefine);*/

      getfem::mesh_im mim_refined(mcut_refined);
      mim_refined.set_integration_method(getfem::int_method_descriptor
                    ("IM_TRIANGLE(6)"));

      getfem::mesh_fem mf_refined(mcut_refined, Q);
      mf_refined.set_classical_discontinuous_finite_element(2, 0.0001);
      plain_vector W(mf_refined.nb_dof());

      getfem::interpolation(p.mf_u(), mf_refined, U, W);

#ifdef VALIDATE_XFEM
      p.exact_sol.mf.set_qdim(Q);
      assert(p.exact_sol.mf.nb_dof() == p.exact_sol.U.size());
      plain_vector EXACT(mf_refined.nb_dof());
      getfem::interpolation(p.exact_sol.mf, mf_refined,
                p.exact_sol.U, EXACT);

      plain_vector DIFF(EXACT); gmm::add(gmm::scaled(W,-1),DIFF);
#endif

    ////////////////////////////////////////////////
    // compute the strain

    size_type N = p.mf_u().get_qdim();
    cout << "Test" << N << '\n';
```

```
    plain_vector vStr(N*mf.nb_dof());
    getfem::compute_gradient(mf, mf, V, vStr);

    // end comp strain
    ////////////////////////////////////////////////////

      if (p.PARAM.int_value("VTK_EXPORT")) {
    getfem::mesh_fem mf_refined_vm(mcut_refined, 1);
    mf_refined_vm.set_classical_discontinuous_finite_element(1, 0.0001);
    cerr << "mf_refined_vm.nb_dof=" << mf_refined_vm.nb_dof() << "\n";
    plain_vector VM(mf_refined_vm.nb_dof());

    cout << "computing von mises\n";
    getfem::interpolation_von_mises(mf_refined, mf_refined_vm, W, VM);

    plain_vector D(mf_refined_vm.nb_dof() * Q),
      DN(mf_refined_vm.nb_dof());

#ifdef VALIDATE_XFEM
    getfem::interpolation(mf_refined, mf_refined_vm, DIFF, D);
    for (unsigned i=0; i < DN.size(); ++i) {
      DN[i] = gmm::vect_norm2(gmm::sub_vector(D, gmm::sub_interval(i*Q, Q)));
    }
#endif

    cout << "export to " << p.datafilename + ".vtk" << "..\n";
    getfem::vtk_export exp(p.datafilename + ".vtk",
                   p.PARAM.int_value("VTK_EXPORT")==1);

    exp.exporting(mf_refined);
    //exp.write_point_data(mf_refined_vm, DN, "error");
    exp.write_point_data(mf_refined_vm, VM, "von mises stress");

    //exp.write_point_data(mf_refined, W, "elastostatic_displacement");

#ifdef VALIDATE_XFEM

    plain_vector VM_EXACT(mf_refined_vm.nb_dof());

    /* getfem::mesh_fem_global_function mf(mcut_refined,Q);
       std::vector<getfem::pglobal_function> cfun(4);
       for (unsigned j=0; j < 4; ++j)
       cfun[j] = getfem::isotropic_crack_singular_2D(j, p.ls);
       mf.set_functions(cfun);
```

```
      getfem::interpolation_von_mises(mf, mf_refined_vm, p.exact_sol.U,
      VM_EXACT);
    */

    getfem::interpolation_von_mises(mf_refined, mf_refined_vm, EXACT, VM_EXACT);
    getfem::vtk_export exp2("crack_exact.vtk");
    exp2.exporting(mf_refined);
    exp2.write_point_data(mf_refined_vm, VM_EXACT, "exact von mises stress");
    exp2.write_point_data(mf_refined, EXACT, "reference solution");

#endif

    cout << "export done, you can view the data file with (for example)\n"
      "mayavi -d " << p.datafilename << ".vtk -f "
      "WarpVector -m BandedSurfaceMap -m Outline\n";
      }

#ifdef VALIDATE_XFEM
      cout << "L2 ERROR:"<< getfem::asm_L2_dist(p.mim, p.mf_u(), U,
                      p.exact_sol.mf, p.exact_sol.U)
       << endl << "H1 ERROR:"
       << getfem::asm_H1_dist(p.mim, p.mf_u(), U,
                p.exact_sol.mf, p.exact_sol.U) << "\n";

      /* cout << "OLD ERROR L2:"
     << getfem::asm_L2_norm(mim_refined,mf_refined,DIFF)
     << " H1:" << getfem::asm_H1_dist(mim_refined,mf_refined,
     EXACT,mf_refined,W)  << "\n";

     cout << "ex = " << p.exact_sol.U << "\n";
     cout << "U  = " << gmm::sub_vector(U, gmm::sub_interval(0,8)) << "\n";
      */
#endif
    }
  }
  DAL_STANDARD_CATCH_ERROR;

  return 0;
}
```

B.1.2 Parameter file

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% parameters for crack program                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%% pde parameters :                                    %%%%%
MU = 1.0;           % Lam coefficient.
LAMBDA = 1.0;       % Lam coefficient.
MUINC = 100.0;          % Lam coefficient.
LAMBDAINC = 100.0;      % Lam coefficient.
BIMATERIAL = 1;         % 1 : To enable the bimaterial case.
CUTOFF_FUNC = 0.2; CUTOFF1 = 0.2; CUTOFF0 = 0.2; LAMBDA_UP = 0.2; LAMBDA_DOWN = 0.2;

%%%%%   discretisation parameters  :                    %%%%%
MESH_TYPE = 'GT_PK(2,1)';         % linear triangles
%MESH_TYPE = 'GT_LINEAR_QK(2,1)'; % linear rectangles
%MESH_TYPE = 'GT_PRISM(3,1)';     % 3D prisms
NX = 10;                          % space step.
MESH_NOISE = 0; % Set to one if you want to "shake" the mesh
%FEM_TYPE = 'FEM_PK_WITH_CUBIC_BUBBLE(2, 2)';
FEM_TYPE = 'FEM_PK(2, 1)';  % PK element
%FEM_TYPE = 'FEM_QK(2,1)';  % Q1 fem for quadrangles
%FEM_TYPE = 'FEM_HERMITE_SEGMENT'; % (broken) Hermite fem on a segment
%FEM_TYPE = 'FEM_PK_HIERARCHICAL(2,2)'; % Hierarchical PK on simplexes
MIXED_PRESSURE=0;       % Mixed version or not.
% FEM_TYPE_P = 'FEM_PK(3,1)';  % P1 for triangles
FEM_TYPE_P = 'FEM_PK_DISCONTINUOUS(2,0)';  % Discontinuous P1 for triangles
DIRICHLET_WITH_MULTIPLIERS = 0;

% DATA_FEM_TYPE must be defined if your main FEM is not Lagrangian
DATA_FEM_TYPE = 'FEM_PK(2,1)';
% DATA_FEM_TYPE = 'FEM_PK(2,2)';

%INTEGRATION = 'IM_TETRAHEDRON(6)'; % quadrature rule for polynomials up
                                    % to degree 6 on tetra
SIMPLEX_INTEGRATION = 'IM_STRUCTURED_COMPOSITE(IM_TRIANGLE(6), 5)'; INTEGRATION =
'IM_STRUCTURED_COMPOSITE(IM_TRIANGLE(6), 5)';
%INTEGRATION = 'IM_EXACT_SIMPLEX(2)'; % exact integration on triangles
%INTEGRATION = 'IM_NC(2,6)';     % newton-cotes of degree 6 on triangles
%SIMPLEX_INTEGRATION = 'IM_TRIANGLE(6)';
%INTEGRATION = 'IM_TRIANGLE(6)';
ADDITIONAL_CRACK = 0;
ENRICHMENT_OPTION = 2;  % 0 = Pas d'enrichissement
                    % 1 = global functions with cutoff
                      % 2 = standard XFEM on a fixed zone
                        % 3 = spider fem alone
                    % 4 = spider fem enrichment
MESHNAME='meshes/poly-035.msh';
```

```
RADIUS_ENR_AREA = 0.1; CUTOFF=0.2;
% RADIUS_ENR_AREA = 0.2;
SPIDER_RADIUS =  0.3;
SPIDER_NR = 20;          % size of the cartesian mesh in r for spider fem
SPIDER_NTHETA = 20;       % size of the cartesian mesh in theta for spider fem
SPIDER_K=1;              % order of the  spider fem
RESIDUE = 1E-9;          % residue for iterative methods if any.

%%%%%   saving parameters                                        %%%%%
ROOTFILENAME = 'crack';     % Root of data files.
VTK_EXPORT = 1 % export solution to a .vtk file ?
```

REFERENCES

[1] BABUŠKA, I. and MELENK, J., "Partition of unity method," *International Journal for Numerical Methods in Engineering*, vol. 40, pp. 727–758, 1997.

[2] BECHET, E., MINNEBO, H., MOES, N., and BURGARDT, B., "Improved implementation and robustness study of the x-fem for stress analysis around cracks," *International Journal for Numerical Methods in Engineering*, vol. 64, pp. 1033–1056, 2005.

[3] BELYTSCHKO, T. and BLACK, T., "Elastic crack growth in finite elements with minimal remeshing," *International Journal for Numerical Methods in Engineering*, vol. 45, pp. 601–620, 1999.

[4] BELYTSCHKO, T., CHEN, H., XU, J., and ZI, G., "Dynamic crack propagation based on loss of hyperbolicity and a new discontinuous enrichment," *International Journal for Numerical Methods in Engineering*, vol. 58, pp. 1873–1905, 2003.

[5] BELYTSCHKO, T., MOES, N., USUI, S., and PARIMI, C., "Arbitrary discontinuities in finite elements," *International Journal for Numerical Methods in Engineering*, vol. 50, pp. 993–1013, 2001.

[6] CHESSA, J. and BELYTSCHKO, T., "Arbitrary discontinuities in spacetime finite elements by level sets and x-fem," *International Journal for Numerical Methods in Engineering*, vol. 61, pp. 2595–2614, 2004.

[7] CHOPP, D. and SUKUMAR, N., "Fatigue crack propagation of multiple coplanar cracks with the coupled extended finite element, fast marching method," *International Journal of Engineering Science*, vol. 41, pp. 845–869, 2003.

[8] DAUX, C., MOES, N., MORAN, B., and BELYTSCHKO, T., "Arbitrary branched and intersecting cracks with the extended finite element method," *International Journal for Numerical Methods in Engineering*, vol. 48, pp. 1741–60, 2000.

[9] DE BORST, R., GUTIERREZ, M., WELLS, G., REMMERS, J., and ASKES, H., "Cohesive-zone models, higher-order continuum theories and reliability methods for computational failure analysis," *International Journal for Numerical Methods in Engineering*, vol. 60, pp. 289–315, 2004.

[10] DOLBOW, J., MOES, N., and BELYTSCHKO, T., "Modelling fracture in mindlinreissner plates with the extended finite element method," *International Journal for Solids Structure*, vol. 37, pp. 7161–83, 2000.

[11] DOLBOW, J., MOES, N., and BELYTSCHKO, T., "An extended finite element method for modeling crack growth with frictional contact," *Computer Methods in Applied Mechanics and Engineering*, vol. 190, pp. 6825–46, 2001.

[12] DONAHUE, R. and FABIYI, P., "Manufacturing feasibility of all- aluminum automotive engines via application of high silicon aluminum alloy," *SAE Transactions Journal of Materials and Manufacturing*, vol. 109, pp. 16–26, 2000.

[13] FAN, S., LIU, X., and LEE, C., "Enriched partition-of-unity finite element method for stress intensity factors at crack tips," *Computational Mechanics*, vol. 82, pp. 445–461, 2004.

[14] FLEMING, M., CHU, Y., MORAN, B., and BELYTSCHKO, T., "Enriched element-free galerkin methods for crack-tip fields," *International Journal for Numerical Methods in Engineering*, vol. 40, pp. 1483–504, 1997.

[15] GRAVOUIL, A., MOES, N., and BELYTSCHKO, T., "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*, vol. 53, pp. 2569–2586, 2002.

[16] HEINTZ, P., "On the numerical modeling of quasi-static crack growth in linear elastic fracture mechanics," *Chalmers Finite Element Center*, vol. 2005-02, pp. 1–18, 2005.

[17] IARVE, E., "Mesh independent modelling of cracks by using higher order shape functions," *International Journal for Numerical Methods in Engineering*, vol. 56, pp. 869–882, 2003.

[18] LABORDE, P., POMMIER, J., RENARD, Y., and SALAUN, M., "High-order extended finite element method for cracked domains," *International Journal for Numerical Methods in Engineering*, vol. 64, pp. 351–381, 2005.

[19] LEE, S., SONG, J., YOON, Y., ZI, G., and BELYTSCHKO, T., "Combined extended and superimposed finite element method for cracks," *International Journal for Numerical Methods in Engineering*, vol. 59, pp. 1119–1136, 2004.

[20] LIANG, J., HUANG, R., PREVOST, J., and SUO, Z., "Evolving crack patterns in thin films with the extended finite element method," *International Journal of Solids and Structures*, vol. 40, pp. 2343–2354, 2003.

[21] LIU, X., XIAO, Q., and KARIHALOO, B., "Xfem for direct evaluation of mixed mode sifs in homogeneous and bi-materials," *International Journal for Numerical Methods in Engineering*, vol. 59, pp. 1103–1118, 2004.

[22] MELENK, J. and BABUŠKA, I., "The partition of unity finite element method: basic theory and applications," *Computer Methods in Applied Mechanics and Engineering*, vol. 139, pp. 289–314, 1996.

[23] MOES, N., DOLBOW, J., and BELYTSCHKO, T., "A finite element method for crack growth without remeshing," *International Journal for Numerical Methods in Engineering*, vol. 46, pp. 131–50, 1999.

[24] MOES, N., GRAVOUIL, A., and BELYTSCHKO, T., "Non-planar 3d crack growth by the extended finite element and level sets. part i: mechanical

model," *International Journal for Numerical Methods in Engineering*, vol. 53, pp. 2549–2568, 2002.

[25] OSHER, S. and SETHIAN, J., "Fronts propagating with curvature dependent speed: algorithms based on hamiltonjacobi formulations," *Journal of Computer Physics*, vol. 79, pp. 12–49, 1988.

[26] RENARD, Y. and POMMIER, J., *getfem++, A generic finite element library in C++*. INSAT, Toulouse, France, 2005.

[27] RIBO, R. and PASENAU, M., *GiD, The personal pre and post processor.* CIMNE, Brcelona, Spain, 2005.

[28] RUBINSTEIN, A., "Computational aspects of crack path development simulation in materials with nonlinear process zone," *International Journal of Fracture*, vol. 119, pp. L15–L20, 2003.

[29] SETHIAN, J., *Level set methods and fast marching methods.* Cambridge University Press, 1999.

[30] STAZI, F., BUDYN, E., CHESSA, J., and BELYTSCHKO, T., "An extended finite element method with higher-order elements for curved cracks," *Computational Mechanics*, vol. 31, pp. 38–48, 2003.

[31] STOLARSKA, M., CHOPP, D., MOES, N., and BELYTSCHKO, T., "Modelling crack growth by level sets in the extended finite element method," *International Journal for Numerical Methods in Engineering*, vol. 51, pp. 943–60, 2001.

[32] SUKUMAR, N., MOES, N., MORAN, B., and BELYTSCHKO, T., "Extended finite element method for three-dimensional crack modelling," *International Journal for Numerical Methods in Engineering*, vol. 48, pp. 1549–70, 2000.

[33] VENTURA, G., BUDYN, E., and BELYTSCHKO, T., "Vector level sets for description of propagating cracks in finite elements," *International Journal for Numerical Methods in Engineering*, vol. 58, pp. 1571–1592, 2003.

[34] XIAO, Q. and KARIHALOO, B., "Direct evaluation of accurate coefficients of the linear elastic crack tip asymptotic field," *Fatique and Fracture of Engineering Materials and Structures*, vol. 26, pp. 719–730, 2003.

VITA AUCTORIS

Name: Siamak Tavoosfard

Place of Birth: Shiraz, Iran

Year of Birth: 1965

Education: Kherad High School, Shiraz 1979-1983

Iran University of Science and Technology, Tehran 1984-1988 B.Sc

Humber College of Art and Technology, Toronto, Ontario 1998-2000

University of Windsor, Windsor, Ontario 2004-2006 M.Sc