

Stochastic Fatigue Failure Prediction of Adhesive Bonded Joints

by
Sergio Candelario

A thesis submitted in partial fulfillment of the requirements for the degree of

Master in Science
in
Mechanical Engineering

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS

May 2017

Approved by:

David Serrano, Sc.D.
President, Graduate Committee

Date

Ricky Valentín, Ph.D.
Member, Graduate Committee

Date

Paul Sundaram, Ph.D.
Member, Graduate Committee

Date

Paul Sundaram, Ph.D.
Chairperson of the Department

Date

Roberto Seijo, Ph.D.
Representative of Graduate Studies

Date

A Dissertation
by Sergio Candelario

The author used his best effort in preparing this thesis. These efforts include the development, research, and testing of the theories and formulas to determine their effectiveness in fatigue failure prediction of adhesive bonded joints. The author shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these formulas.

Type Set in Times Roman 12 pt using L^AT_EX.

Stochastic Fatigue Failure Prediction of Adhesive Bonded Joints

Sergio Candelario

Master in Science in Mechanical Engineering

University of Puerto Rico at Mayagüez

Dr. David Serrano, Faculty Advisor, Mechanical Engineering

(ABSTRACT)

This work describes the implementation of the Extended Finite Element Method (a partition of unity finite element method) for the study of fatigue failure of adhesive bonded joints when subjected to variable loading. The main advantage of the Extended Finite Element Method (XFEM) is the independence of the finite element mesh to describe the delamination hence, it eliminates the need of re-meshing when the delamination front is propagated. This advantage is particularly useful when modeling delamination under fatigue as no remeshing is needed for each loading cycle. Also in this work, the Extended Finite Element Method is extended to include incompatible elements, with the addition of internal degrees of freedom that allow nonlinear mathematical distortion of a four node bilinear element. These elements are used to model a composite double cantilever beams to study fatigue delamination due to fatigue under random loading. The Yang-Manning's stochastic model for fatigue delamination was modified and good agreement with experimental data was observed.

Predicción de Falla en Fatiga Estocástica de Uniones Adhesivas

Sergio Candelario

Maestría en Ciencia en Ingeniería Mecánica

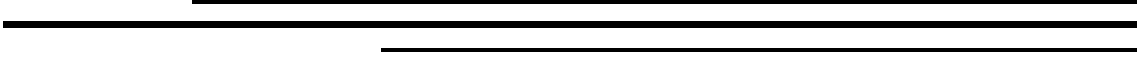
Universidad de Puerto Rico en Mayagüez

Dr. David Serrano, Profesor Consejero, Ingeniería Mecánica

(RESUMEN)

Este trabajo describe la implementación del Método de Elementos Finitos Extendidos (un método de partición de unidad en elementos finitos) para el estudio de la falla en fatiga de las juntas en unión adhesiva cuando se someten a cargas aleatorias. La principal ventaja del Método de Elementos Finitos Extendidos (XFEM) es la independencia de la malla de elementos finitos para describir la delaminación, por lo tanto, elimina la necesidad de generar una malla nueva cada vez que se propaga el frente de delaminación. Esta ventaja es particularmente útil cuando se modela la delaminación bajo fatiga, ya que no se necesita generar una malla para cada ciclo de carga. También en este trabajo, el Método de Elementos Finitos Extendidos se amplía para incluir elementos incompatibles, que es la adición de grados internos de libertad que permiten la distorsión matemática no lineal de un elemento bilineal de cuatro nodos. Estos elementos se usan para modelar una doble viga en voladizo de material compuesto para estudiar la delaminación por fatiga debido a la fatiga en carga aleatoria. Se modificó el modelo estocástico de Yang-Manning para la delaminación por fatiga y se observó un buen acuerdo con los datos experimentales.

Dedication



To my family, my girlfriend Frances, and above all to God.

This page has been intentionally left blank.

Acknowledgments

First and foremost, I am grateful to God for helping, guiding and providing the strength through this journey. Next, I would like to thank my parents, Guillermo and Marisol, who supported me through these years. Also, to María S. Vázquez, mother of Frances, who also supported me through these years. But my greatest strength came from Frances M. Guerrero, my girlfriend from many years who helped me go through this journey.

I am grateful for Dr. Goyal's dedication during the research leading to this thesis. Dr. Goyal supervised much of the work presented here. After his departure from the University, Dr. Serrano kindly stepped in, supervising the final stages of the thesis defense. In order to recognize Dr. Goyal's efforts in this research endeavor the author will list him as the second co-author of any forthcoming publications of the work.

I want to thank the Department of Mechanical Engineering of the University of Puerto Rico (UPRM) for this opportunity and support through the years and for providing me with the Graduate Teaching Assistantship during the first years of my graduate studies.

I am thankful for my graduate committee members Dr. Serrano, Dr. Valentín and Dr. Sundaram for their support. Lastly, I would like to thank all my colleagues at the Mechanical Engineering Department for their support.

Sergio Candelario

This page has been intentionally left blank.

Table of Contents

List of Figures	xi
List of Tables	xiii
List of Symbols	xiv
Chapter 1. Preliminary Remarks	1
1.1 Motivation	1
1.1.1 Current State of the art and downfalls	1
1.1.2 The Need for a New Computational Technique	4
1.2 Project Description	6
1.2.1 Problem Description	6
1.2.2 Assumptions	7
1.2.3 Overall Goals	7
1.2.4 Intellectual Merit	8
1.2.5 Broader Impacts	8
1.3 Approach	9
1.3.1 Technical Approach	9
1.3.2 Overview	11
1.3.3 Thesis Outline	12
Chapter 2. Numerical Simulation Details	13
2.1 Modeling adhesive bonded joints	13
2.2 Finite element formulation	14
2.2.1 Incompatible elements	17
2.3 The Extended Finite Element formulation	18
2.3.1 Discontinuity field description	19
2.3.2 Level set method	22
2.3.3 Blending elements	24
2.3.4 Degrees of freedom	25

2.3.5	Element integration	26
2.4	Fracture Mechanics	28
2.4.1	Crack tip near fields	30
2.5	Case study: Double Cantilever Beam	32
2.5.1	Analytical solution	32
2.5.2	Convergence study	33
Chapter 3.	Implementation and Results	36
3.1	Fracture mechanics for the adhesive	36
3.1.1	Fatigue and crack propagation	37
3.1.2	Stochastic fatigue model	37
3.1.3	Simulation by loading cycles	38
3.1.4	Code structure	40
3.2	Benchmark example: graphite epoxy DCB	42
3.2.1	Constant amplitude loading	42
3.2.2	Implementation with random loading	47
3.2.3	Evaluation of load variation	50
3.2.4	Evaluation of seed influence in random data generation	55
Chapter 4.	Final Remarks	57
4.1	Conclusion	57
4.2	Recommendations	58
4.3	Future Work	59
Appendix A.	MATLAB scripts	60
References		144
Vita		150

List of Figures

Figure 1.1	Adhesive bonded joint example	2
Figure 1.2	XFEM mesh representation	5
Figure 1.3	Mesh comparison between XFEM and FEM	6
Figure 1.4	2D analysis of a double cantilever beam	10
Figure 1.5	Overview of the research process	12
Figure 2.1	Some examples of adhesive bonded joints	14
Figure 2.2	Body with prescribed displacement and loads	15
Figure 2.3	Node coordinates for the quadrilateral element in the computa- tional space	17
Figure 2.4	Q4 element in bending	18
Figure 2.5	Crack coordinate convention	21
Figure 2.6	Level sets	22
Figure 2.7	Enrichment space example	24
Figure 2.8	Example of element sub-triangulation	26
Figure 2.9	Example of element partitioning using sub-quadrilaterals	28
Figure 2.10	J integral area of integration	30
Figure 2.11	Fracture modes	31
Figure 2.12	Double cantilever beam in displacement control	33
Figure 2.13	Energy release rate mesh convergence test	34
Figure 2.14	Mesh example for convergence study	35
Figure 3.1	Loading history example	39
Figure 3.2	Flowchart for crack propagation	41
Figure 3.3	Test configuration	42
Figure 3.4	Mesh for fatigue simulation of DCB (initial delamination in red color)	44
Figure 3.5	Delamination growth rate comparison with experimental data . . .	45
Figure 3.6	Von Mises stresses contour plot (Constant amplitude loading) . . .	46
Figure 3.7	Delamination length results per cycles for constant amplitude stochas- tic fatigue	47
Figure 3.8	Delamination growth rate comparison with random loading	48

Figure 3.9	Delamination length results per cycle for random amplitude load stochastic fatigue	49
Figure 3.10	Energy release rate for low deviation test case	50
Figure 3.11	Energy release rate for medium deviation test case	51
Figure 3.12	Energy release rate for high deviation test case	52
Figure 3.13	Final cycles to failure for low deviation test case	53
Figure 3.14	Final cycles to failure for medium deviation test case	54
Figure 3.15	Final cycles to failure for high deviation test case	55
Figure 3.16	Cycles to failure prediction comparison by seed	56

List of Tables

Table 1.1	Literature survey overview of XFEM implementations in the analysis of adhesive bonded joints	3
Table 1.2	Commercial softwares comparison	4
Table 3.1	Material properties for unidirectional Graphite/Epoxy Prepreg . .	43
Table 3.2	Material properties for epoxy resin	43
Table 3.3	Benchmark loading parameters	43
Table 3.4	Geometry of DCB for benchmark example	43
Table 3.5	One-way ANOVA for comparison of seed variation	56

List of Symbols

H	Heaviside enrichment function
F^k	Near-tip asymptotic field enrichment function
u	Displacement (standard finite element) degree of freedom
a	Heaviside degree of freedom
b^k	Near-tip degree of freedom for the k -th function
ζ	Signed distance function
$\phi(x, t)$	Crack orthogonal level set
$\psi(x, t)$	Crack normal level set
ξ	Abcissa local coordinate in the reference element space
η	Ordinate local coordinate in the reference element space
$\mathbf{N}(\xi, \eta)$	Element shape function matrix
$\mathbf{B}(\xi, \eta)_{mp}$	Strain-displacement matrix (shape function derivatives); m and p indexes represent the u , a and b degrees of freedom
\mathbf{J}	Jacobian matrix
σ	Stress vector
ε	Strain tensor
$k\kappa$	Kolosov's constant
E	Young's modulus
$k\mu$	Shear modulus
\mathbf{K}	Stiffness matrix
$R(\xi, \eta)$	Ramp function
K_I, K_{II}	Stress intensity factors for mode I and II respectively

G	Energy release rate
\mathbf{D}	Material stiffness matrix
β	Loading ratio
Δa	Delamination increment
ΔN	Cycles in a loading block
$\frac{da}{dN}$	Delamination growth rate
J	J-contour integral (elastic-plastic parameter)

Chapter 1

Preliminary Remarks

This work pertains to the application of the Extended Finite Element Method (XFEM) to predict fatigue failure of adhesive bonded joints when subjected to cyclic loading. The Extended Finite Element Method is coupled with the Level Set Method (LSM) and Linear Elastic Fracture Mechanics (LEFM) theory to describe the delamination of the adhesive joint. The main goal of this work is to develop a numerical tool to describe the fatigue failure prediction of adhesive bonded joints using the Extended Finite Element Method. This methodology is developed in MATLAB which, although a high level language, is widely known and used by engineers and researchers.

1.1 Motivation

1.1.1 Current State of the art and downfalls

Currently, aerospace industries are opting to utilize composites materials to build aircraft components and fuselages. The next generation of aircraft will probably be all built out of composites because of their higher strength to weight ratio compared to the metallic materials currently used. Nonetheless, these materials have to pass regulations and certifications which increase costs. This situation drives the motivation to replace the numerous expensive testing methods with reliable and accurate predictive models (Lord and Ngah 2005)

Several methods exist for joining or fastening materials together to form a structure e.g. welding, bolting, riveting, etc. When compared to some of these methods, the adhesion of materials provides both economical and performance advantage. These advantages range from the ability to form lightweight joints, the possibility to join dissimilar materials and improved stress distribution, which can improve fatigue life, among other benefits (Tong and Steven 1999). Due to these benefits, naval and aerospace industries

are utilizing adhesive bonded joints more. As an example, such as the Boeing 787 Dreamliner commercial passenger aircraft uses 50% advanced composites. And the Boeing 777 proved that composite structures require less scheduled maintenance in comparison with non-composite materials (Boeing Commercial Airplanes 2006). Hence, with the increase in use of composites and adhesive joints, more efficient ways to model damage in adhesive bonded materials is of great importance.

An efficient predictive model must conform to the shapes and configurations of the parts designed in the aerospace industry. In the early stages of bonded structure analyses, theoretical studies were popular among scholars (Campilho et. al 2011). The analytical methods had the advantage of analysing a structure quickly but comprised multiple assumptions which make them inadequate to accurately predict complex real life situations. Because of this, the Finite Element Method is a more attractive method of simulation.

Composite structures are commonly comprised of layers of adherents bonded by an adhesive to form a solid structure. The adhesive bonds are advantageous because they distribute the loads over a wider area and therefore, the stresses over the bonded area are less critical when compared to other fastening techniques (Frostig et. al 1999 and Hart et. al 2002). Therefore, the stress distribution on the adherents is improved by using adhesive bonded joints which avoids point stress concentrations. Figure 1.1 shows a schematic of the adherent and the adhesive interacting with each other.

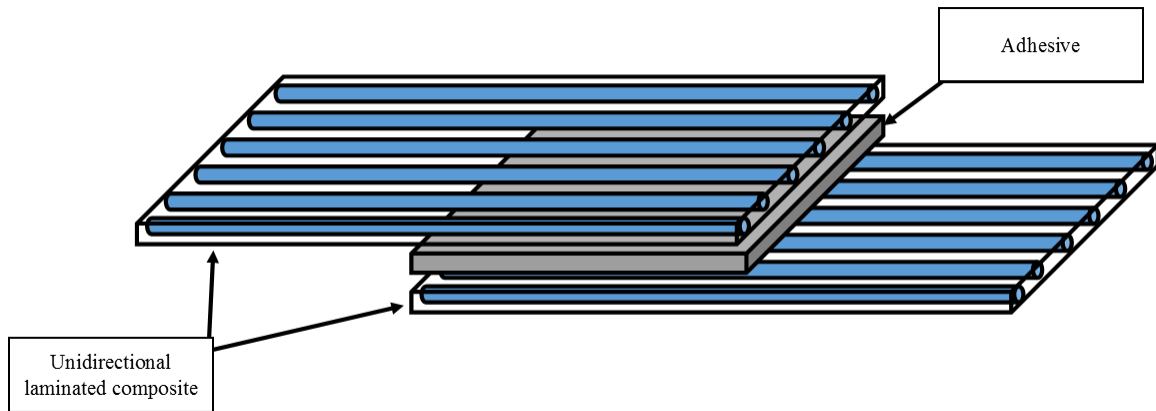


Figure 1.1: Adhesive bonded joint example

Delamination is the separation of the adherents due to cracking of the adhesive and is one of the major failure modes encountered in composites (de Borst and Remmes

2006). The majority of the composite materials use some type of adhesive that when dry, behaves in a brittle manner. It is known that brittle materials do not yield but fail rapidly for which reason this type of failure is of much concern. Nevertheless, failure of the adhesive layer does not mean that the whole composite structure has failed but rather that it degrades the reliability of the component or structure. Therefore, a predictive model for failure of composite structures must take into account failure of the adhesive layer.

A common test to determine the strength of a bond is by using double cantilever beam or single lap joints among others. Some examples of joint classifications are single and double slab, single cover plate or double cover plate (See Figure 2.1). In this figure, the arrows represent the direction of pull on the tests. In this configuration the adherent is being sheared or a Mode II or shear stress condition is created (Figure 2.1, 2a and 2b) but other configurations such as simple tension (Figure 2.1, 2c), compression or a mixed mode in the adherent (Mode I, Mode III and mixed) can also be developed and tested.

Several researchers have devoted time in the implementation of new FEM techniques for the modeling of adhesive bonded joints. Campilho et. al 2011 used the XFEM with strong enrichment functions (refer to Section 2.3.1) to test adhesive strength in a double cantilever beam analysis. de Borst and Remmers 2006 studied the delamination of Glare in a DCB using a cohesive interphase to model the adhesive layer. Motamedi et. al. 2013 studied the delamination process of Polyphenylene sulfide/Glass fiber reinforced polymer DCB and performed a 3D analysis in Abaqus/MATLAB. A summary of the current XFEM implementations in the analysis of adhesive bonded joints and model capabilities is provided in Table 1.1.

Table 1.1: Literature survey overview of XFEM implementations in the analysis of adhesive bonded joints

Authors	Fatigue analysis	Near-tip field	Stochasticity	Cohesive model	3D analysis
de Borst and Remmers 2006				✓	
Campilho et. al 2011				✓	
Campilho et. al 2011				✓	
Motamedi et. al 2013			✓	✓	✓
Motamedi et. al 2014			✓	✓	✓
Sosa and Karapurath 2012		✓			
This dissertation	✓	✓	✓		

The fatigue failure is known to be statistical in nature (Wu and Ni 2003). Some

commercially available finite element programs are capable of performing probabilistic analysis. For instance, LSTC's LS-OPT software is a standalone design optimization and probabilistic analysis package that can interface with LS-DYNA and ANSYS has released the Probabilistic Design System and the ANSYS DesignXplorer tools. However, not all finite element commercial softwares are capable of performing a probabilistic analysis as shown in Table 1.2.

Table 1.2: Commercial softwares comparison

Softwares	XFEM	Fatigue analysis	Near-tip field growth simulation	Probabilistic capabilities	3D analysis
LS DYNA		✓		✓	✓
ANSYS	(plug-in)	✓		✓	✓
ABAQUS	✓	✓			✓
Nastran		✓			✓
This dissertation	✓	✓	✓	✓	

1.1.2 The Need for a New Computational Technique

The Extended Finite Element Method (XFEM) originated with the work by Belytschko and Black in 1999 in which discontinuous functions were added to the finite element approximation (trial or interpolation functions) to include the presence of the crack into the finite element (refer to Equation 2.16). In their work, the authors applied an extrinsic approximation for the addition of the discontinuous (enrichment) functions. Their method was further enhanced by the works of Moës et. al 1999 with the inclusion of enrichment functions to model the discontinuous field of the crack away from the crack tip (Haar function) thus incorporating both Crack and Near-tip enrichment functions; see also Dolbow et. al 2000; refer to Figure 1.2.

Several advantages of the XFEM over FEM can be identified:

- mesh boundaries do not have to coincide with the discontinuity (i.e. discontinuous fields can be modeled within an element), refer to Figure 1.3; (Moës et. al 1999)
- discontinuities are modeled independently from the mesh thus there is no need for re-meshing for evolving discontinuities (e.g propagating cracks); (Belytschko and Black 1999)

- due to the inclusion of known analytical solutions to the shape functions, accurate finite element solutions can be achieved even for relatively coarse meshes (e.g asymptotic stress near a crack tip can be modeled with a relatively coarse mesh); (Moës et. al 1999)

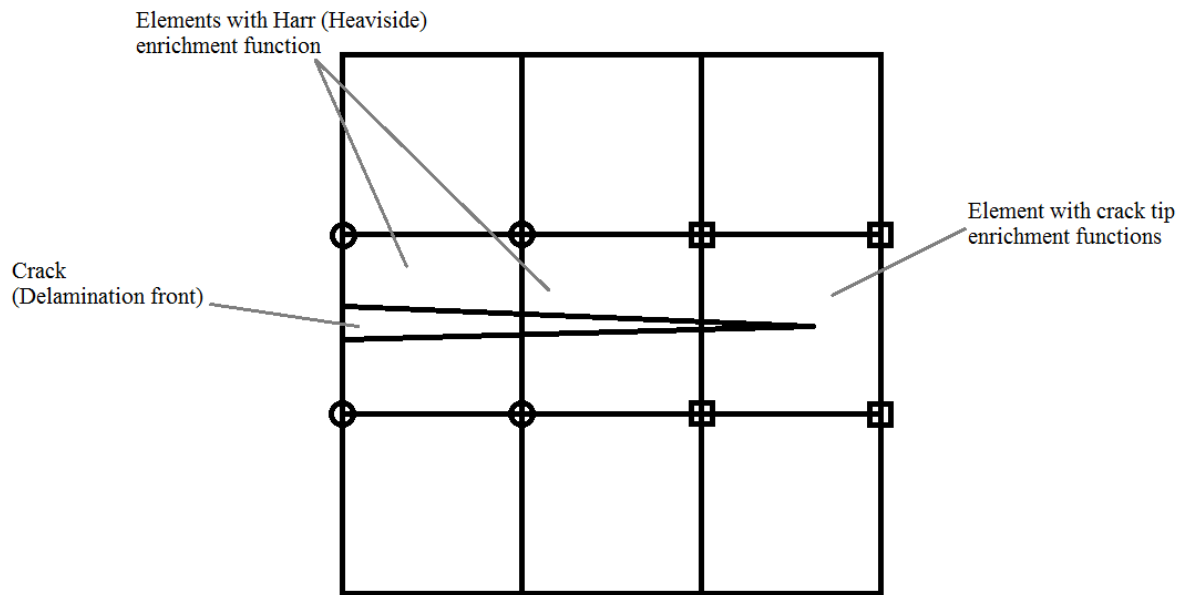


Figure 1.2: XFEM mesh representation

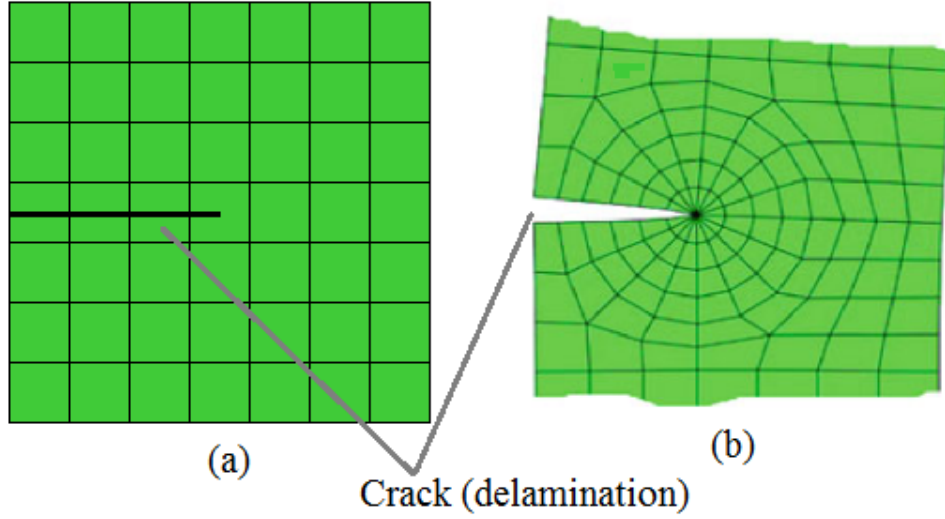


Figure 1.3: (a) XFEM mesh, delamination is embedded within the mesh; (b) FEM mesh, delamination is modeled between element boundaries (from Kuna 2013)

However, current commercial implementations of the XFEM do not take into account the near-tip enrichment functions during crack propagation. A model which takes into account the random nature of fatigue failure and random loading in composite adhesive bonded joints have not been coupled with the XFEM. Moreover, in this work, a complete energetic approach for delamination characterization has been implemented (i.e. fracture based on energy release rate). Hence, a model that incorporate these features will be attractive as it provides a more realistic approach to fatigue simulation.

1.2 Project Description

1.2.1 Problem Description

The composite adhesive bonded joints are modeled as orthotropic adherends (plies) adhered by an isotropic linear elastic adhesive layer. The load is applied to the adherends with constant magnitude for one loading cycle. The simulation is then repeated several times to simulate cyclic loading under service. For variable amplitude loading, the amplitude is changed after each loading cycle. Small deformations are imposed into the model; this will allow to describe the deformation using linear relationships, i.e. infinitesimal strain theory.

1.2.2 Assumptions

In order to provide a manageable system for analysis, the following assumptions were made:

- The adhesive material is assumed to be sufficiently brittle as to be able to be modeled as a linear elastic material hence, Linear Elastic Fracture Mechanics (LEFM) theory is applicable for the description of the crack near-field. This assumption implies that the stress field near the crack tip is asymptotic and plastic deformation is confined within a small area near the crack tip (small scale yielding). However, due to the utilization of higher order terms in the William's expansion, the system can deviate from the small scale yielding assumption.
- The adhesive is modeled as a linear elastic material and the crack is allowed to propagate only in the adhesive. By this assumption, a cohesive delamination type of failure is induced and a Mode I type of failure is approximated.
- The composite adherends are modeled as orthotropic linear elastic materials. This assumption reduces the discontinuity present between composite layers and allows the plies to be modeled as a continuum. This reduces the need to model each ply independently and thus simplifying the numerical model.
- The joint transverse dimension is assumed to be very small compared to the joint depth thus, the system is modeled as a two dimensional mesh in plane strain condition in the transverse dimension of the bonded joint. This allows for reduction of a three-dimensional problem into a two-dimensional one, thus simplifying the analysis.

1.2.3 Overall Goals

The goal of this work is to develop an Extended Finite Element Method based algorithm for the modeling of adhesive bonded joint failure under fatigue loading. The following tasks will be achieved in this work:

1. Develop an algorithm to predict the delamination process of a double cantilever beam
 - (a) Include the stochastic nature of the fatigue process into the simulation

- (b) Model the adherends as orthotropic materials to simulate composite construction
- 2. Simulate the delamination process of a double cantilever beam under fatigue loading
 - (a) Understand the effects of fatigue loading under constant and variable (random) loading conditions
 - (b) Perform several experiments with different levels of load variation
- 3. Validate the delamination results with available data in the literature

The development of the computer algorithm will then be provided as a contribution to further develop the Extended Finite Element Method in the scientific community.

1.2.4 Intellectual Merit

Due to the increase in composite adhesive bonded joints in the aerospace industry, there is a high interest in modeling the failure process in these joints. There is a need to study the failure of such joints as there is currently limited knowledge available in their behavior during flight. As these structures are subjected to random loading scenarios during flight, a numerical analysis considering the random nature of such scenarios and the inherent random nature of the fatigue process is highly desirable. The development of an efficient computational technique to study such scenarios will benefit greatly the aerospace industry.

A code that is both efficient for the delamination modeling of adhesive bonded joints in fatigue loading and that takes into account the stochastic nature of the fatigue process has not been developed yet. This work will provide an efficient tool for the simulation of adhesive bonded joints in fatigue loading coupling variable loads and stochastic fatigue modeling. The implemented code is developed in MATLAB as it is a very flexible scripting language that can be coupled with other finite element softwares.

1.2.5 Broader Impacts

The immediate beneficiaries of the work presented here are the scientists in the field of fracture mechanics and material science as the stochasticity of real life bonded joints can be assessed with the developed tool. As the algorithm was developed in MATLAB,

it can easily be coupled with other commercial finite element programs to enhance its capabilities like more complex meshes and faster processing times.

An area of application of this work is in the reduction of test costs during the development process of aircraft parts. A reduction in testing can be done as the developed algorithm is capable of simulating and predicting the fatigue delamination process of adhesive bonded joints. Thus, different levels of loading and fatigue variations can be tested numerically thus reducing the testing size. This will in turn translate into a reduction of costs at the development stage.

1.3 Approach

1.3.1 Technical Approach

It is of interest in this research to develop a tool to predict the fatigue failure of adhesive bonded joints capable of simulation of real life behavior of joint failure under fatigue loading. Therefore, the basic approach taken is as follows:

- Construct an efficient finite element code to run fatigue analysis in an adhesive bonded joint hence, the use of the Extended Finite Element Method is proposed to efficiently model the delamination process without remeshing.
- Account for the stochastic nature of the fatigue in order to simulate real life behavior of the fatigue process
- Evaluate the effects of random loading on cycles to failure of the adhesive bonded joint
- Validate model using published data
- Propose areas of future work

In this work, a double cantilever beam (DCB) composed of composite adherends and epoxy adhesive is analyzed within the framework of the Extended Finite Element Method. A DCB was selected for the analysis as it is a commonly used configuration to study delamination in composites and strength of the adhesive (Banea and da Silva 2009, Biel and Stigh 2007). The DCB will be analyzed in a two dimensional space and

discretized using a Lagrangian mesh. Hence, they will be analyzed in their transverse cross section (i.e. only length and thickness dimensions are meshed) and as such, plane strain conditions are imposed in the analysis (refer to Figure 1.4). The Lagrangian mesh will be enriched with partition of unity functions utilizing in the level set functions to describe the discontinuity fields in the mesh i.e. delamination and delamination front asymptotic field. The delamination will be advanced in each analysis step with a selected delamination increment until the predefined final delamination length, maximum allowable cycles or sudden fracture occurs.

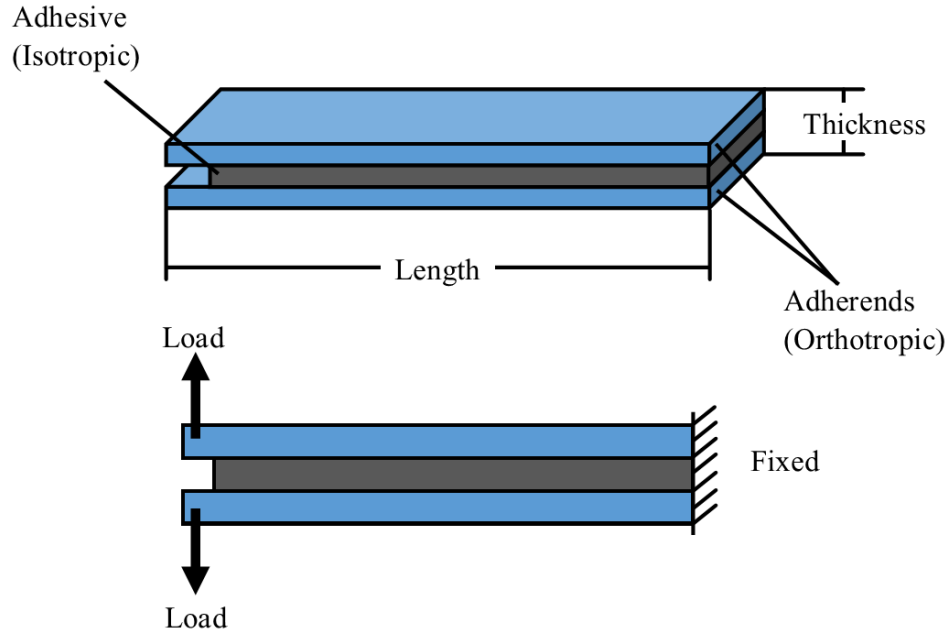


Figure 1.4: 2D analysis of a double cantilever beam

In the algorithm developed, an initial crack is embedded in the adhesive layer. The structure, in this context the DCB, is subjected to prescribed displacements or to random sampled displacements from a normal probability density function. Afterwards, the structure displacements and hence the stresses are found via the Extended Finite Element Method. The crack is then advanced using a modified Paris-Erdogan power model to determine the cycles needed to propagate the crack to the prescribed crack extension. The modified Paris-Erdogan equation is modified via the Yang-Manning's model to induce stochasticity. Comparison with experimental and computer simulation results extracted from the literature is performed to evaluate the validity of the model.

From the above established test case, a computer based algorithm in MATLAB for the delamination prediction of a DCB will be created. From this computer code, the cycles until failure will be calculated and compared with data in the literature. Furthermore, the cycles to failure during random loading will be evaluated for different degrees of relative standard deviations and analyzed to see the impact on the average cycles to failure due to random loading and stochastic fatigue law.

1.3.2 Overview

The first step into the research process for this work was to select the computational technique for analysis. In this study, the XFEM was selected as no remeshing is needed for the simulation thus lowering the computational cost. After the numerical method was selected, a survey of the implementations of the XFEM was performed to study the details of the method, current implementation practices and limitations of the technique.

After general knowledge of the computational technique was attained, the development process for the computer code was started. Initially, bilinear quadrilateral elements were used for the analysis but convergence issues were present thus, instead of increasing the element amount in the analysis, and thus increasing the computational cost, incompatibility elements were introduced. Also, at the beginning of the development process, a sub triangulation technique for integration of the element was utilized. However, its use was discontinued as this technique will require remapping of the integration points during delamination increasing the computational cost.

A graphical representation of the research process is shown in Figure 1.5 below. As shown, the study focuses in the development of a tool for simulation of stochastic fatigue delamination in: 1) constant load simulation and 2) random loading simulation.

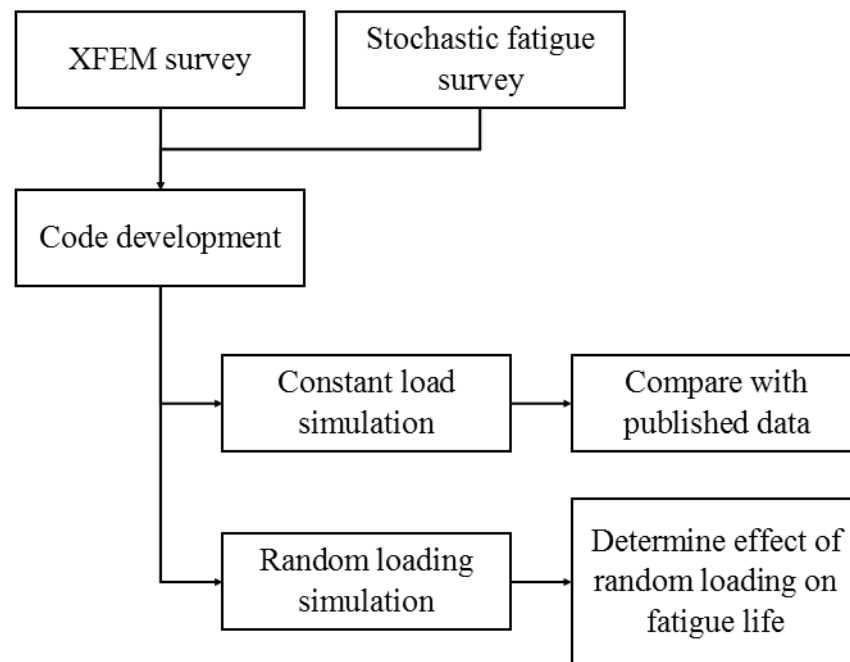


Figure 1.5: Overview of the research process

1.3.3 Thesis Outline

This thesis is divided into four chapters. Chapter 1 is an introduction with an overview of the methodology used, motivation and goals. Chapter 2 provides an in-depth look at the finite element formulation and convergence study. Chapter 3 provides the implementation of the model and comparison of simulation results. Lastly, Chapter 4 contains the conclusion of this work and suggestions to various areas of improvement and future work.

Chapter 2

Numerical Simulation Details

Delamination initiation and propagation can occur at the interface of the adherent and the adhesive due to the high magnitude of stresses developed parallel and perpendicular to the interface. Failures in adhesively bonded joints can occur by peeling or shearing of the adhesive, delamination of an adherent or by tension or compression failure of the adherents. Cohesive failure through the adhesive can also occur due to existing voids or cavities. In this chapter, the mathematical foundations of the Extended and Standard Finite Element Method and fracture theory for the simulation of adhesive bonded joints are provided.

2.1 Modeling adhesive bonded joints

Adhesive bonded joints have been previously studied numerically through the XFEM and experimentally compared using single and double lap joint (see Figure 2.1 (Campilho et al. 2011 and 2011)). Their work was based in coupling the extended finite element method with the cohesive zone model (CZM). However, although the CZM allows for modeling of the fracture phenomena as a degradation process, by virtue of a cohesive law, the cohesive elements must be placed where the delamination is expected hence, the delamination path must be known a priori. Prediction of delamination of bonded joints can be classified into four (4) categories: traditional stress/strain methods, fracture mechanics based methods, cohesive zone models and the extended finite element method (Pascoe et. al 2013). de Borst and Remmers 2006 studied the delamination of Glare in a DCB using a cohesive interphase to model the adhesive layer and exploiting the partition of unity property (XFEM). Motamedi et. al. 2013 studied the delamination process of Polyphenylene sulfide/Glass fiber reinforced polymer DCB and performed a 3D analysis in Abaqus/MATLAB using XFEM capabilities. Stochasticity was incorporated into the model via material properties.

Other researchers have also implemented the extended finite element method for the analysis of carbon fiber composite laminates (Cahill et. al 2014), metal fiber laminates

(Sosa and Karapurath 2012) and particle-reinforced composites (Ye et. al 2012). Cahill et. al 2014 utilized an orthotropic variation of Equation 2.16 proposed by Asadpoure and Mohammadi in 2007 alongside with the Heaviside and bi-material interface enrichment functions and demonstrated that the maximum hoop stress criterion, for the determination of delamination propagation direction, is unsuitable for fracture of orthotropic materials. Sosa and Karapurath 2012 modelled a DCB using a bimaterial definition for Equation 2.16 proposed by Sukumar et. al in 2004 however, only incremental loading was tested. Ye et. al 2012 utilized the XFEM capabilities of ABAQUS with user-defined subroutines to model a plate with reinforcing inclusions and used the Paris equation in Equation 3.5 to simulate fatigue loading. These research show the versatility of the XFEM when modeling fracture.

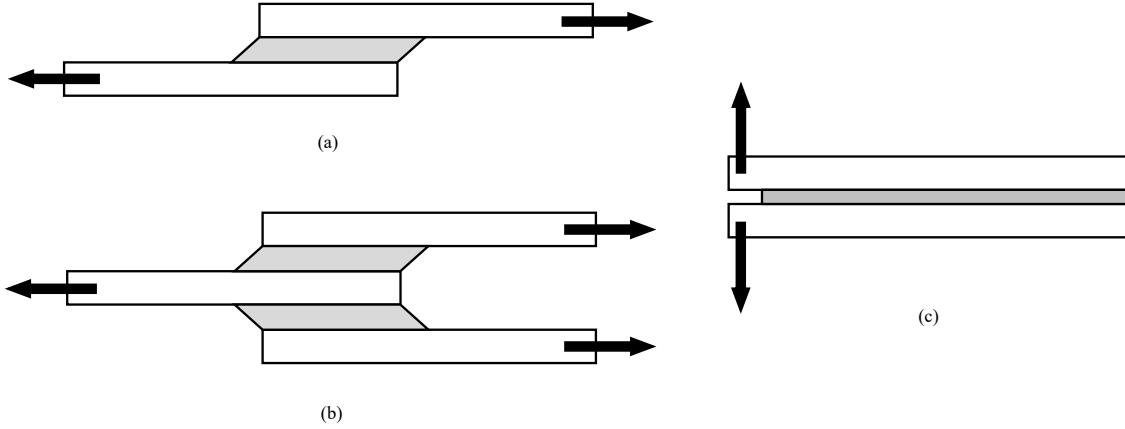


Figure 2.1: Single lap joint (a), double lap joint (b) and double cantilever beam (c); arrows indicate loading conditions

2.2 Finite element formulation

The structural deformation of the adhesive bonded joint can be described by the Principle of Virtual Work where the external work due to external forces must be balanced by the structure due to internal forces (stresses); refer to Kuna 2013. Let the external virtual work be defined by δW_{ext} and the internal work by δW_{int} . The external work δW_{ext} is comprised by the contribution of point loads \mathbf{f}^p applied at the mesh nodes, body force per unit volume \mathbf{f}^b (e.g. material weight) acting on the element differential volume dV and traction forces per unit area \mathbf{f}^t acting on the differential surface dS . Moreover, let

us denote the internal work δW_{int} as equal to the strain energy δU which arises due to the material response to the mechanical loading. Hence, both the expressions for δW_{ext} and δW_{int} can be written as:

$$\begin{aligned}\delta W_{ext} &= \oint_S \delta \tilde{\mathbf{d}} \cdot \mathbf{f}^t dS + \int_V \delta \tilde{\mathbf{d}} \cdot \mathbf{f}^b dV + \delta \tilde{\mathbf{d}} \cdot \mathbf{f}^p \\ \delta W_{int} &= \delta U = \int_V \boldsymbol{\sigma} : \delta \boldsymbol{\varepsilon} dV = \int_V \sigma_{ij} \delta \varepsilon_{ji} dV\end{aligned}\tag{2.1}$$

where $\boldsymbol{\sigma}$ represents the structure stresses and $\boldsymbol{\varepsilon}$ the strains. The displacement vector $\tilde{\mathbf{d}}$ represents the nodal displacements which are interpolated from the calculated displacements at the integration (Gauss) points by an expression of the form $\tilde{\mathbf{d}}(\mathbf{X}) = \mathbf{N}\mathbf{d}$. In the standard FEM formulation, \mathbf{N} represents the shape function matrix and \mathbf{d} the nodal displacements.

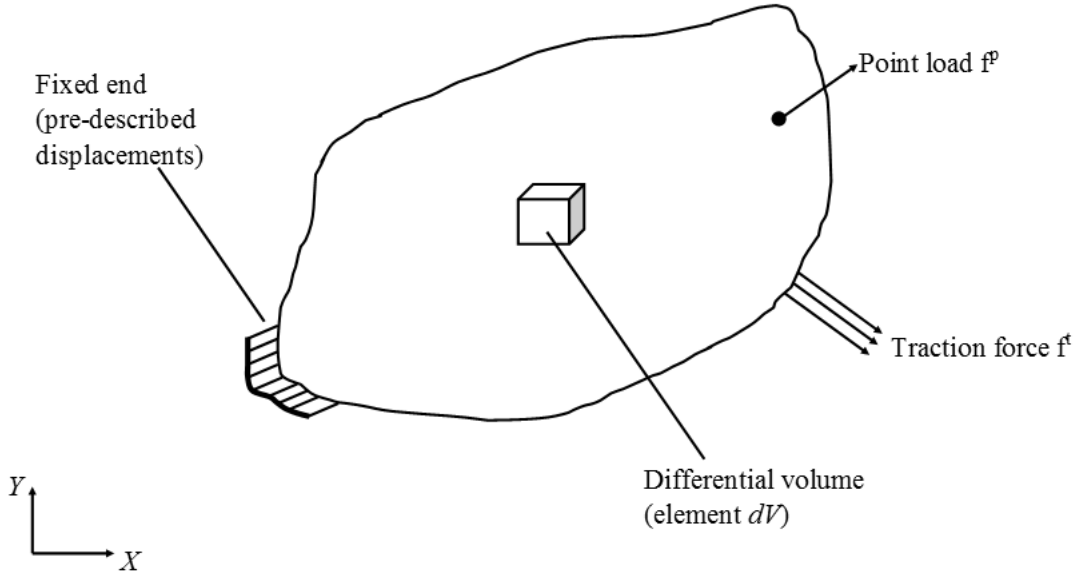


Figure 2.2: Body with prescribed displacement and loads

Application of the Principle of Virtual Work, which result in $\delta W_{int} = \delta W_{ext}$, neglecting traction and body forces for simplicity and reducing the system to a two dimensional space results in:

$$h \int_{\Omega} \boldsymbol{\sigma} : \delta \boldsymbol{\varepsilon} d\Omega = \delta \mathbf{d} \cdot \mathbf{f}^p\tag{2.2}$$

were $d\Omega$ is the element differential surface and h the through-the-thickness distance.

Assuming small displacements and rotations of the structure, a linear approximation of strains can be used. Furthermore, if a linear relationship between stresses and strains is assumed, Hooke's law can be implemented ($\boldsymbol{\sigma} = \mathbf{D} : \boldsymbol{\varepsilon}$). Application of the compatibility condition for small deformations ($\varepsilon = \partial d_i / \partial x_j$) results in the following system of equations:

$$\left[\left(h \int_{\Omega} \mathbf{B}(X, Y)^T \mathbf{D} \mathbf{B}(X, Y) d\Omega \right) \mathbf{d} - \mathbf{f}^p \right] \delta \mathbf{d} = 0 \quad (2.3)$$

Because $\delta \mathbf{d}$ is an arbitrary virtual nodal displacement, for the equation to hold true the term in brackets must vanish. Therefore, the term within brackets represent the differential boundary value problem to solve. Furthermore, the term in parenthesis is the stiffness matrix of the structure. Given by:

$$\mathbf{K} = h \cdot \bigcup_{\Omega} \int_{\Omega} \mathbf{B}(X, Y)^T \mathbf{D} \mathbf{B}(X, Y) d\Omega = h \cdot \bigcup_{\Omega} \mathbf{B}(\xi, \eta)^T \mathbf{D} \mathbf{B}(\xi, \eta) |\mathbf{J}| w \quad (2.4)$$

were w is a weight value for numerical integration, $|\mathbf{J}|$ is the Jacobian matrix determinant and \mathbf{B} is the matrix of shape function derivatives or the so called strain/displacement matrix:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial X}{\partial \xi} & \frac{\partial Y}{\partial \xi} \\ \frac{\partial X}{\partial \eta} & \frac{\partial Y}{\partial \eta} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \frac{\partial N_i}{\partial X} & 0 \\ 0 & \frac{\partial N_i}{\partial Y} \\ \frac{\partial N_i}{\partial Y} & \frac{\partial N_i}{\partial X} \end{bmatrix} \quad (2.5)$$

and transformation of $\mathbf{B}(X, Y)$, in terms of global coordinate system (X, Y) , to $\mathbf{B}(\xi, \eta)$, in terms of element reference coordinate system (ξ, η) (refer to Figure 2.3):

$$\begin{Bmatrix} \frac{\partial N_i}{\partial X} \\ \frac{\partial N_i}{\partial Y} \end{Bmatrix} = \mathbf{J}^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{Bmatrix} \quad (2.6)$$

where (ξ, η) are customarily used to accurately simulate irregularities in the elements (e.g. curved elements).

Hence, the system of equation to solve for the structure displacement is:

$$\mathbf{K}(\xi, \eta) \mathbf{d} = \mathbf{F} \quad (2.7)$$

where the imposed boundary conditions are point loads in the surface of the structure,

crack surfaces are traction free and prescribed displacements in the boundary. In equation form these can be written as:

$$\begin{aligned}\boldsymbol{\sigma} \cdot \mathbf{n} &= 0 && \text{crack surface} \\ \mathbf{d} &= 0 && \text{prescribed surface}\end{aligned}\tag{2.8}$$

2.2.1 Incompatible elements

The classical bilinear finite elements (commonly referred to as Q4 elements), provides linear variation of the strain within its volume. The element has been widely used for many structural problems and within the Extended Finite Element Method. It employs the following set of bilinear shape functions to describe the displacement within its area:

$$N(\eta, \xi)_i = \frac{1}{4}(1 + \xi_i \xi)(1 + \eta_i \eta)\tag{2.9}$$

where (ξ_i, η_i) represent the natural coordinate values at each node (see Figure 2.3 bellow).

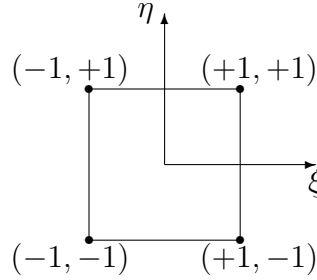


Figure 2.3: Node coordinates for the quadrilateral element in the computational space

In the bilinear quadrilateral element, quantities of interest can be linearly interpolated from known values at the node via the shape functions as follows:

$$z(\xi, \eta) = \sum_{i=1}^4 N_i(\xi, \eta) z_i\tag{2.10}$$

However, the element is known to be too stiff in bending and even suffer from shear locking for elements with big aspect ratios (Logan 2011); refer to Figure 2.4. This becomes an inconvenience when studying long narrow structures, as in adhesive bonded joints as these structures are commonly subjected to bending. Of course, to circumvent this limitation, a large number of elements can be used along the bonded joint (Logan

2011). However, this increases the computational time, and more so for the XFEM as each node has an increased number of degrees of freedom. As such, in this work an improved bilinear quadratic element known as Q6 is employed. This element was first introduced by Wilson et. al. in 1973. The procedure is to add quadratic terms to the interpolation in Equation 2.10. The interpolation is then expanded to:

$$z(\xi, \eta) = \sum_{i=1}^4 N_i(\xi, \eta) z_i + (1 - \xi^2)g_1 + (1 - \eta^2)g_2 \quad (2.11)$$

The additional degrees g_1 and g_2 are internal and thus do not contribute to the global stiffness matrix.

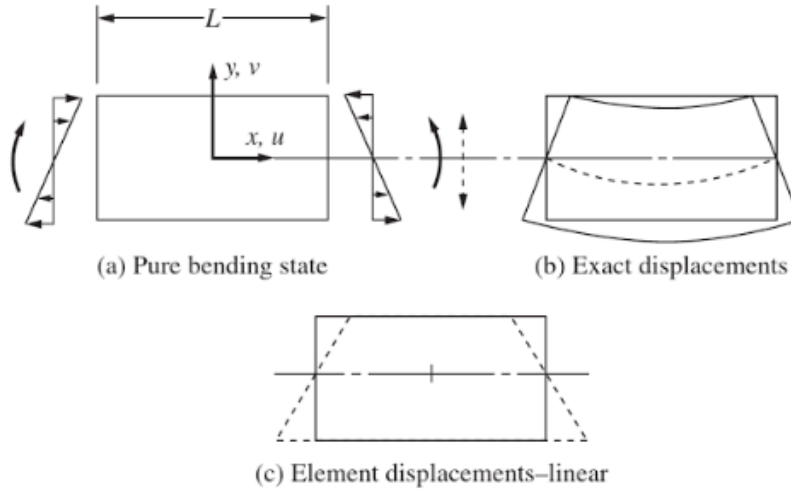


Figure 2.4: Q4 element in bending, from Logan 2011

2.3 The Extended Finite Element formulation

The Extended Finite Element Method is based on the Partition from Unity method (Belytschko and Black 1999). Two distinct approaches can be followed, intrinsic or extrinsic. The former approach deals with the addition of information from the analytical solution to the basis function thus increasing the order of completeness (Mohammadi 2008 and 2012). The latter approach deals with the addition of additional unknowns (degrees of freedom) to add the information from the analytical solution. The method has been used for description of cracks in isotropic media (Belytschko and Black 1999), for cracks in a bimaterial interphase (Sukumar et. al 2004), for orthotropic media (Asadpoure and

Mohammadi 2007)

2.3.1 Discontinuity field description

The strong discontinuity enrichment function is used to model the cracked elements (i.e the strong discontinuity field). Many definitions of the Heaviside function have been adopted through the years (Mohammadi 2008). Definitions such as step functions and smoothed step functions have also been proposed. The selected version of Heaviside function in this work is the signed distance function which has the form of:

$$\mathcal{H}(\psi) = \text{sign}(\psi) = \begin{cases} +1 \\ -1 \end{cases} \quad (2.12)$$

where ψ is the normal level set of the delamination which in turn is defined as the normal distance from the crack face (refer to Section 2.3.2).

It is also important to note that the derivative of the Heaviside function is the Dirac delta function which becomes zero except at the strong discontinuity:

$$\frac{\partial \mathcal{H}(\psi)}{\partial X_i} = \begin{cases} 1 & \text{when } \phi = 0 \\ 0 & \text{when } \phi \neq 0 \end{cases} \quad (2.13)$$

Now the strong discontinuity enrichment function and its derivative can be defined as:

$$H = N_k (\mathcal{H} - \mathcal{H}_k) \quad (2.14)$$

$$\frac{\partial H}{\partial X_i} = \frac{\partial N_k}{\partial X_i} (\mathcal{H} - \mathcal{H}_k) \quad (2.15)$$

Note that k is a nodal index and X_i is the spatial coordinate of the real space.

To describe the asymptotic field near the delamination front (crack tip), several enrichment functions exist in the literature. These enrichment functions are derived from asymptotic crack tip displacement fields and are thus dependent on the material definition inside the domain where these are applied (i.e. isotropic or orthotropic and linear or non-linear elastic for the scope of this study). Similar to the above enrichment functions, a shifting procedure is also needed to preserve interpolation.

The near-tip field equations for isotropic media were originally proposed in the framework of XFEM by Belytschko and Black 1999. However, these were first introduced by Fleming in 1997 (see also Fleming et. al 1997).

$$f^l(r, \theta) = \left(\sqrt{r} \sin \frac{\theta}{2}, \sqrt{r} \cos \frac{\theta}{2}, \sqrt{r} \sin \frac{\theta}{2} \sin \theta, \sqrt{r} \cos \frac{\theta}{2} \sin \theta, \right) \quad (2.16)$$

These functions span the asymptotic field close to the crack tip 'near-tip field' which are based on a first order approximation of the stress field solution by William in 1957. The complete solution of the stress field is given by a infinite series expansion of eigenvalues and eigenfunctions. Hence, this expansion results in a non-singular higher order stress field. The first order terms in the William expansion represent the stress singularity (\sqrt{r}) and thus contain the stress intensity factors. However, the second order terms describe a uniform non-singular stress parallel to the crack tip. This stress is commonly known in the fracture mechanics literature as the 'T-stress'. As a consequence, two new crack tip enrichment functions are used to increase the accuracy of the model:

$$f^{5,6}(r, \theta) = (r \cos \theta, r \sin \theta) \quad (2.17)$$

Higher order terms have been reportedly used in the literature by Xiao and Karihaloo (2006) however, no clear definition was provided for the enrichment functions. Nevertheless, the inclusion of the second terms in the William's expansion does improve the accuracy of the stress field near the crack tip.

The functions in Equations 2.16 and 2.17 describe the asymptotic strain field in the vicinity of the crack tip where θ and r are the polar coordinates to the material point (integration point) in the structure relative to the crack plane. The coordinate convention is shown in Figure 2.5.

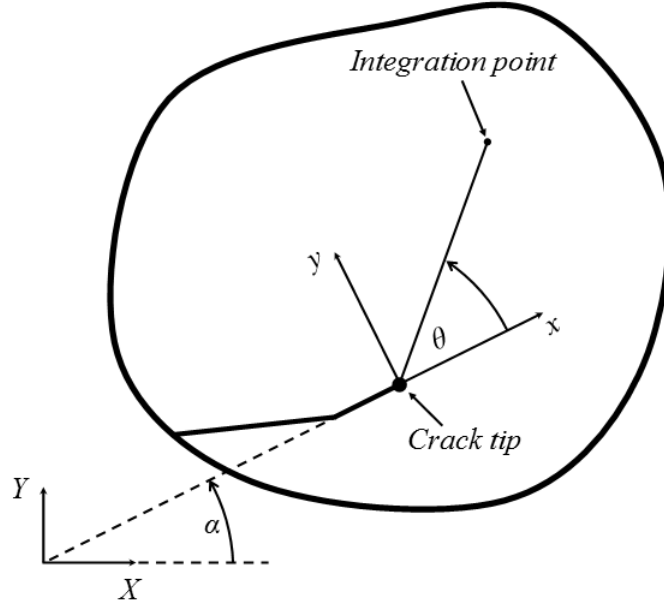


Figure 2.5: Crack coordinate convention

These enrichment functions are known to produce inaccuracies in the blending elements thus (Fries 2008) hence, in this work the corrected model developed by Fries in 2008 is employed. In their general form, the near-tip enrichment functions can be expressed as:

$$F^l = N_k (f^l - f_k^l) R \quad (2.18)$$

where $R(x)$ is a ramp function. The derivative of f^l near tip functions can be computed from the chain rule as:

$$\frac{\partial F^l}{\partial X_i} = \frac{\partial N_k}{\partial X_i} (f^l - f_k^l) R + N_k \left(\frac{\partial f^l}{\partial X_i} - \frac{\partial f_k^l}{\partial X_i} \right) R + N_k (f^l - f_k^l) \frac{\partial R}{\partial X_i} \quad (2.19)$$

Therefore, the complete displacement approximation of the solution for a quadrilateral element can be expressed as:

$$d(x_j) = \sum_{i=1}^4 N_i u_i + \sum_{i=1}^4 N_i (\mathcal{H} - \mathcal{H}_i) a_i + \sum_{l=1}^k \sum_{i=1}^4 R(x_j) N_i (F^l - F_i^l) b_i^l \quad (2.20)$$

where $R(x_j)$ is a ramp function and N_i the interpolation functions for the quadrilateral

element.

2.3.2 Level set method

The level set method (LSM) was first proposed by Osher and Sethian in 1988 for tracking of moving surfaces. Functions of higher order than the interface being modeled are used to track the evolution of the crack. This method of representing interfaces has been proven to be effective in modeling of inclusions in the mesh (Sukumar et. al 2001) (closed interfaces). Modeling of open interfaces such as cracks requires an extension of the LSM. Stolarska et. al (2001) extended the LSM for the purpose of describing open surfaces by tracking the crack as a combination of two level set functions. A tangential level set function ψ describes the crack interface whereas a orthogonal level set function ϕ describes the location of the crack tip. Both functions describe the location at its zero level set as shown in Figure 2.6.

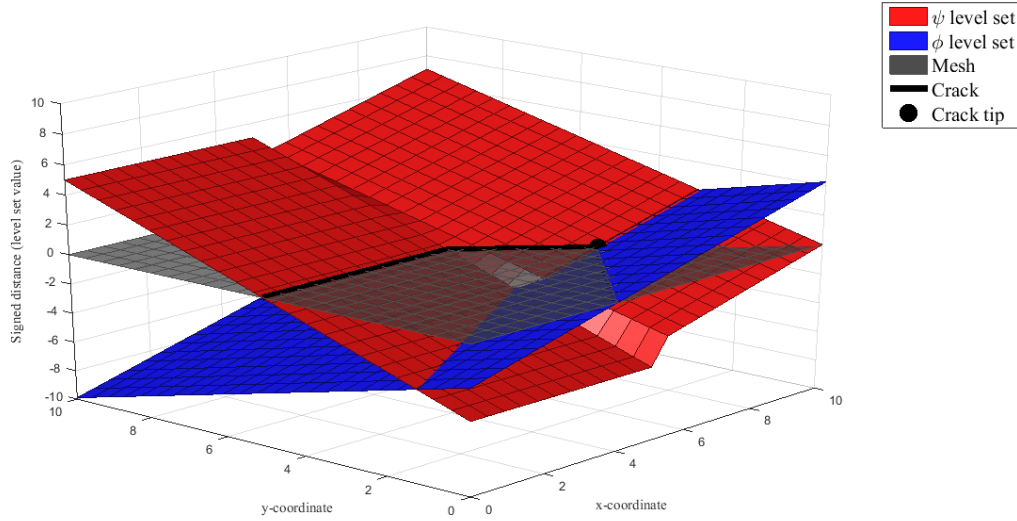


Figure 2.6: Level sets

The normal distance level set is constructed from values from a signed distance function $\chi(x)$ (Sukumar and Prévost 2003). This function has its value defined at point x for a distance measured from boundary x_Γ (crack). Both the tangential level set ψ and

normal level set ϕ can then be defined via the signed distance function as:

$$\begin{aligned}\psi &= \min ||\mathbf{X} - \mathbf{X}_\Gamma|| \cdot \text{sign}(\mathbf{n}_n \cdot (\mathbf{X} - \mathbf{X}_\Gamma)) \\ \phi &= \min ||\mathbf{X} - \mathbf{X}_\Gamma|| \cdot \text{sign}(\mathbf{n}_t \cdot (\mathbf{X} - \mathbf{X}_\Gamma))\end{aligned}\tag{2.21}$$

where \mathbf{n}_n and \mathbf{n}_t are the normal and tangential unit vectors, for the crack segment.

Stolarska et. al (2001) also provided a method for updating the level sets for propagating cracks. Furthermore, they also provided a method for node classification for enrichment and polar coordinate computation based on level set values. Nevertheless, both the node identification and polar coordinate schemes have been proven to be inaccurate (Ahmed 2009). However, inaccuracies in element enrichment can be fixed by evaluating each enriched element for containment of the crack tip geometrically in the subset identified as near-tip elements. The method is explained below.

For the vector of the new crack segment \mathbf{F} and the vector of the previous crack segment \mathbf{V} , the angle α can be computed. The normal level sets for the nodes ahead of the crack front ($\phi > 0$) are updated by:

$$\psi^{n+1} = -\text{sign}(\alpha) (\mathbf{x} - \mathbf{x}_{\text{crack tip}}) \times \frac{\mathbf{F}}{||\mathbf{F}||}\tag{2.22}$$

Note that ψ is only updated if $\alpha \neq 0$. Now ϕ is updated via:

$$\phi^{n+1} = (x - x_{\text{crack tip}}) \frac{F_x}{||\mathbf{F}||} + (y - y_{\text{crack tip}}) \frac{F_y}{||\mathbf{F}||}\tag{2.23}$$

The procedure for node selection for enrichment follows the same procedure as established by Stolarska (2001). If the nodes in an element with $\phi < 0$ and $\phi_{\min} \cdot \phi_{\max} \leq 0$ the element is classified as Heaviside elements (completely cut by the crack). If $\psi_{\min} \cdot \psi_{\max} \leq 0$ and $\phi_{\min} \cdot \phi_{\max} \leq 0$, the crack tip might be within the element. As shown by Ahmed in 2009, this procedure has its flaws, specifically for the condition of crack tip enrichment. This can be remedied by doing a simple geometric query on the crack tip enriched elements (following the above criteria). If the crack tip is found within the element area, the enrichment is preserved, if not, the enrichment is dropped.

The element enrichment scheme is adopted here as it is accurate for cohesive failure of the adhesive layer. However, calculation of polar coordinates for use in near-tip enrichment functions are computed geometrically to avoid inaccuracies. Impact to the computational time is minimal as polar coordinates need only to be calculated at near-tip

enriched nodes.

With the aid of the level sets, the enriched space can be defined around the crack and the discontinuity functions for crack description embedded into the analysis. An example of an enriched space is represented in Figure 2.7

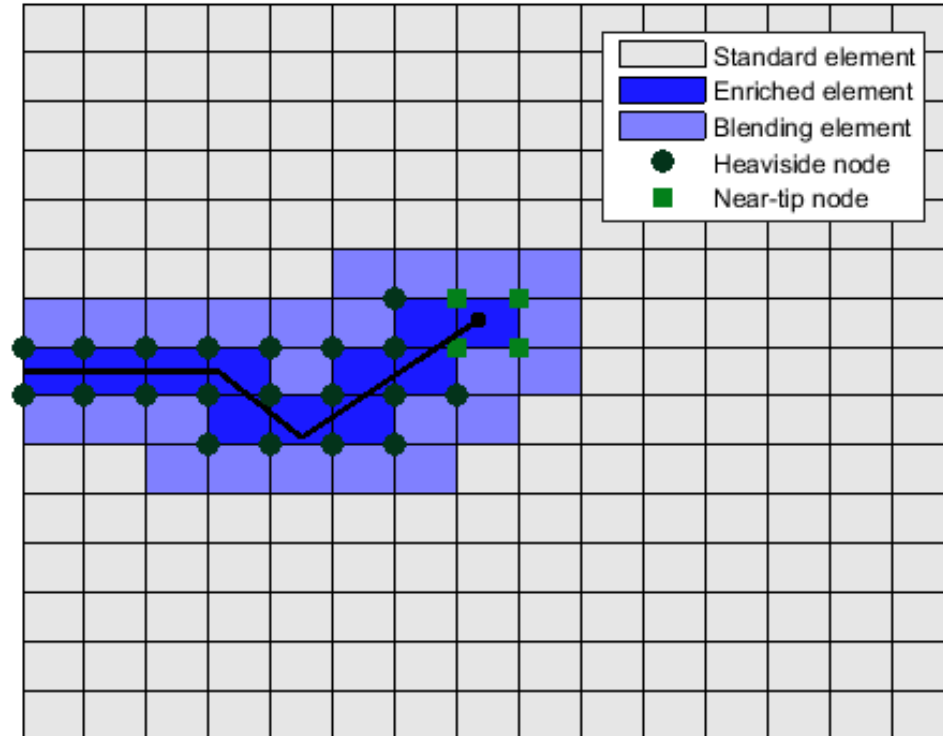


Figure 2.7: Enrichment space example

2.3.3 Blending elements

Fries 2008 addresses the issue of blending elements by using a Ramp Function built using standard shape functions. In standard un-enriched elements the partition of unity holds (see Eq 2.24). For blending elements, which have some of their nodes enriched, the opposite occurs; the sum of the shape function does not equal 1.

$$\begin{aligned} \sum_{i \in I} N_i(\mathbf{x}) &= 1 \\ R(\mathbf{x}) &= \sum_{i \in J} N_i(\mathbf{x}) \end{aligned} \tag{2.24}$$

Strong discontinuities in this work do not pose a problem in blending elements. As explained in Fries work, the functions used for strong discontinuity enrichment are of constant value as long as the shape functions used for the enrichment are of equal or less order as those of the standard FEM part of the approximation. This is the case for Heaviside and Sign enrichment functions used in this thesis.

2.3.4 Degrees of freedom

In the framework of the standard FEM, the nodal degrees of freedom correspond to the x and y displacements of the structure (for two dimensional general analysis). In the XFEM framework, the displacement field is enriched with functions that describe the phenomenological fields of interest. Modeling of the discontinuous field of a crack, the Heaviside function H is introduced. To describe the asymptotic field produced by the delamination front (crack tip) the Near-tip functions F^l are introduced. The displacement field is then the linear combination of the physical x and y displacements with the enrichment functions. In equation form, this is described as:

$$d = \{u_{k_x} \quad u_{k_y} \quad a_{k_x} \quad a_{k_y} \quad b_{k_x}^l \quad b_{k_y}^l\}^\top \tag{2.25}$$

where u represents the displacements DoFs, a the strong discontinuity DoFs and b^l the DoFs for the near-tip discontinuity for l enrichment function. Note that the incompatible degrees of freedom, as introduced in Section 2.2.1, are not included as these are internal DoFs that are condensed out before assembly of the global stiffness matrix \mathbf{K}

Now that all the enrichment functions have been introduced, the continuum and energy concepts presented in Section 2.2 are presented in the framework of the extended finite element method. The strain displacement matrix for the element is now a combination of the contribution of each enrichment function in x and y coordinates. In equation form this is expressed as:

$$\mathbf{B} = \begin{bmatrix} B_i^u & B_i^a & B_i^b \end{bmatrix} \tag{2.26}$$

where B^u are the strain/displacement matrix for the quadrilateral element as provided in Section 2.2

$$\mathbf{B}^a = \begin{bmatrix} \frac{\partial}{\partial \xi}(N_i(H - H_i)) & 0 \\ 0 & \frac{\partial}{\partial \eta}(N_i(H - H_i)) \\ \frac{\partial}{\partial \xi}(N_i(H - H_i)) & \frac{\partial}{\partial \eta}(N_i(H - H_i)) \end{bmatrix} \quad (2.27)$$

$$\mathbf{B}^b = \begin{bmatrix} \frac{\partial}{\partial \xi}(N_i(F - F_i)R(X)) & 0 \\ 0 & \frac{\partial}{\partial \eta}(N_i(F - F_i)R(X)) \\ \frac{\partial}{\partial \xi}(N_i(F - F_i)R(X)) & \frac{\partial}{\partial \eta}(N_i(F - F_i)R(X)) \end{bmatrix} \quad (2.28)$$

2.3.5 Element integration

In this work, the integration scheme proposed by Dolbow 1999 is implemented. Here, the enriched elements are subdivided into integration sub-cells. This subdivision does not introduce additional degrees of freedom as they are only used for integration. Several advantages are identified on this scheme when compared to the sub-triangulation (see example in Figure 2.8), the later technique well explained in the work of Nguyen 2005.

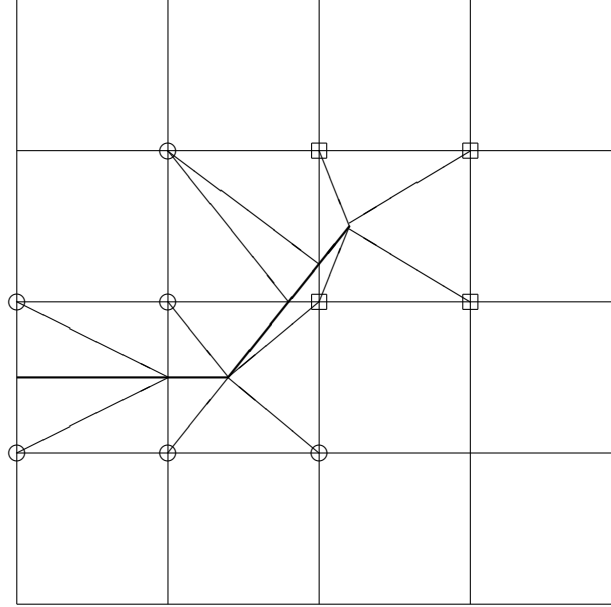


Figure 2.8: Example of element sub-triangulation

Sub-tessellation using triangles requires the isoparametric mapping of the Gauss points from each triangle into the element space Ω . As no close form equation exists for this mapping, an iterative solution (e.g. Newton-Raphson) has to be performed

which imposes additional burdens to the computer code. Another disadvantage of sub-tessellation using triangulation arises when handling delamination propagation. As the delamination front moves during the simulation, the integration points might have to be moved (re-subtriangulation) and the computed integration point quantities have to be recalculated.

Sub-tessellation using quadrilaterals does come with its disadvantages. As the integration is now performed in quadrants that do not conform to the discontinuity, the equivalence between the weak and strong form of the finite element formulation is lost (Sukumar and Prévost 2003). However, if sufficient quadrilaterals are used, the errors introduced by this method of integration is reduced.

Take into consideration the stiffness equation for a linear elastic formulation where \mathbf{B} is the strain/displacement matrix and \mathbf{D} the material matrix:

$$\mathbf{K} = \int_{\Omega} \mathbf{B}^T \mathbf{D} \mathbf{B} d\Omega \quad (2.29)$$

The Gauss/Legendre approximation in two dimensions for the above formula takes the following form:

$$\mathbf{K} = \sum_{j=1}^s \left(\sum_{i=1}^g \mathbf{B}_i^T \mathbf{D} \mathbf{B}_i w_i \right)_j \quad (2.30)$$

where s is the total number of subcells within the element and g the number of Gauss points.

The procedure for this integration scheme is as follows: The element is subdivided into quadrilaterals and 4 point integration is adopted on each sub-quadrilateral (see Figure 2.9). The integration points on each sub-quadrilateral mapped from the reference $([-1, 1], [-1, 1])$ domain are mapped to the element Ω space via Equation 2.10. The integration points in Ω are then mapped back to the reference space of the element for integration. It is customary to use an iterative solution (e.g. Newton-Raphson) to perform the inverse mapping of the integration points. However, to decrease the computational burden, the inverse mapping proposed by Hua in 1990 is employed. Note that also the blending elements around the crack tip are partitioned into quadrilaterals (see Figure 2.7). After the integration points are mapped, the weights are also mapped using the equation below as described by Nguyen in 2008:

$$w_{\text{in element}} = w_{\text{in sub-quad}} \cdot \|\mathbf{J}\|_{\text{in sub-quad}} \quad (2.31)$$

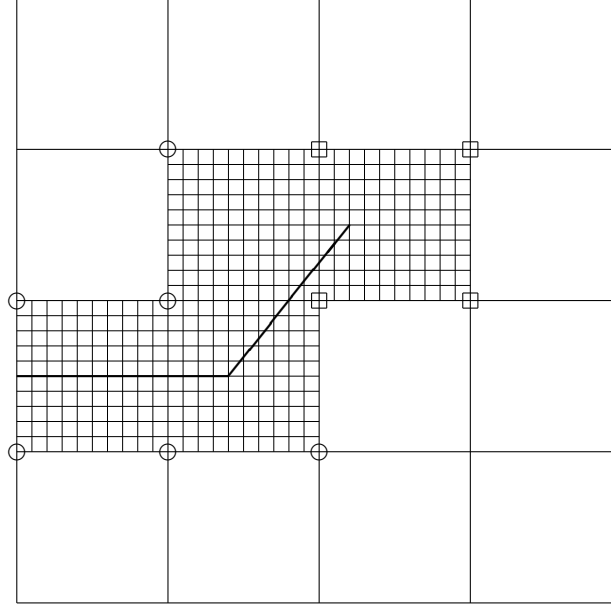


Figure 2.9: Example of element partitioning using sub-quadrilaterals

2.4 Fracture Mechanics

The first law of thermodynamics, which is an expression of the conservation of energy principle, states that energy in a system must be conserved. Hence, when a system goes from a state of imbalance to a state of equilibrium, there will be a net decrease in energy. Griffith (1921) applied this principle to the formation of a crack. Application of this principle to through-thickness crack in an infinitely wide plate in tension yields the following expression:

$$G = \frac{\pi \sigma^2 a}{E} \quad (2.32)$$

where G is the energy for crack formation per unit area known as the energy release rate as introduced by Irwin (1957). The energy release rate is a measure of the available energy for crack extension. Therefore, a critical value of this parameter provides a material fracture property, known as fracture toughness G_c . Note that the expression in 2.32 shows that the energy available for crack formation or extension depends on the material elastic modulus E , the stress state σ and crack length a .

The expression in 2.32 is limited to both loading configuration, material model and crack-tip plasticity (crack plasticity is contained in a small region). The path-independent contour integral, independently introduced by Cherepanov in 1967 and Rice in 1968 can

then be used to define a non-linear energy release rate. As introduced by Rice, the path independent integral takes the form of:

$$J = \int_{\Gamma} \left(w dy - T \cdot \frac{\partial u}{\partial x} ds \right) \quad (2.33)$$

where ds is the differential path in Γ , T corresponds to the traction acting normal to the integration path and w is the strain energy:

$$w = \int_0^{\varepsilon_{ij}} \sigma_{ij} d\varepsilon_{ij} \quad (2.34)$$

In order to facilitate the computation of energy release rate J in the finite element framework, an equivalent domain integral is implemented (Shih et al. 1986). The attractiveness of the equivalent domain integral is its versatility and its simplicity when implemented in a finite element code. The integral in equation 2.33 then becomes the following equivalent domain integral:

$$J = \sum_{\Omega} \sum_n \left[\left(\sigma_{ij} \frac{\partial u_j}{\partial x_1} - w \delta_{1i} \right) \frac{\partial q}{\partial x_i} \right]_n |\mathbf{J}|_n w_n \quad (2.35)$$

Note that the equivalent domain integral in 2.35 does not take into account traction forces at the crack surface, body forces, thermal or inelastic strains or nonlinearities which are in accordance with this work assumptions (refer to Kuna 2013 for a more general expression). In equation 2.35, q is an arbitrary continuous differentiable weighing function that becomes zero outside of the integration area. As it is arbitrary, the following definition is applicable:

$$q = \begin{cases} 0 & \text{on } \Gamma \\ 1 & \text{on } \Gamma_{\epsilon} \end{cases} \quad (2.36)$$

The weighing function q above is computed at the nodes. Interpolation into the integration points is then performed similar to other quantities by using the shape functions of the isoparametric element:

$$\begin{aligned} q(x) &= N_i q_i \\ \frac{\partial q(x)}{\partial x_j} &= \frac{\partial N_i}{\partial \xi_j} \frac{\partial \xi_j}{\partial x_k} q_i \end{aligned} \quad (2.37)$$

The contribution to the elasto-plastic J-integral is only on elements on which there is a gradient of the weighing function q , ($0 < q < 1$). Figure 2.10 depicts the area of

integration for the double cantilever beam used for analysis. The yellow colored area represents the area where the weight function q has a value of 1 and the blue area where it has a value of 0, as defined in Equation 2.36. The equivalent domain integral is calculated where q is not equal to 0 as defined in Equation 2.35 but only the elements in the periphery of the yellow area (where $0 < q < 1$) contribute to the calculation of the energy release rate J .



Figure 2.10: J integral area of integration

2.4.1 Crack tip near fields

Several closed-form expressions for stresses in a body have been published for isotropic linear elastic material behavior (Anderson 2005 and Perez 2004). For the coordinate system defined in Figure 2.5, the stress field in a linear elastic fractured body can be defined by:

$$\sigma_{ij}^k = \frac{K_k}{\sqrt{2\pi r}} f_{ij}^k(\theta) + \sum_{m=0}^{\infty} A_m r^{\frac{m}{2}} g_{ij}^m(\theta) \quad (2.38)$$

where the term $f_{ij}^k(\theta)$ represent a dimensionless term dependent of the crack tip angle θ with respect to the query point. Also, K_k is the stress intensity factor (SIF) where the subscript k denotes the mode of loading i.e. K_I, K_{II} or K_{III} ; see Figure 2.11.

The first term in the expansion associates the stress field around the crack tip with the stress intensity factor, a measurement of the stress state. Normally, based on the small scale yielding assumption (small plastic zone when compared with crack length), only the first term of the expansion is used (William 1957).

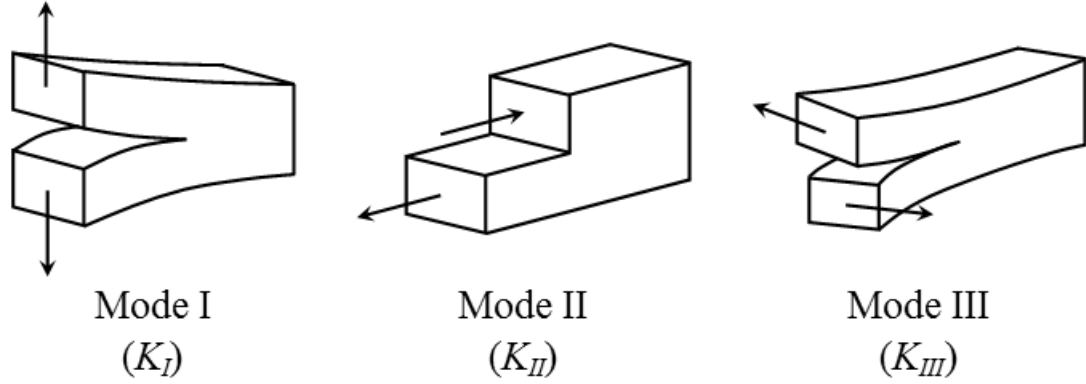


Figure 2.11: Fracture modes

In Equation 2.38, $f_{ij}^k(\theta)$ is a trigonometric function that has to be derived analytically (Perez 2004). Derivation of the stress field ahead of the crack tip can be found through the literature. However, a more general expression of the stresses and displacements (needed for the definition of the near-tip enrichment functions in Equations 2.16 and 2.17) can be found in the work of Xiao et. al. in 2004.

Hence, for isotropic media, the displacement field ahead of the crack tip (using only the first two terms in the expansion) is described by:

$$\begin{aligned}
 u_x &= \frac{\sqrt{r}}{2\mu} \left\{ a_1 \cos\left(\frac{\theta}{2}\right) \left[\kappa + 1 - 2 \cos\left(\frac{\theta}{2}\right)^2 \right] - b_1 \sin\left(\frac{\theta}{2}\right) \left[\kappa + 1 + 2 \cos\left(\frac{\theta}{2}\right)^2 \right] \right\} + \\
 &\quad \frac{r}{2\mu} \{ a_2 \cos(\theta) - b_2 \sin(\theta) \} (\kappa + 1) \\
 u_y &= \frac{\sqrt{r}}{2\mu} \left\{ a_1 \sin\left(\frac{\theta}{2}\right) \left[\kappa + 1 - 2 \cos\left(\frac{\theta}{2}\right)^2 \right] - b_1 \cos\left(\frac{\theta}{2}\right) \left[\kappa - 3 + 2 \cos\left(\frac{\theta}{2}\right)^2 \right] \right\} + \\
 &\quad \frac{r}{2\mu} \{ a_2 \sin(\theta)(3 - \kappa) - b_2 \cos(\theta)(1 + \kappa) \}
 \end{aligned} \tag{2.39}$$

where μ is the shear modulus and κ the Kolosov's constant given by:

$$\kappa = \begin{cases} 3 - 4\nu & \text{plane strain} \\ \frac{3-\nu}{1+\nu} & \text{plane stress} \end{cases} \quad (2.40)$$

Note that a_1 and b_1 are related to the stress intensity factors by:

$$a_1 = \frac{K_I}{\sqrt{2\pi}} \quad b_1 = -\frac{K_{II}}{\sqrt{2\pi}} \quad (2.41)$$

and a_2 related to the T-stress which accounts for a constant stress parallel to the crack.

The near field displacement equations presented above are the basis for the near-tip enrichment functions in Equations 2.16 and 2.17 used for modeling of the near field stresses around the delamination front in the adhesive.

2.5 Case study: Double Cantilever Beam

2.5.1 Analytical solution

A typical test configuration for fracture toughness (delamination) in composites and strength of the adhesive layer is the double cantilever beam (DCB) (Banea and da Silva 2009, Biel and Stigh (2007)). An example of a double cantilever beam is shown in Figure 2.12. The displacement δ (which coincides with the displacement of the loading point) can be computed using beam theory. According to Timoshenko beam theory, the displacement can be calculated analytically by:

$$\delta = \frac{2Pa^3}{3EI} + \frac{Ph^2a}{4\mu I} \quad (2.42)$$

In the above equation, P is the applied load, I the second moment of inertia, and E and μ the Young's and shear modulus respectively. Note that if Euler-Bernoulli beam theory is used, the second term in Equation 2.42 is neglected. To obtain the deflection of one of the cantilever end, δ is divided by 2.

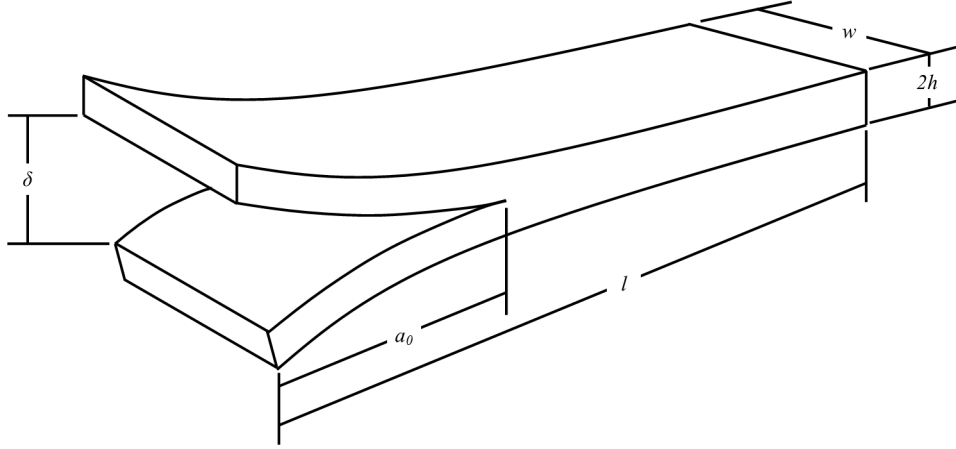


Figure 2.12: Double cantilever beam in displacement control

Analytical solutions for the energy release rate exists through the literature. For the classical beam theory, the following expression can be obtained (Anderson 2005):

$$G = \frac{12P^2a^2}{w^2h^3E} \quad (2.43)$$

or from the modified beam theory method (Prasad et. al 2011):

$$G_I = \frac{3P\delta}{2ta} \quad (2.44)$$

However, these equations will overestimate the energy release rate as they do not account for the rotations at the loaded ends of the double cantilever beam. This can be corrected by introduction of an effective crack length (Nairn 2000).

2.5.2 Convergence study

A convergence study was performed to determine the minimum required number of elements needed for the solution of adhesive fatigue failure of the double cantilever beam. This configuration is selected as close form analytical solutions exist and because it is a common test setup to evaluate adhesive strength. The test case is selected so that it compares to the benchmark double cantilever beam setup as in Krueger's report (2010). The material properties selected for benchmark are: $E = 70,000 \text{ N/mm}^2$, $\nu = 0.33$ (material

properties for isotropic media for convergence study only) and the analysis carried out in plane strain conditions (here, the epoxy presence is ignored in the analytical solution but modeled in the XFEM framework). This assumption is valid due to the thin section of the adhesive layer.

As shown in Figure 2.13 the finite element solution starts to oscillates between 2% of the analytical solution at about 750 elements, with refinement at the loading end and near the delamination as depicted in Figure 2.14. As shown, the incompatible element can be used to predict the energy release rate. However, the incompatible element tends to be significantly less stiffer thus, a higher number of elements must be used in the transverse direction of the beam. Figure 2.14 shows an example of the mesh used where the red colored line represent the crack or initial delamination.

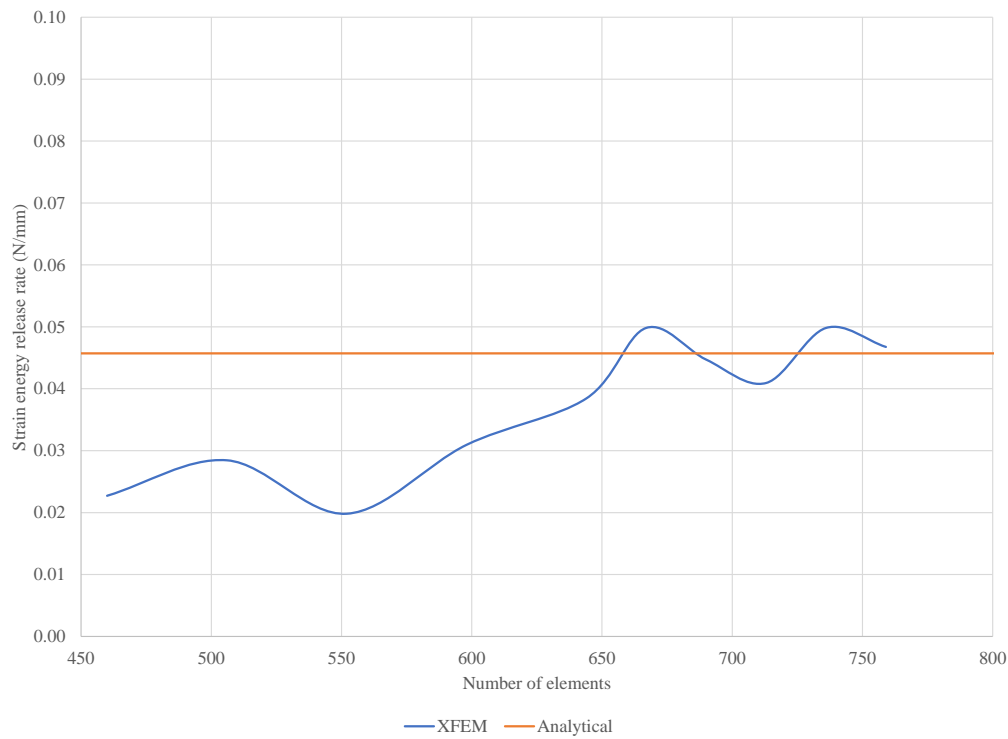


Figure 2.13: Energy release rate mesh convergence test

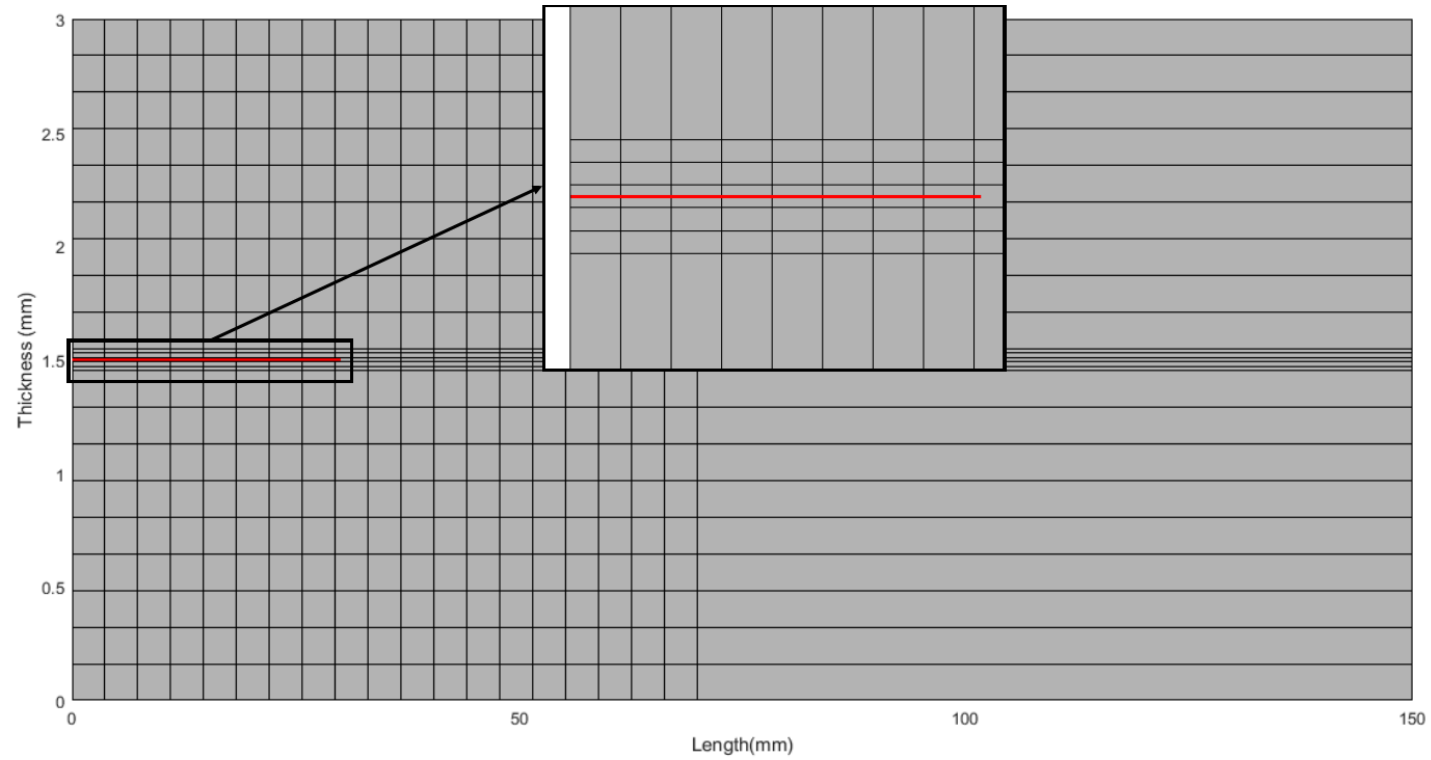


Figure 2.14: Mesh example for convergence study; convergence is improved by adding elements in the X direction

Chapter 3

Implementation and Results

Failure prediction of an adhesive bonded double cantilever beam is performed within the framework of the Extended Finite Element Method. An initial delamination is embedded within the adhesive layer and modeled independently from the mesh via a Heaviside function (strong discontinuity). The asymptotic near-tip field is modeled with Linear Elastic Fracture Mechanic based enrichment functions. A stochastic fatigue propagation model based on the Paris-Erdogan equation with the maximum strain energy as a fracture parameter is used to simulate delamination growth data. Two cases are studied, a constant and random amplitude fatigue test cases.

3.1 Fracture mechanics for the adhesive

As previously stated, the adhesive layer is modeled as a linear elastic material. The near tip functions for enriched are based on linear elastic fracture mechanics theory. However, an energy approach is used in this work to characterize the fracture state of the double cantilever beam. As such, the maximum energy release rate criteria (introduced by Nuismer in 1975) is used. According to the criteria, delamination propagation will start after the maximum energy release rate reaches a critical value (i.e. Fracture Toughness G_{cr}). The delamination will propagate in a radial direction where the energy release rate is at its maximum. However, its criteria is based on stress intensity factors.

In 1983, Nishioka proposed an equation for the energy release rate as a function of the J-integral. The energy release rate is then given by the following expression:

$$G = J_1 \cos \theta + J_2 \sin \theta \quad (3.1)$$

were the propagation angle can be derived by maximizing the above expression thus, the propagation angle is given by:

$$\theta = \arctan \left(\frac{J_2}{J_1} \right) \quad (3.2)$$

From the previous definition of the equivalent domain J-integral, the following expression for J_1 and J_2 can be derived:

$$J_k = \sum_{\Omega} \sum_n \left[\left(\sigma_{ij} \frac{\partial u_j}{\partial x_k} - w \delta_{ki} \right) \frac{\partial q}{\partial x_i} \right]_n |\mathbf{J}|_n w_n \quad (3.3)$$

3.1.1 Fatigue and crack propagation

Fracture mechanics has been used for characterization of crack propagation under cyclic loading since the 1960's with the work of Paris and Erdogan 1960 and Paris et. al. 1961. Fatigue behavior of adhesive and bonded joints using fracture mechanics has been studied since the 1970's from the works of Roderick 1975. Fatigue crack propagation was first introduced by Paris in the form of:

$$\frac{\Delta a}{\Delta N} = f(K_{max}, \beta) \quad (3.4)$$

where K is the stress intensity factor, N the number of cycles, a the crack length and $\beta = K_{min}/K_{max}$. Expressions in the literature range from (Pascoe et. al 2016):

$$\frac{da}{dN} = C \Delta K^m \quad \text{or} \quad \frac{da}{dN} = C \Delta G^m \quad \text{or} \quad \frac{da}{dN} = C G_{max}^m \quad (3.5)$$

where C and m are material constants.

From the above relations, the delamination extension Δa can be computed and the crack extended. To do so, the stress intensity factor range ΔK , the energy release rate range or max (ΔG or G_{max}) needs to be computed. Furthermore, other researchers have also proposed to normalize the energy release rate by the fracture toughness ($\Delta G/G_c$ or G_{max}/G_c) (Pascoe et. al 2013).

3.1.2 Stochastic fatigue model

It is known that fatigue crack growth is stochastic in nature (Li et. al. 2011). Where here, stochastic pertains to the random nature of the fatigue process. Hence, a stochastic approach is embedded within the code to simulate the random nature of fatigue. In this work, the Yang-Manning's model (1990) is implemented but with the modification of using the maximum strain energy instead of the stress intensity factor range ($\Delta K = K_{max} - K_{min}$). The model is described as follows:

The Paris-Erdogan model is assumed for delamination propagation with the strain energy as the fracture parameter (see Equation 3.5). However, the delamination propagation rate is multiplied by a correlation time parameter to account for the stationary random process per number of loading cycles.

$$\frac{da}{dN} = X(N)CG_{max}^m \quad (3.6)$$

The function $X(N)$ is sometimes reduced to a log-normal random variable and as such, the delamination propagation function can be rewritten as:

$$\frac{da}{dN} = x_p CG_{max}^m \quad (3.7)$$

where:

$$x_p = \lg^{-1}(-\lambda\mu_p s_z) \quad (3.8)$$

In the above equations, λ is the correction factor of the standard deviation as a function of the number of samples the model is based on and is defined as:

$$\lambda = \sqrt{\frac{n-1}{2}} \cdot \frac{\Gamma\left(\frac{n-1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)} \quad (3.9)$$

μ_p a standard normal variate for a probability p and s_z is the mean square of the experimental fatigue data defined for the experiment at hand as:

$$s_z = \sqrt{\sum_{i=1}^n \left\{ \lg\left(\frac{da}{dN}\right)_i - \lg[C(G_{max_i})^m] \right\}^2 / (n-2)} \quad (3.10)$$

The calculation for s_z was estimated from the plotted data in König et. al. 1997 for a double cantilever beam.

3.1.3 Simulation by loading cycles

In the present study, the simulations are performed via loading cycles. This means that the entire loading spectrum is subdivided into small spectra on which a delamination increment Δa is tested. In this dissertation, the simulation scheme selected is by preselecting a delamination increment and determine the cycles needed to achieve that

delamination increment by inverting the Paris-Erdogan power equation to characterize fatigue.

$$\Delta N_i = \frac{X(N)^{-1}}{C} G_{max}^{-m} \Delta a \quad (3.11)$$

In each loading cycle, the finite element simulation is performed for two load cases (a maximum and minimum load). From these solution (the maximum energy release rate G_{max} is calculated). It is important to note that crack closure effects are not considered in the present study (high loading ratios β).

The total crack extension for a delamination increment technique is a summation of crack extensions for each loading cycle or in equation form:

$$a = a_0 + k_i \Delta a \quad (3.12)$$

where k_i is the loading cycle number and Δa is the delamination increment which is a simulation input parameter. For clarification, an example of a simulation loading history for a variable amplitude case is provided in Figure 3.1 below.

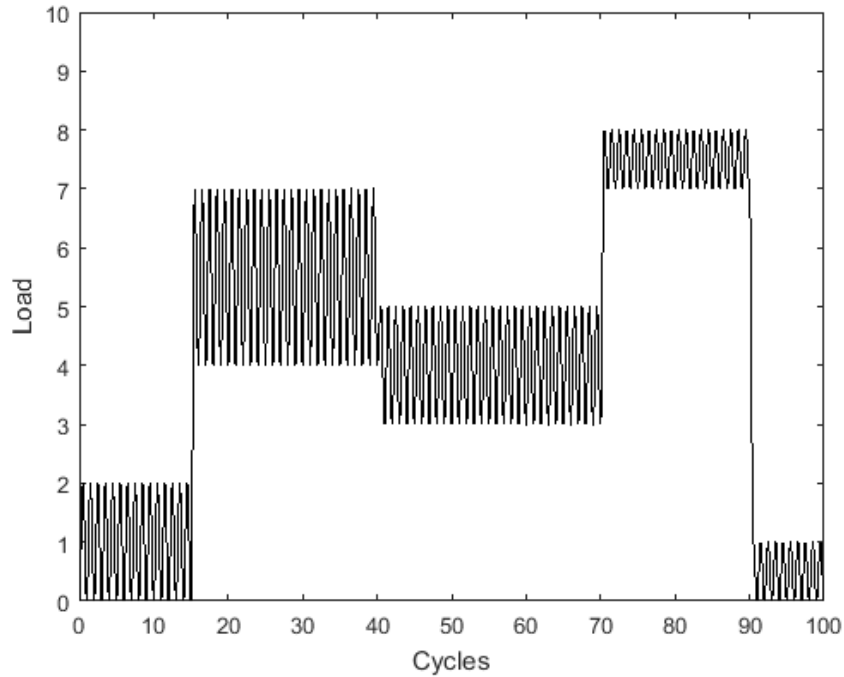


Figure 3.1: Loading history example

3.1.4 Code structure

The structure of the constructed code is shown in Figure 3.2. As previously stated, an initial delamination is embedded into the adhesive. The enriched space is then determined based on the initial delamination position. The global stiffness matrix is then determined and the energy release rate computed. If the energy release rate attains a threshold value, delamination will occur. If the energy release rate is above a critical value, failure of the adhesive will occur and the simulation is stopped. Furthermore, if the cycles computed do not attain an onset value, delamination will not occur. The simulation is stopped after final delamination length is attained or if the total number of cycles is obtained.

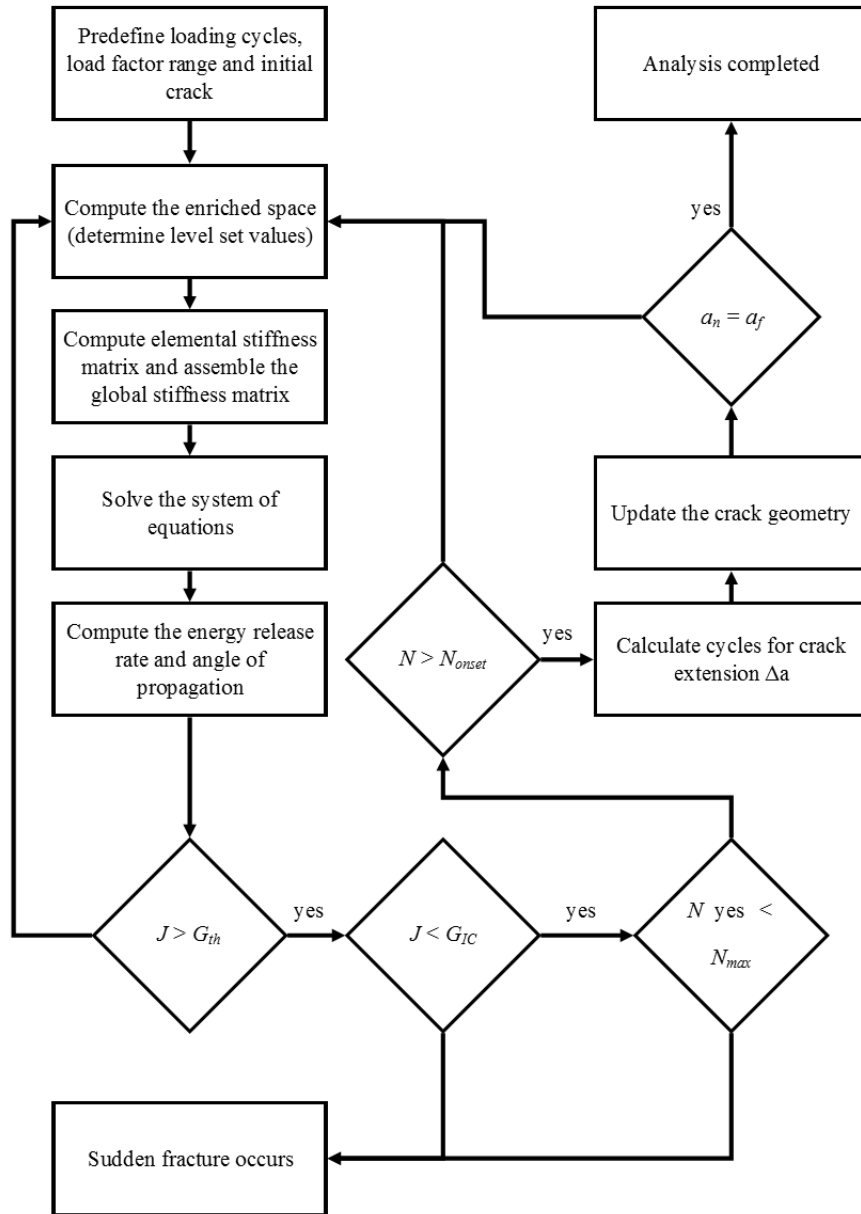


Figure 3.2: Flowchart for crack propagation

3.2 Benchmark example: graphite epoxy DCB

3.2.1 Constant amplitude loading

The developed finite element code was compared with experimental results by König et. al (1997) and numerical results published by Krueger in 2010. The material properties for the plies are the same as published in their work and summarized in Table 3.1 below for a unidirectional Graphite/Epoxy prepreg with a $[0]_{24}$ stacking sequence. However, the material properties for the adhesive were not provided in their study, thus the material properties published for the epoxy resin by Car et. al. in 2000. Refer to Tables 3.1 and 3.2 for the material properties used in the analysis.

In the benchmark example by Krueger, a displacement controlled experiment was performed with loading parameters provided in Table 3.3. Similarly, a delamination increment of $\Delta a = 0.50 \text{ mm}$ was selected for the simulation as it provided stable results in the developed MATLAB script. The test configuration is depicted below for clarity.

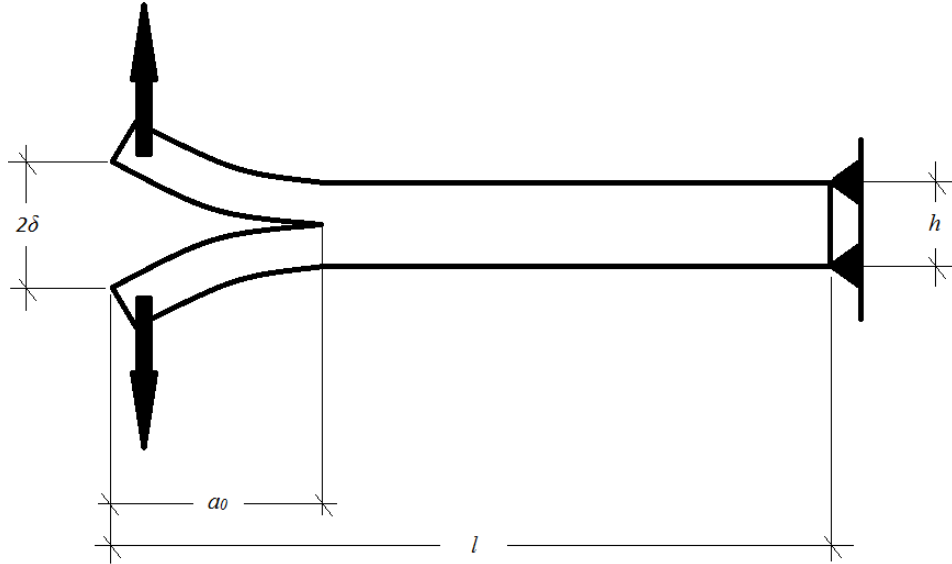


Figure 3.3: Test configuration

Table 3.1: Material properties for unidirectional Graphite/Epoxy Prepreg

Young's modulus	E_{11}	E_{22}	E_{33}
[GPa]	139.40	10.16	10.16
Shear modulus	μ_{12}	μ_{13}	μ_{23}
[GPa]	4.60	4.60	3.54
Poisson's ratio	ν_{12}	ν_{13}	ν_{23}
	0.300	0.300	0.436

Table 3.2: Material properties for epoxy resin

Young's modulus	E
[GPa]	13
Poisson's ratio	ν
	0.325

Table 3.3: Benchmark loading parameters

Maximum displacement [mm] (δ_{max})	0.67
Minimum displacement [mm] (δ_{min})	$\beta \cdot \delta_{max}$

The delamination growth rate is computed with Equation 3.6. The double cantilever geometry, as defined in Figure 3.3, has the following geometric values provided in Table 3.4 which are the same geometric parameters as provided by Krueger (2010).

Table 3.4: Geometry of DCB for benchmark example

Width (w) [mm]	25.0
Thickness (h) [mm]	3.0
Length (l) [mm]	150.0
adhesive thickness [mm]	0.1
Initial delamination (a_0) [mm]	30.0

As with the work of Krueger in 2010, delamination propagation values were extracted from the results published in 1997 by König et. al. The model proposed here is limited to region II of the fatigue delamination growth plot, which is the region governed by the Paris-Erdogan power law.

A Cutoff or threshold energy release rate of 0.060 kJ/m^2 and fracture toughness of 0.17 kJ/m^2 were also adopted into the analysis. A delamination growth increment of 0.5 mm was selected to compare the results with those published by Krueger and a maximum delamination length of 40 mm and 10,000,000 cycles were also incorporated into the analysis. The analysis was performed as a plane strain linear extended finite element simulation in MATLAB (developed code provided in the Appendix). A depiction of the mesh used for the analysis is shown in Figure 3.5. The delamination (shown in red in Figure 3.5) is mathematically embedded into the finite element via the enrichment functions. The elements depicted in blue represent the carbon/epoxy unidirectional adherends and the gray elements represent the adhesive layer.

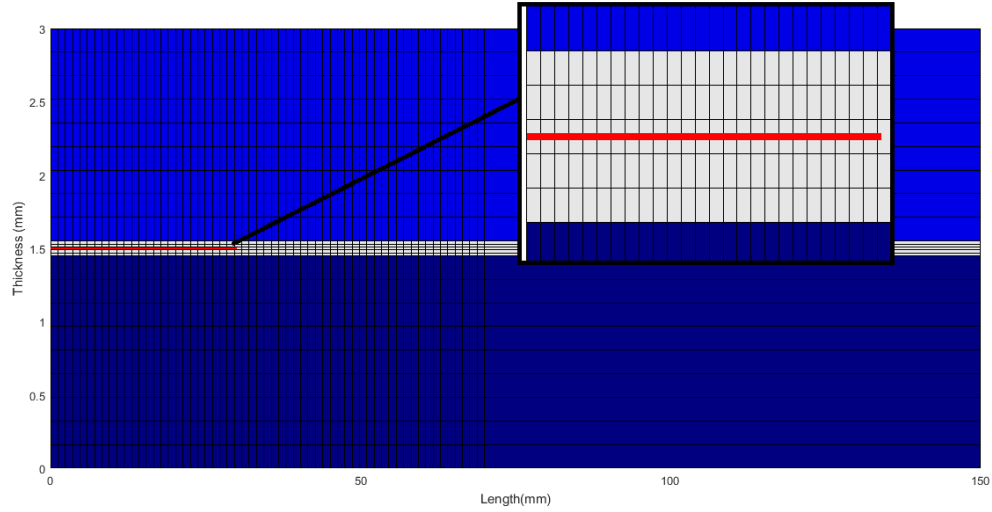


Figure 3.4: Mesh for fatigue simulation of DCB (initial delamination in red color)

The delamination growth rate closely resembles the data fit in König et. al. 1997 as plotted in Figure 3.5. The analysis consisted of 20 loading cycles for the selected delamination increment of 0.50 mm .

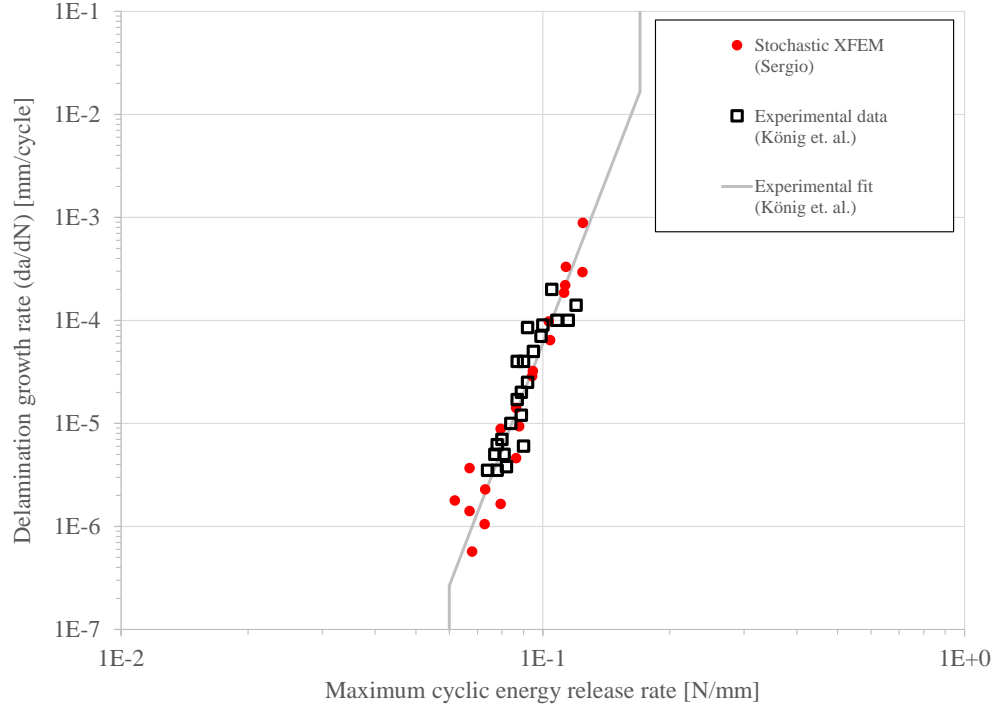


Figure 3.5: Delamination growth rate comparison with experimental data

A Von Mises contour plot of the final loaded beam is shown in Figure 3.6. It can be observed the stress distribution round the delamination front (crack tip) which compares to the expected stress distribution of a plane strain state of stress for an elastic material (Anderson 2005). However, it is important to note the effect the adherends have in the stress distribution where the stress propagates to the adherends in a considerable area.

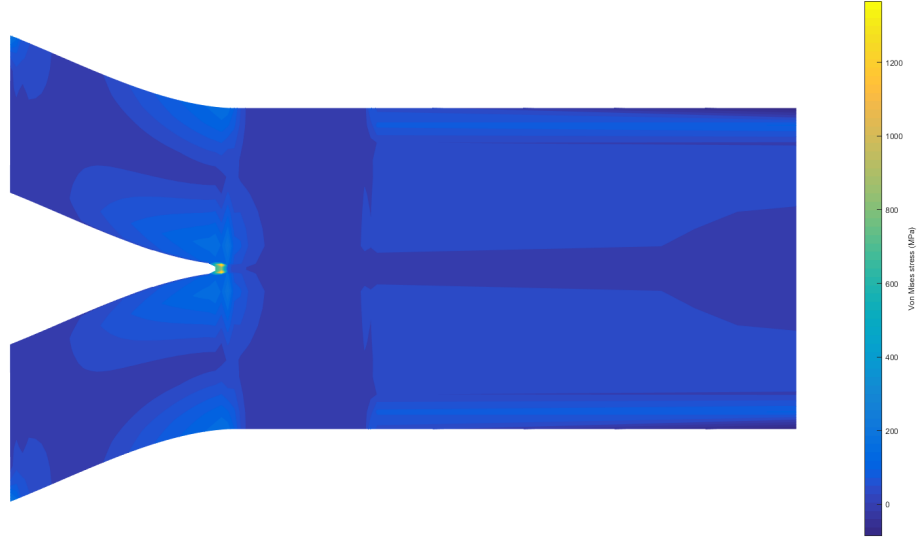


Figure 3.6: Von Mises stresses contour plot (Constant amplitude loading)

The delamination length vs. cycle are provided in Figure 3.7. The results of the stochastic constant amplitude fatigue simulation are plotted along with the results reported by Krueger in 2010 for the simulation performed in ABAQUS software with CPE4 plane strain elements. The stochastic results obtained by the constant amplitude condition compares with the delamination data for the simulation with CPE4 elements. Hence, the numerical results produced by the numerical approach in this dissertation are comparable to the numerical results using ABAQUS.

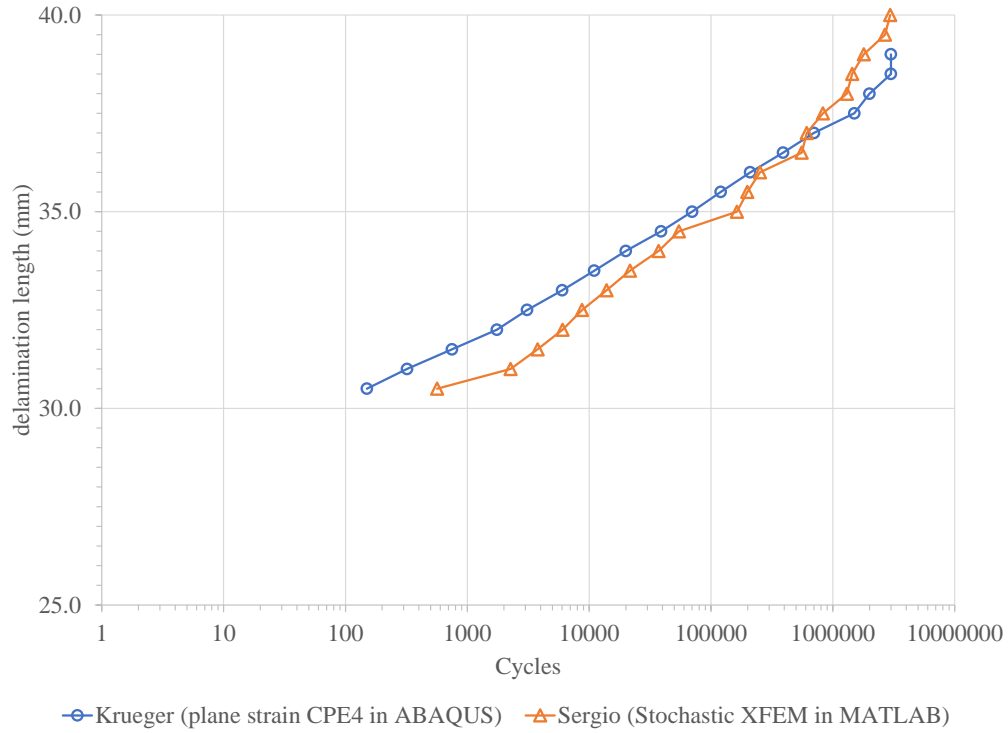


Figure 3.7: Delamination length results per cycles for constant amplitude stochastic fatigue

3.2.2 Implementation with random loading

Variable loading fatigue simulation is modeled by subjecting the structure to consecutive small constant amplitude loading spectra that differ in amplitude between each other, thus the loading spectrum can be considered of variable amplitude (Anderson 2005). If the loading values are not fixed but randomly extracted from a probability density function, it is considered in this sense random loading. The assumption being that a structure is designed for a type or magnitude of loading, but in reality, unpredicted or stochastic processes can subject the structure to random loading. For the case at hand, a simple normal distribution is selected as the probability density function of which

random data points will be extracted at each loading cycle. The average load and ratio β values are provided in Table 3.3. Several standard deviations are selected as to observe their effect on the simulation. The results of the simulation are provided in Figure 3.8. It is important to note that as the uncertainty of the input load (displacements) increases, a larger number of simulations must be carried out as some of the cycles do not produce contribute to delamination as the strain energy release rate does not exceeds the threshold release rate. Another important feature from the analysis is the limitation to model correctly regions I and III for the delamination growth plots. This can be overcome by adopting an equation for delamination growth rate that characterizes the complete sigmoidal curve. However, empirical data is needed to feed this information to the stochastic function.

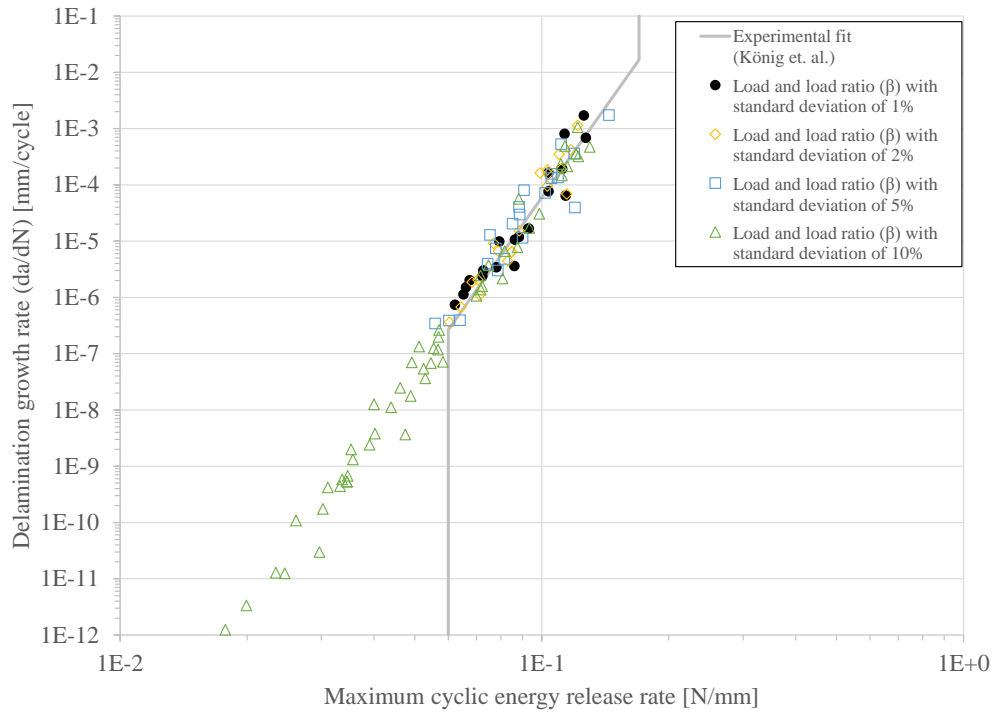


Figure 3.8: Delamination growth rate comparison with random loading

Figure 3.9 below shows a comparison of the delamination length per cycle with varying

load and load ratio standard deviations. From the figure, an increase in randomness can be seen as higher standard deviations of input parameters is increased as expected. An important feature is that a retardation effect is produced with an increase in the standard deviation of the input load meaning that the slope of the cycles vs. displacement tends to increase at the end of the delamination. This can be explained as the double cantilever beam is subjected to a larger number of loading cycles. However, a number of these loading cycles do not contribute to delamination as the energy release rates produced are below the threshold value as can be seen in Figure 3.8. However, these cycles do accumulate in the structure and thus we observe the increase in slope at the end of the delamination simulation.

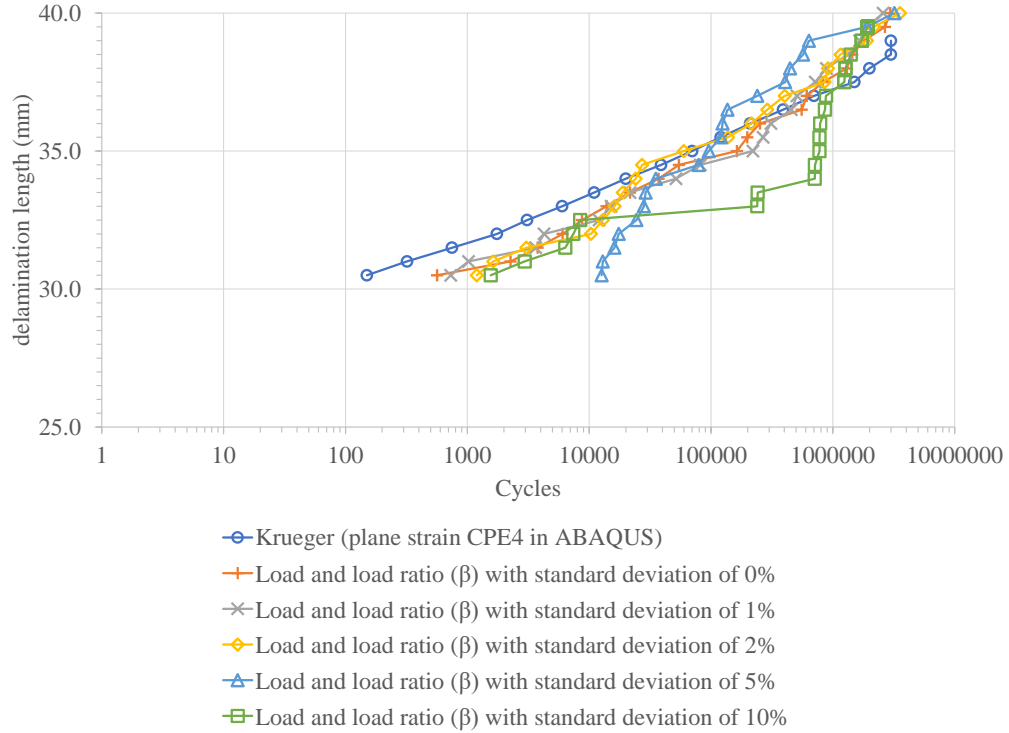


Figure 3.9: Delamination length results per cycle for random amplitude load stochastic fatigue

3.2.3 Evaluation of load variation

The impact of random loading was further evaluated by selecting three test cases: low, medium and high variation from load mean and load ratio. The amounts of deviation were selected as 1%, 5% and 10% for both load and ratio for low, medium and high test cases respectively. However, the test case scenario remains the same as for the benchmark example.

A total of 15 experiments were performed for each test case. The strain energy release results for these experiments is shown in Figures 3.10, 3.11 and 3.12 for low, medium and high load variations, respectively. For a load variation of 10%, catastrophic delamination was observed for 5 of the 15 experiments run. This indicates that unstable delamination can occur for random loads above 10% of the selected configuration.

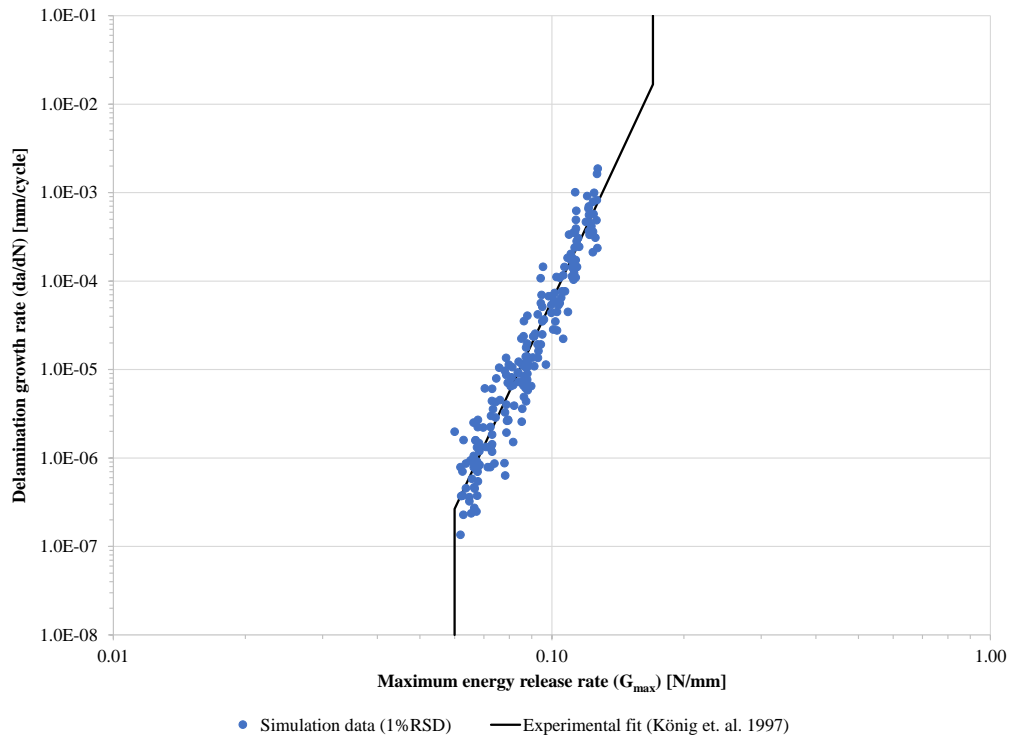


Figure 3.10: Energy release rate for low deviation test case

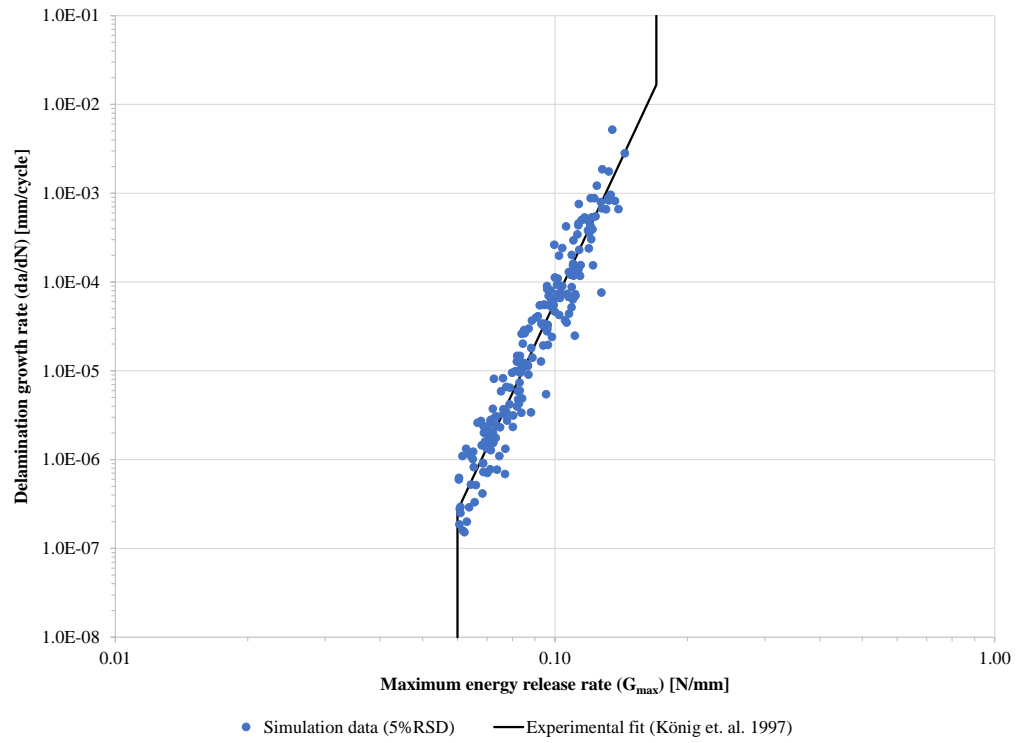


Figure 3.11: Energy release rate for medium deviation test case

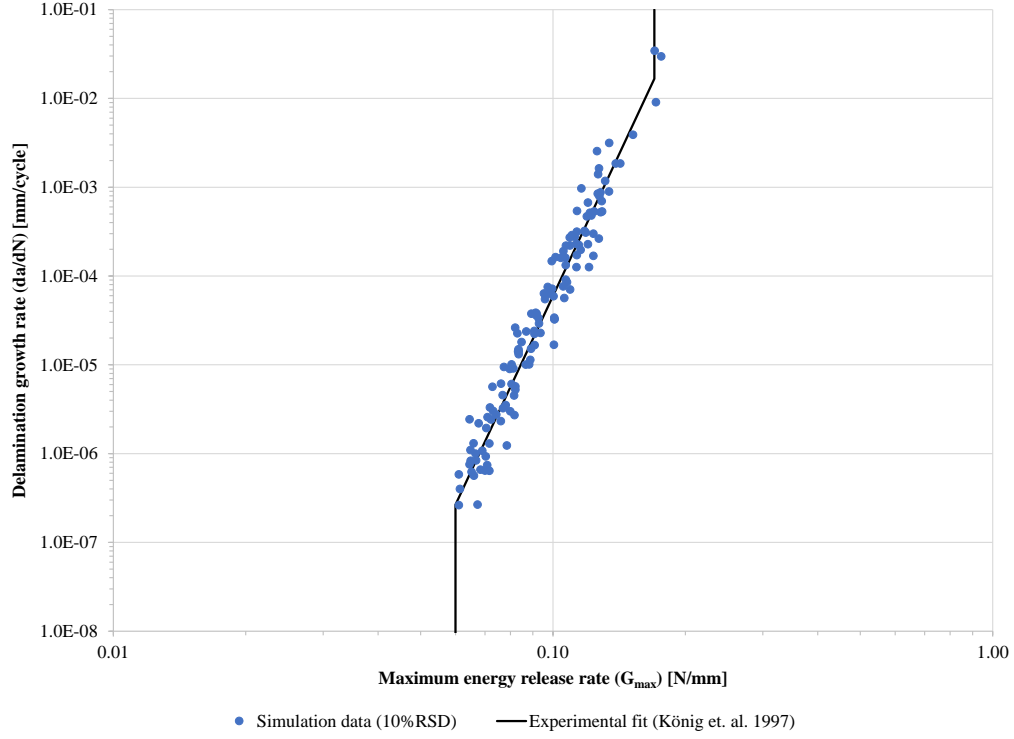


Figure 3.12: Energy release rate for high deviation test case

The final cycles to failure obtained for each test case were extracted from the result data set and plotted in a probability plot, refer to Figures 3.13, 3.14 and 3.15 for the low, medium and high variable loading cases respectively. The obtained values follow a normal distribution as shown for the high p-values and low Anderson-Darling statistics. However, the results for the high variable loading test case are marginally described by a normal distribution due to its low p-value. This might be attributed to the extreme cases of catastrophic failure (i.e. low cycles to failure observed for 5 samples).

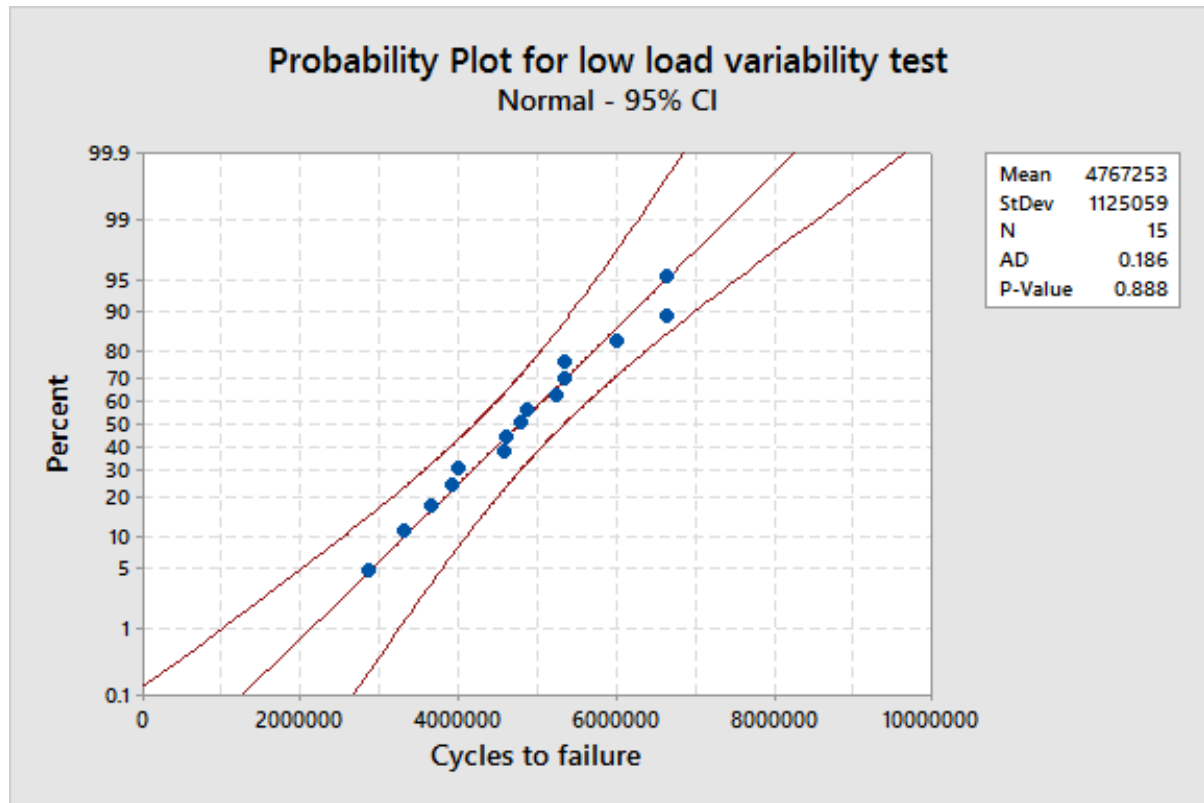


Figure 3.13: Final cycles to failure for low deviation test case

Low variability in load test results are shown in Figure 3.13. The predicted cycles to failure by the developed code for the low variability test follow a normal distribution. The cycles to adhesive failure are estimated to be 4,767,253 cycles for the sample set.

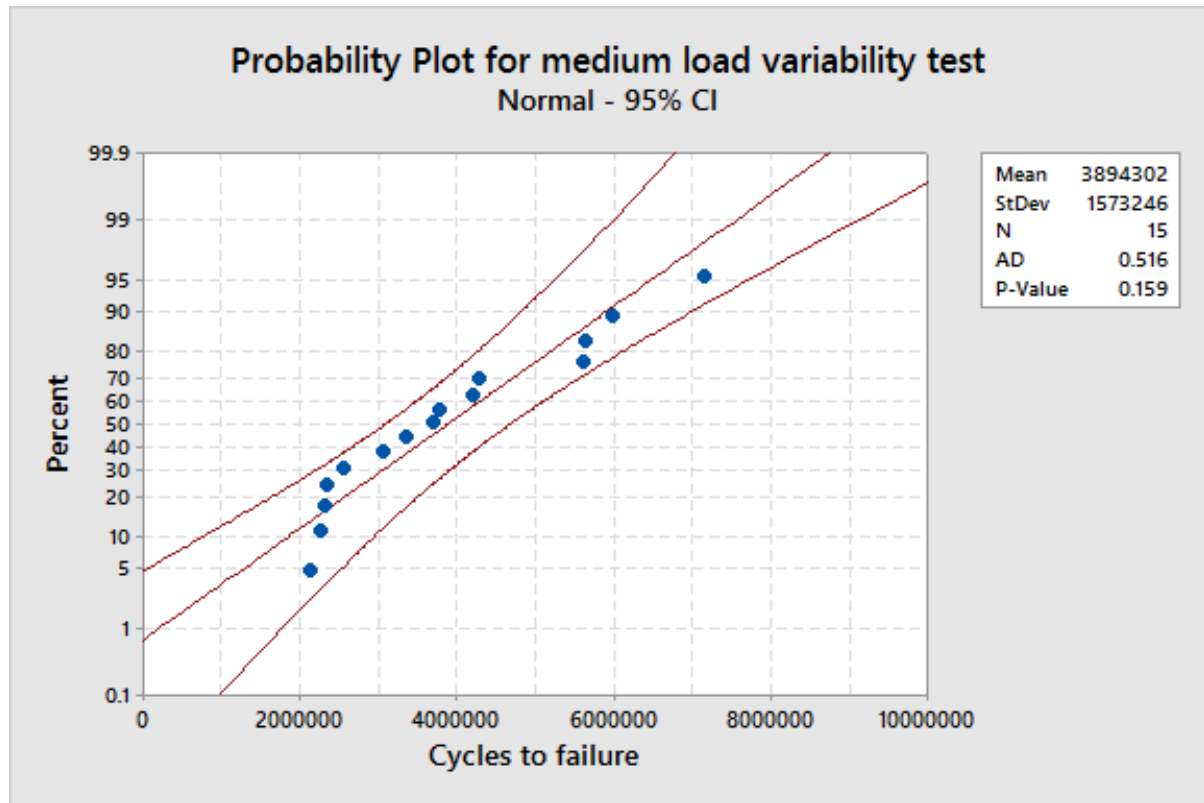


Figure 3.14: Final cycles to failure for medium deviation test case

Medium variability in load test results are shown in Figure 3.14. The predicted cycles to failure by the developed code for the medium variability test follow a normal distribution. The cycles to adhesive failure are estimated to be around 3,894,302 cycles for the sample set.

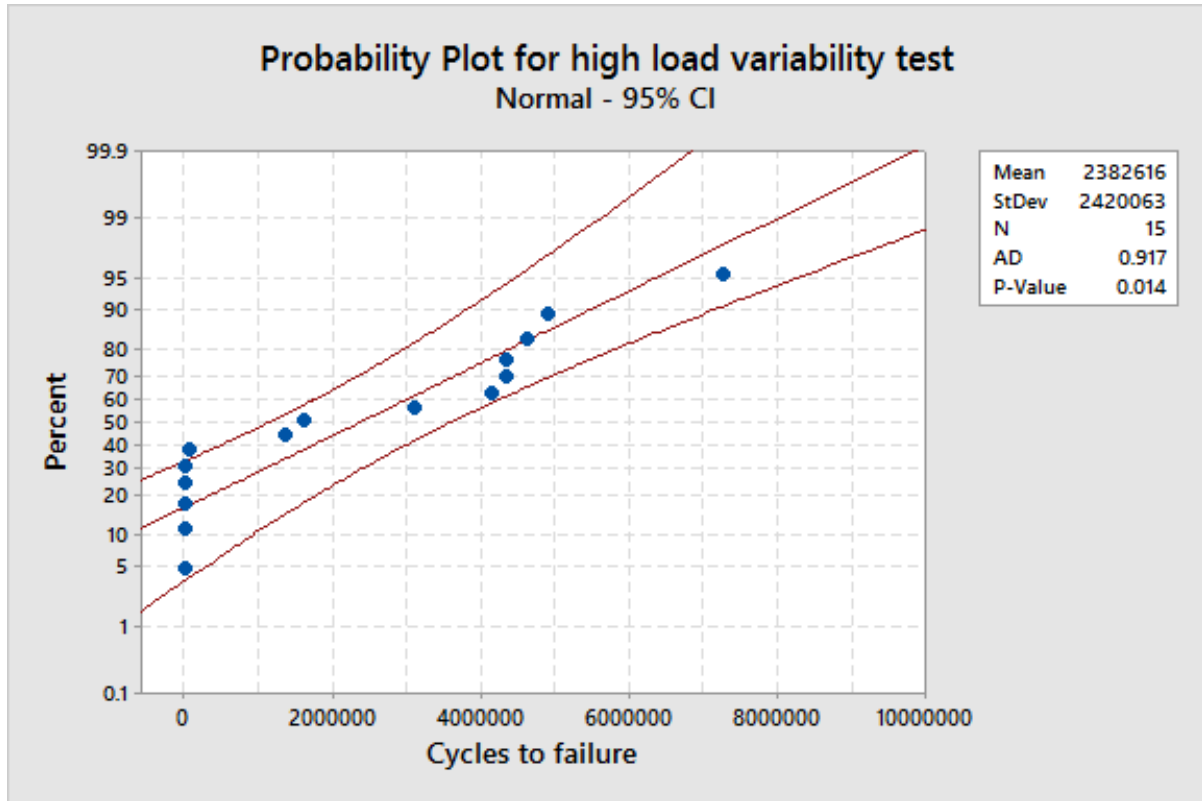


Figure 3.15: Final cycles to failure for high deviation test case

High variability in load test results are shown in Figure 3.15. The cycles to adhesive failure are estimated to be around 2,382,616 cycles for the sample set which is significantly less than for the low and medium variability test cases. However, the test shows that the data do not follow a normal distribution due to the low p-value (< 0.05). This is attributed to sudden or catastrophic failure of the adhesive as observed for 5 of the samples.

3.2.4 Evaluation of seed influence in random data generation

The previously presented data was based on default random number generation settings within MATLAB. In order to evaluate the influence in the simulated random data, two different seeds were randomly selected using the *rng* function within MATLAB with the 'shuffle' input argument. This sets the seed for random number generation to a value based on the current computer time. The two cases selected were ran at 5% deviation

from loading and loading ratio input means. Figure 3.16 shows a box plot of the data with the default seed of 0 and the seeds randomly selected. A one-way ANOVA test was performed to determine if there is a statistical difference for the predicted cycles to failure due to a change in seed. The results shown in Table 3.5 shows a p-value greater than 0.05 thus there is no statistical difference between the average cycles to failure between the tested seed values. Therefore, the influence of the seed selection for random analysis is deemed not statistically significant.

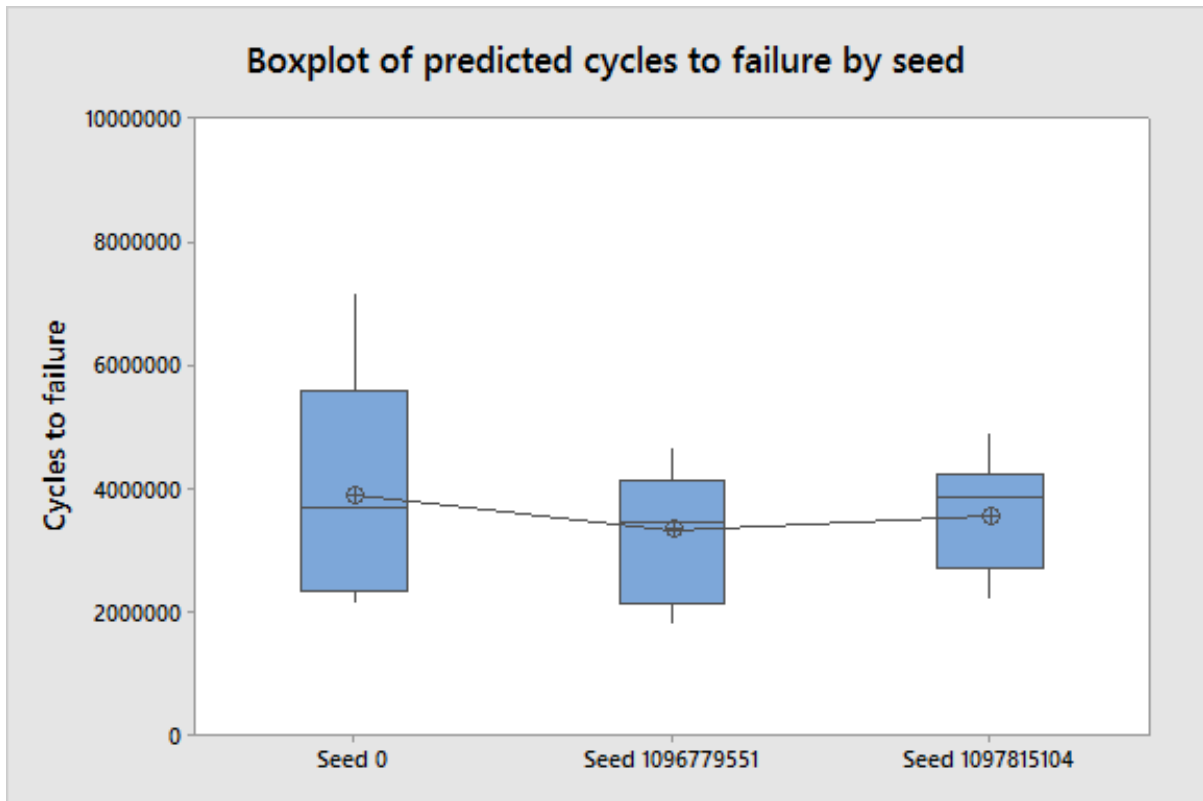


Figure 3.16: Cycles to failure prediction comparison by seed

Table 3.5: One-way ANOVA for comparison of seed variation

Source	Degrees of Freedom	Adj. Sum of Squares	Adj. Mean Square	F-Value	P-Value
Factor	2	2.40060×10^{12}	1.20030×10^{12}	0.90	0.415
Error	42	5.61838×10^{13}	1.33771×10^{12}		
Total	44	5.85844×10^{13}			

Chapter 4

Final Remarks

4.1 Conclusion

In this work, the Extended Finite Element Method was used to study the stochastic fatigue delamination in a composite adhesive bonded joint. The configuration selected was a double cantilever beam made of graphite/epoxy (T300/914C) unidirectional composite with a $[0]_{24}$ stacking sequence. A double cantilever beam is a commonly used configuration to study delamination in composites and strength of the adhesive (Banea and da Silva 2009, Biel and Stigh 2007). The adhesive was modeled as an isotropic, linear elastic material confined between two orthotropic linear elastic adherends. The cohesive delamination was modeled by enriching the adhesive layer with extra degrees of freedom to include the crack influence, independently from the mesh. The stochastic nature of the fracture process was modeled using a modification of the Yang-Manning's model but using the maximum strain energy release rate as the fracture parameter with good estimates when compared with the experimental data by Krueger in 2010.

The implementation of incompatible element "Q6" (first introduced by Wilson et. al. in 1973) was successfully tested with the Extended Finite Element Method. The developed code in MATLAB was tested against the benchmark results published by Krueger in 2010 and experimental data published by König et. al. in 1997 with comparable results. Finally, the double cantilever system was subjected to random loading conditions in which both the stress ratio and load are randomly extracted values from a normal probability distribution. The random load values are determined given an average load, load ratio and standard deviation. The stochastic fatigue analysis is carried out in loading blocks with a predetermined delamination increment and damage accumulation calculated. The use of quadrilateral finite elements for modeling of the DCB proved to be, albeit possible, inefficient as they become overly stiff in bending simulation and thus require a larger number of elements in the longitudinal direction for the solution to converge.

The work presented here provides a unique combination of the XFEM with both a stochastic model for fatigue delamination and random loading scheme. Testing of different levels of deviation from average loading shows a reduction in average cycles to failure with an increase in deviation. These results are in accordance with the expected behavior as high loads can be produced with an increase in deviation from the average load that will induce high energy release rates. This will in turn translate to a delamination process for relatively low number of cycles.

Normal distributions for cycles to failure were found for 1%, 5% and 10% load deviations. However, marginal normality test results were observed for high load deviations (10%). This is attributed to the high number of tests (5 of 15) on which the adhesive bonded joint failed catastrophically (sudden delamination with low number of cycles). Hence for high load variations, the average cycles to failures is found to be significantly lower than for the other load deviations tested (in the order of 2,000,000 cycles lower than for the 1% deviation case).

The developed tool will prove valuable for cost reductions in the development process of bonded joints. A reduction in test samples is foreseen as numerous test cases, with different levels of load dispersion and mean values can be tested numerically. Thus, this reduction in cost in testing can be beneficial to companies in the aerospace industries as they rely more through the years in composites and adhesive bonded joints for the construction of aircrafts.

4.2 Recommendations

The algorithm was successful in simulating delamination of a double cantilever beam in fatigue loading. However, this code was developed using a linear formulation of the extended finite element. Relatively high deformation due to bending can be present for higher load magnitudes hence, it is recommended to reformulate the code for non-linear analysis if higher loads are to be tested. Furthermore, it was demonstrated that a change in seed for the randomization of the load produces comparable average life cycle predictions. However, it is recommended to randomize the seed selection every time a new session is started to minimize the influence of the seed when generating random data.

4.3 Future Work

There are many areas of improvement in this work to model the strength of the adhesive layer. One improvement would be to include the adhesive force as a model parameter thus allowing the delamination to propagate between the adhesive and adherend. As the model is limited by Linear Elastic Fracture Mechanics, future work should focus on studying plastic deformation of the adhesive layer. Furthermore, the simulation was limited to linear finite element theory hence, a great area of opportunity exists to expand the model to nonlinear simulations. Another area of improvement is to implement higher order elements to study their effect in a double cantilever beam configuration and to improve convergence.

This work can be expanded to Mode II failure or mixed mode between Mode I and II (refer to Figure 2.11) as minimal work may be required for the generalization of the code to other bonded joint configurations (Figure 2.1). Furthermore, other materials, e.g. different adhesives and metallic adherends can be tested to evaluate their effect in stress distribution and consequently in their fatigue life.

The analysis performed in this work could be improved by establishing a joint effort with a materials testing laboratory to obtain real results and compare them with the finite element model. A good stochastic model needs to be fed analytical data to successfully simulate the variability in fatigue simulation.

Appendix A

MATLAB scripts

Input geometry script

```

1 function [NODE,ELEMENT,BC,CRACK,SIM,MATERIAL] = finput_DCB()
2 %finput_DCB DCB test case.
3 % This function is a test case for an aluminum Double
   Cantilever beam with
4 % an embeded crack.
5 %
6 % * +F
7 % |
8 % |
9 % o - - - - - o<|
10 % |                Material 3                |
11 % | ~~~~~~|
12 % |                |
13 % + - - - - - Material 1                |
14 % |                |
15 % | ~~~~~~|
16 % |                Material 2                |
17 % o - - - - - o<|
18 % |
19 % |
20 % * -F
21 %
22 % Node numbering scheme
23 % 4          3
24 % o - - - - o
25 % |          |
26 % |          |
27 % |          |

```

```

28 %   o - - - - o
29 %   1           2
30
31 % Space discretisation
32 adhesive = [1.45 1.55];
33 a0        = 30; % initial crack
34 SIM.a0     = a0;
35
36 x_1 = linspace(0,70,60);
37 x_2 = linspace(70,150,2);
38 y_1 = linspace(0,adhesive(1),10);
39 y_2 = linspace(adhesive(1),adhesive(2),6);
40 y_3 = linspace(adhesive(2),3,10);
41
42 % Space characterization
43 x = unique([x_1 x_2]);
44 y = unique([y_1 y_2 y_3]);
45 nn_x = length(x);
46 nn_y = length(y);
47 ne_x = nn_x - 1;
48 ne_y = nn_y - 1;
49 nn    = nn_x*nn_y; % number of nodes
50 ne    = ne_x*ne_y; % number of elements
51 fprintf('(!) Model with %i elements in X direction.\n',
52         length(x)-1);
53
54 fprintf('(!) Model with %i elements in Y direction.\n',
55         length(y)-1);
56
57 % Generate mesh
58 [xn,yn,CMAT] = mesher(x,y);
59
60 % Elements in material 2
61 x1 = [0 max(x) max(x) 0 0];
62 y1 = [0 0 1.4 1.4 0];
63
64 % Elements in material 3
65 x3 = [0 max(x) max(x) 0 0 ];
66 y3 = [1.6 1.6 max(y) max(y) 1.6];

```

```

64
65 % Elements in material 1
66 x2 = [0          max(x)  max(x)   0          0          ];
67 y2 = [max(y1)  max(y1) min(y3)  min(y3) max(y1)];
68
69 % Element centroid
70 if size(CMAT,1) == 1
71     mean_dim = 1;
72 else
73     mean_dim = 2;
74 end
75 elem.xc = mean(xn(CMAT),mean_dim);
76 elem.yc = mean(yn(CMAT),mean_dim);
77
78 % Topology matrix (used for plotting solutions)
79 topo = zeros(nn_y,nn_x);
80 c1 = -nn_x;
81 for n1 = nn_y:-1:1
82     c1 = c1 + nn_x;
83     topo(n1,:) = (1:nn_x) + c1;
84 end
85
86 % Find elements in material
87 in1 = inpolygon(elem.xc,elem.yc,x1,y1);
88 in2 = inpolygon(elem.xc,elem.yc,x2,y2);
89 in3 = inpolygon(elem.xc,elem.yc,x3,y3);
90
91 % Storing mesh solutions
92 NODE.X          = xn;
93 NODE.Y          = yn;
94 NODE.ID         = (1:nn)';
95 SIM.CONMAT      = CMAT;
96 SIM.TOPOGRAPHY = topo;
97
98 ELEMENT(ne) = struct('ID',0,'NODES',[0,0,0,0],'MATERIAL',
    uint16(0));
99 for n1 = 1 : ne
100     ELEMENT(n1).NODES      = SIM.CONMAT(n1,:);

```

```

101     ELEMENT(n1).ID      = n1;
102 end
103
104 % Assigning material to elements
105 [ELEMENT(in1).MATERIAL] = deal(2);
106 [ELEMENT(in2).MATERIAL] = deal(1); % Adhesive
107 [ELEMENT(in3).MATERIAL] = deal(3);
108
109 % Crack
110 CRACK.POINT = [1 1]';
111 CRACK.X      = [min(x) min(x)+a0]';
112 CRACK.Y      = [1 1]'*(max(y)+min(y))/2;
113
114 % Boundary conditions
115 f1 = NODE.ID(NODE.X == max(x) & NODE.Y == min(y));
116 f2 = NODE.ID(NODE.X == max(x) & NODE.Y == max(y));
117 f3 = NODE.ID(NODE.X == min(x) & NODE.Y == min(y));
118 f4 = NODE.ID(NODE.X == min(x) & NODE.Y == max(y));
119
120 BC.DISP.NODE = [f1;f2]';
121 BC.DISP.UX   = zeros(1,length([f1;f2]')); % m
122 BC.DISP.UY   = zeros(1,length([f1;f2]')); % m
123
124 BC.FORCE.NODE = [f3 f4];
125 BC.FORCE.FX   = [ 0 0 ]; % N
126 BC.FORCE.FY   = [-1 1 ]; % N
127
128 % Simulation values
129 SIM.THICKNESS = 100; % mm
130 SIM.LOADING   = 'plane strain';
131 SIM.NODES     = nn;
132 SIM.ELEMENTS  = ne;
133
134 % Material 4 (reference material)
135 MATERIAL(4).NAME = 'aluminum';
136 MATERIAL(4).TYPE = 'isotropic';
137 MATERIAL(4).V    = 0.33;
138 MATERIAL(4).E    = 70000; % MPa

```

```
139 MATERIAL(4).C      = 1.5E-10;
140 MATERIAL(4).m      = 3.8;
141
142 % Material 1
143 % Epoxy resin
144 % Material properties from:
145 % An anisotropic elasto-plastic constitutive model for large
    strain
146 % analysis of fiber reinforced composite material" E. Car
    2000
147 MATERIAL(1).NAME = 'epoxy';
148 MATERIAL(1).TYPE = 'isotropic';
149 MATERIAL(1).V      = 0.325;
150 MATERIAL(1).E      = 26E3;
151 MATERIAL(1).C      = 2.44E6;
152 MATERIAL(1).m      = 10.61;
153 MATERIAL(1).Gcri   = 0.17; % N/mm
154
155 % Material 2
156 MATERIAL(2).NAME = 'graphite/epoxy';
157 MATERIAL(2).TYPE = 'orthotropic';
158 MATERIAL(2).V12   = 0.30;
159 MATERIAL(2).V21   = 0.30;
160 MATERIAL(2).V13   = 0.30;
161 MATERIAL(2).V31   = 0.30;
162 MATERIAL(2).V23   = 0.436;
163 MATERIAL(2).V32   = 0.436;
164 MATERIAL(2).E1    = 139.4e3;
165 MATERIAL(2).E2    = 10.16e3;
166 MATERIAL(2).E3    = 10.16e3;
167 MATERIAL(2).G12   = 4.6e3;
168
169 % Material 3
170 MATERIAL(3).NAME = 'graphite/epoxy';
171 MATERIAL(3).TYPE = 'orthotropic';
172 MATERIAL(3).V12   = 0.30;
173 MATERIAL(3).V21   = 0.30;
174 MATERIAL(3).V13   = 0.30;
```

```

175 MATERIAL(3).V31 = 0.30;
176 MATERIAL(3).V23 = 0.436;
177 MATERIAL(3).V32 = 0.436;
178 MATERIAL(3).E1 = 139.4e3;
179 MATERIAL(3).E2 = 10.16e3;
180 MATERIAL(3).E3 = 10.16e3;
181 MATERIAL(3).G12 = 4.6e3;
182
183 % Defining material matrices for analysis
184 for n1 = 1 : size(MATERIAL,2)
185     switch MATERIAL(n1).TYPE
186         case('isotropic')
187             E = MATERIAL(n1).E;
188             v = MATERIAL(n1).V;
189             G = E/(2*(1+v));
190             switch SIM.LOADING
191                 case('plane stress')
192                     E1 = E/(1-v^2);
193                     E2 = v*E1;
194                 case('plane strain')
195                     E1 = E*(1-v)/((1+v)*(1-2*v));
196                     E2 = v*E1/(1-v);
197                 otherwise
198                     error('Undefined test case')
199             end
200             MATERIAL(n1).G = G; % Shear modulus
201             MATERIAL(n1).D = [E1 E2 0;E2 E1 0;0 0 G]; %
202             % Material matrix
203             case('orthotropic')
204                 EX = MATERIAL(n1).E1;
205                 EY = MATERIAL(n1).E2;
206                 vXY = MATERIAL(n1).V12;
207                 vYX = MATERIAL(n1).V21;
208                 vYZ = MATERIAL(n1).V23;
209                 vZY = MATERIAL(n1).V32;
210                 vXZ = MATERIAL(n1).V13;
211                 vZX = MATERIAL(n1).V31;
212                 GXY = MATERIAL(n1).G12;

```



```

212         switch SIM.LOADING
213             case('plane stress')
214                 d = 1-vXY*vYX;
215                 E1 = EX;
216                 E2 = vXY*EX;
217                 E3 = vYX*EY;
218                 E4 = EY;
219                 E5 = d*GXY;
220             case('plane strain')
221                 d = (1-vXZ*vZX)*(1-vYZ*vZY)-(vXY+vXZ*vZY)*(
222                     vYX+vYZ*vZX);
223                 E1 = (1-vYZ*vZY)*EX;
224                 E2 = (vXY+vXZ*vZY)*EX;
225                 E3 = (vYX+vYZ*vZX)*EY;
226                 E4 = (1-vXZ*vZX)*EY;
227                 E5 = d*GXY;
228             otherwise
229                 error('Undefined test case')
230         end
231     MATERIAL(n1).G = (1+vYX)/EX + (1+vXY)/EY; %
232     % Approximated
233     MATERIAL(n1).D = 1/d*[E1 E2 0;E3 E4 0;0 0 E5];
234     otherwise
235         error('Undefined material type')
236     end
237 end
238
239 % Plotting mesh
240 XY = [xn yn];
241 clear figure
242 patch('Faces',CMAT(in2,:), 'Vertices',XY, 'FaceColor', [1 1
243     1]*0.9);
244 hold on
245 patch('Faces',CMAT(in1,:), 'Vertices',XY, 'FaceColor', [0 0
246     1]*0.5);
247 patch('Faces',CMAT(in3,:), 'Vertices',XY, 'FaceColor', [0 0
248     1]*0.9);
249 plot(CRACK.X,CRACK.Y, '-r')

```

```
245 hold off
246 xlabel('Length(mm)')
247 ylabel('Thickness (mm)')
248 %axis equal
249 end
```

Main script

```

1  % Clear memory and command window
2  clc; clear all; close all; format short;
3  fprintf('(!) Program started.\n')
4  time = tic;
5
6  % Extracting input mesh and materials and simulation
   parameters
7  fprintf('(P) Extracting domain inputs...\n')
8  [NODE,ELEMENT,BC,CRACK,SIM,MATERIAL] = finput_DCB();
9  pause(2)
10
11 % Test case parameters
12 SIM.RSD = 5;
13 % Load input value [N]
14 SIM.LOAD.AVG = 50;
15 SIM.LOAD.STD = SIM.LOAD.AVG * SIM.RSD/100;
16
17 % Displacement load input [mm]
18 SIM.DISP.AVG = 0.3;
19 SIM.DISP.STD = SIM.DISP.AVG * SIM.RSD/100;
20
21 % Loading ratio input
22 SIM.R.AVG = 0.1;
23 SIM.R.STD = SIM.R.AVG * SIM.RSD/100;
24
25 % Mode mixity
26 SIM.MIXITY = 1;
27
28 % Fatigue parameters
29 SIM.CASE = 'displacement control';
30 SIM.da = .5; % mm
31 SIM.a_end = 40; % final crack
   length
32 SIM.N_onset = 150; % cycles
33 SIM.Fail = 10E6; % cycles
34 SIM.Gth = 0.06; % N/mm 0.06

```

```

35 SIM.stochastic      = 0;
36 % Optimization parameters
37 SIM.SUBCELLS        = 10;                                % Subcell
    parameter
38 SIM.enrichr          = 0;                                % Radius or
    periphery levels for tip enrichment
39 SIM.ELEMENT_TYPE    = 'Q4';
40
41 total_blocks = (SIM.a_end - SIM.a0)/SIM.da;
42
43 % Initialization of variables
44 ERR          = 0;
45 theta        = 0;
46 SIM.lock     = 0;
47
48 % Dirichlet BC
49 d0_x = BC.DISP.UX;
50 d0_y = BC.DISP.UY;
51
52 %% Determination of the enriched space
53 % First the level sets for the crack based on the last
    segment of the crack
54 % in CRACK structure are calculated. A new field within NODE
    called PSI is
55 % created to store the Psi level set values at the nodes (
    Normal distance).
56 % Similarly, a new field called PHI is created to store the
    nodal Phi
57 % values (tangential distance).
58 fprintf('(P) Finding enriched space...\n')
59 [NODE.PSI,NODE.PHI] = fgeo_signed(NODE.X,NODE.Y,CRACK.X([1
    end]),CRACK.Y([1 end]));
60 [NODE.R,NODE.O]     = fgeo_polarmap(NODE.X,NODE.Y,CRACK.X([1
    end]),CRACK.Y([1 end]));
61
62 %The node identification is based
63 % on the following convention:
64 % Standard node     = 0

```

```

65 % Heaviside node = 1
66 % Near-tip node = 2
67 % The element identification is then based on the following
    convention:
68 % Standard element = 0
69 % Heaviside element = 1
70 % Near-tip element = 2
71 % Blending element = 3
72 [ELEMENT(:).TYPE] = deal(0); % Setting all elements to
    standard FEM
73 NODE.TYPE = zeros(SIM.NODES,1); % All nodes to 0
74 [NODE,ELEMENT] = fxfem_enrich(SIM,NODE,ELEMENT,CRACK,'
    periphery',SIM.enrichr);
75
76 %% Plot the enriched domain
77 figure(1)
78 hold on
79 fplot_nodes(NODE,ELEMENT,SIM);
80 hold off
81 hold on
82 plot(CRACK.X,CRACK.Y,'-xr')
83 hold off
84 pause(5)
85 %% Calculation of Degrees of Freedom and connectivity
    vectors
86 for n1 = 1 : SIM.ELEMENTS
87     ELEMENT(n1).CONVEC = fxfem_dofs(ELEMENT(n1).NODES,SIM.
        NODES,'all');
88     ELEMENT(n1).DOF.U = fxfem_dofs(ELEMENT(n1).NODES,SIM.
        NODES,'standard')';
89     ELEMENT(n1).DOF.A = fxfem_dofs(ELEMENT(n1).NODES,SIM.
        NODES,'heaviside')';
90     ELEMENT(n1).DOF.B = fxfem_dofs(ELEMENT(n1).NODES,SIM.
        NODES,'neartip')';
91 end
92 SIM.a = SIM.a0;
93 SIM.N = 0;
94 nblocks = 0;

```

```

95 DATA      = struct('ID',zeros(total_blocks,1),...
96              'Damage',0,...
97              'D',zeros(total_blocks,1),...
98              'Di',zeros(total_blocks,1),...
99              'Ni',zeros(total_blocks,1),...
100             'dN',zeros(total_blocks,1),...
101             'ai',zeros(total_blocks,1),...
102             'da',zeros(total_blocks,1),...
103             'dadN',zeros(total_blocks,1),...
104             'Gmax',zeros(total_blocks,1),...
105             'avgangle',zeros(total_blocks,1),...
106             'Force',zeros(total_blocks,1),...
107             'MAXLOAD',zeros(total_blocks,1),...
108             'R',zeros(total_blocks,1));
109
110 while SIM.a < SIM.a_end % Loop over loading blocks
111     nblocks = nblocks + 1;
112
113     % Test variables (inputs)
114     R = normrnd(SIM.R.AVG,SIM.R.STD);
115     switch SIM.CASE
116         case('load control')
117             Pavg = normrnd(SIM.LOAD.AVG,SIM.LOAD.STD);
118             TEST.MINL = 2*R/(1-R) .* Pavg;
119             TEST.MAXL = 2 / (1-R) .* Pavg;
120
121         case('displacement control')
122             Davg = normrnd(SIM.DISP.AVG,SIM.DISP.STD);
123             TEST.MIND = 2*R/(1-R) * Davg;
124             TEST.MAXD = 2 / (1-R) * Davg;
125
126         otherwise
127             error('Undefined loading control')
128     end
129
130     % Updating level sets
131     if nblocks > 1 && size(CRACK.X,1) > 2
132         cx_1 = CRACK.X(end-2);

```

```

133     cx_2 = CRACK.X(end-1);
134     cx_3 = CRACK.X(end);
135     cy_1 = CRACK.Y(end-2);
136     cy_2 = CRACK.Y(end-1);
137     cy_3 = CRACK.Y(end);
138     x     = NODE.X;
139     y     = NODE.Y;
140     F     = [cx_3 - cx_2; cy_3 - cy_2; 0];
141     V     = [cx_2 - cx_1; cy_2 - cy_1; 0];
142     a     = cross(V,F);
143     phi_rotated = (x-cx_3)*F(1)/norm(F) + (y-cy_3)*F(2)/
        norm(F);
144     for n1 = 1 : SIM.NODES
145         if NODE.PHI(n1) > 0 && F(2) ~= 0
146             NODE.PSI(n1) = -sign(a(3))*((x(n1)-cx_3)*F
                (1)/norm(F) - (y(n1)-cy_3)*F(2)/norm(F));
147         end
148         NODE.PHI(n1) = phi_rotated(n1);
149     end
150     clear cx_1 cx_2 cx_3 cy_1 cy_2 cy_3 x y
151 end
152 [NODE,ELEMENT] = fxfem_enrich(SIM,NODE,ELEMENT,CRACK,'
    periphery',SIM.enrichr);
153
154 % Domain integration
155 fprintf('(P) Integrating...\n')
156 [ELEMENT,IP] = stiffness(SIM,ELEMENT,NODE,CRACK,MATERIAL
    );
157
158 %% Determination of maximum load
159 fprintf('(M) Block %i/%i...\n',nblocks,total_blocks)
160 switch SIM.CASE
161     case('load control')
162         if TEST.MAXL > TEST.MINL
163             maxload = TEST.MAXL;
164         else
165             maxload = TEST.MINL;
166         end

```

```

167         BC.FORCE.FY = [-SIM.MIXITY 1]*maxload;
168
169         case('displacement control')
170             if TEST.MAXD > TEST.MIND;
171                 maxload = TEST.MAXD;
172             else
173                 maxload = TEST.MIND;
174             end
175             d = [-SIM.MIXITY 1]*maxload;
176             BC.DISP.NODE = [BC.DISP.NODE BC.FORCE.NODE];
177             BC.DISP.UX = [d0_x 0 0];
178             BC.DISP.UY = [d0_y d];
179             BC.FORCE.NODE = [];
180             BC.FORCE.FY = [];
181             BC.FORCE.FX = [];
182         end
183
184         % Activating nodes and sloving the system of equations
185         fprintf('(P) Solving system of equations...\n')
186         NODE.STATE = ones(SIM.NODES,1); % All
187         nodes active
188         NODE.STATE(BC.DISP.NODE) = 0; %
189         Dirichlet BC'oned nodes
190
191         SIM = fxfem_solver(ELEMENT,NODE,BC,
192             SIM);
193         fprintf('(*) Norm of the residual: %0.2e\n',norm(SIM.K*
194             SIM.d - SIM.f))
195
196         %% Extracting elemental and nodal displacement, stress
197         and strain values
198         fprintf('(P) Calculating stresses...\n')
199         IP = fxfem_StrainStressIP(SIM,IP,ELEMENT,MATERIAL);
200         [NODE] = fxfem_disp(SIM,NODE,ELEMENT,CRACK,MATERIAL);
201
202         %% Calculating fracture mechanics quantities
203         fprintf('(P) Solving LEFM...\n')
204         SIM.Jintr = 3;

```



```

199     [NODE,ERR,theta,J1,J2] = Jintegrals(SIM,ELEMENT,NODE,
200         CRACK,IP,'periphery',SIM.Jintr);
201
202     %% Crack propagation
203     C = MATERIAL(1).C;
204     m = MATERIAL(1).m;
205     switch SIM.stochastic
206     case(1)
207         dadN = stochastic() * C * ERR^m;    % [mm/cycle]
208     case(0)
209         dadN = C * ERR^m;    % [mm/cycle]
210     end
211
212     if max(ERR) < SIM.Gth
213         fprintf('(!) ERR below threshold value, (ERR = %0.4f
214             )\n',ERR)
215         dai = 0;
216     else
217         dai = SIM.da;
218     end
219
220     if max(ERR) >= MATERIAL(1).Gcri
221         fprintf('(!) ERR above critical value, (ERR = %0.4f)
222             .\n',ERR)
223         dai = SIM.a_end;
224     end
225
226     if dai/dadN < SIM.N_onset && SIM.a == SIM.a0
227         fprintf('(!) Below onset cycles.\n')
228         dai = 0;
229     else
230         SIM.a = SIM.a + dai;
231     end
232
233     dNi = dai / dadN;
234     SIM.N = SIM.N + dNi;
235
236     DATA.Damage = DATA.Damage + dNi/SIM.Fail;

```

```

234     DATA.D(nblocks)          = DATA.Damage;
235     DATA.Di(nblocks)         = dNi/SIM.Fail; % Palmgreen-Miner
        rule
236     DATA.Ni(nblocks)         = SIM.N;
237     DATA.dN(nblocks)         = dNi;
238     DATA.ai(nblocks)         = SIM.a;
239     DATA.da(nblocks)         = dai;
240     DATA.ID(nblocks)         = nblocks;
241     DATA.dadN(nblocks)       = dadN;
242     DATA.Gmax(nblocks)       = ERR;
243     DATA.angle(nblocks)      = theta;
244     DATA.Force(nblocks)      = abs(SIM.f(1));
245
246     switch SIM.CASE
247         case('load control')
248             DATA.MAXLOAD(nblocks) = TEST.MAXL;
249         case('displacement control')
250             DATA.MAXLOAD(nblocks) = TEST.MAXD;
251     end
252     DATA.R(nblocks) = R;
253
254     % Crack extension
255     if dai ~= 0
256         CRACK.X          = [CRACK.X ; SIM.da*cos(mean(theta))+
            CRACK.X(end)];
257         CRACK.Y          = [CRACK.Y ; SIM.da*sin(mean(theta))+
            CRACK.Y(end)];
258         CRACK.POINT = nblocks + 1;
259     end
260
261     if ERR >= MATERIAL(1).Gcri
262         fprintf('(!) System fractured, (ERR = %0.4f).\n',ERR
            )
263         break;
264     elseif ERR >= SIM.Fail
265         fprintf('(!) Maximum number of cycles reached.\n')
266         break;
267     elseif nblocks > 50

```

```

268         fprintf('(!) Maximum number of loading blocks
                reached.\n')
269         break;
270     end
271 end
272 SIM.TIME = toc(time);
273 fprintf('(M) Simulation ended with %i number of blocks.\n',
        nblocks)
274 fprintf('(M) Analysis runtime: %i minutes.\n',SIM.TIME/60);
275
276 %%
277 figure(1)
278 fplot_generalmesh(SIM.CONMAT,NODE.X,NODE.Y,0);
279 hold on
280 plot(CRACK.X,CRACK.Y,'.r-')
281 hold off
282 hold on
283 fplot_nodes(NODE,ELEMENT,SIM);
284 hold off
285
286
287 %%
288 figure(2)
289 X      = NODE.X + NODE.dx;
290 Y      = NODE.Y + NODE.dy;
291 NODE   = fxfem_StrainStressNODE(IP,ELEMENT,NODE,SIM,'SYY');
292 [~,h] = contourf(X(SIM.TOPOGRAPHY),...
293                 Y(SIM.TOPOGRAPHY),...
294                 NODE.SYY(SIM.TOPOGRAPHY),100);
295
296 hc = colorbar; xlabel(hc,'Stress in Y (MPa)');
297 set(h,'LineColor','flat');
298 axis([min(NODE.X)-10 max(NODE.X)+10 min(NODE.Y)-1 max(NODE.Y
        )+1]);
299 axis off
300
301 %%
302 figure(3)

```

```

303 NODE = fxfem_StrainStressNODE(IP,ELEMENT,NODE,SIM,'SVM');
304 VecPlot = NODE.SVM; VecPlot(NODE.TYPE == 1) = NaN;
305 [~,h] = contourf(X(SIM.TOPOGRAPHY),...
306                 Y(SIM.TOPOGRAPHY),...
307                 VecPlot(SIM.TOPOGRAPHY),50);
308 hc = colorbar; xlabel(hc,'Von Mises stress (MPa)');
309 set(h,'LineColor','flat');
310 axis([min(NODE.X)-10 max(NODE.X)+10 min(NODE.Y)-1 max(NODE.Y
    )+1]);
311 axis off
312
313 %%
314 figure(4)
315 [~,h] = contourf(X(SIM.TOPOGRAPHY),Y(SIM.TOPOGRAPHY),NODE.q(
    SIM.TOPOGRAPHY));
316 set(h,'LineColor','flat');
317 hold on
318 fplot_generalmesh(SIM.CONMAT,X,Y,0);
319 hold off
320 hc = colorbar; xlabel(hc,'Weight factor for J-integral
    evaluation (q)');
321 %axis([min(NODE.X)-10 max(NODE.X)+10 min(NODE.Y)-1 max(NODE.
    Y)+1]);
322 axis off

```

Q6 element definition

```

1 function [Ni,dNido,dNidp,N5,N6,dN5do,dN5dp,dN6do,dN6dp] =
    q6elem(o,p)
2 %q6elem: Four node incompatible quadrilateral element.
3 %
4 % INPUTS:
5 %   o: (n x 1) column vector with x-coordinates of IPs in
    REFERENCE space
6 %   p: (n x 1) column vector with y-coordinates of IPs in
    REFERENCE space
7 %
8 % OUTPUT:
9 %   Ni      : (n x 4) [N1 N2 N3 N4]
10 %   dNido   : (n x 4) [dN1do dN2do dN3do dN4do]
11 %   dNidp   : (n x 4) [dN1dp dN2dp dN3dp dN4dp]
12 %   N5      : (n x 1)
13 %   N6      : (n x 1)
14 %   dN5do   : (n x 1)
15 %   dN5dp   : (n x 1)
16 %   dN6do   : (n x 1)
17 %   dN6dp   : (n x 1)
18
19 % Check inputs
20 if isvector(o) == 0 || isvector(p) == 0
21     error('Inputs are not vectors');
22 else
23     if size(o,1) == 1; o = o'; end
24     if size(p,1) == 1; p = p'; end
25 end
26
27 if length(o) ~= length(p)
28     error('Inputs have unequal length')
29 end
30
31 % Element shape functions
32 Ni = 0.25*[(1-o).*(1-p) (1+o).*(1-p) (1+o).*(1+p) (1-o).*(1+
    p)];

```

```
33 N5 = (1-o.^2);
34 N6 = (1-p.^2);
35
36 % Element shape function derivatives
37 dNido = 0.25*[-(1-p) (1-p) (1+p) -(1+p)];
38 dNidp = 0.25*[-(1-o) -(1+o) (1+o) (1-o)];
39 dN5do = -2*o;
40 dN5dp = 0*o;
41 dN6do = 0*p;
42 dN6dp = -2*p;
43 end
```

J integral calculation

```

1 function [NODE,G,theta,J1,J2] = Jintegrals(SIM,ELEMENT,NODE,
    CRACK,IP,J_area,r)
2 % Function computes J integrals 1 and 2 based on EDI and
    Energy release
3 % rate criterion for angle of propagation
4
5
6 switch J_area
7     case('radii')
8         q = (NODE.X - CRACK.X(end)).^2 + (NODE.Y - CRACK.Y(
            end)).^2 < r^2;
9     case('periphery')
10        q = double(NODE.TYPE==2);
11        for n2 = 1 : r
12            q_test = zeros(SIM.NODES,1);
13            for n1 = 1 : SIM.ELEMENTS
14                if sum(q(ELEMENT(n1).NODES)) > 0
15                    q_test(ELEMENT(n1).NODES) = [1 1 1 1];
16                end
17            end
18            q = q_test;
19        end
20    otherwise
21        error('Undefined J integral criteria')
22 end
23
24 NODE.q = q;
25 J1      = 0;
26 J2      = 0;
27 for n1 = 1 : SIM.ELEMENTS
28
29     qi    = q(ELEMENT(n1).NODES);
30     dqdX  = IP(n1).dNidX * qi;
31     dqdY  = IP(n1).dNidY * qi;
32

```

```

33     [Sxx,Syy,Sxy,Exx,Eyy,Exy,dqdx,dqdy] =
        fsub_localstressesstrain(CRACK,IP,n1,dqdX,dqdY);
34
35     U = sum((Sxx.*Exx + Syy.*Eyy + Sxy.*Exy*2).* IP(n1).W);
36
37     Q1 = U.*dqdx;
38     Q2 = (Sxx.*Exx + Sxy.*Exy).*dqdx + (Sxy.*Exx + Syy.*Exy)
        .*dqdy;
39     J1 = sum((Q1 - Q2) .* IP(n1).W .* IP(n1).detJ) + J1;
40
41     Q3 = U.*dqdy;
42     Q4 = (Sxx.*Exy + Sxy.*Eyy).*dqdx + (Sxy.*Exy + Syy.*Eyy)
        .*dqdy;
43     J2 = sum((Q3 - Q4) .* IP(n1).W .* IP(n1).detJ) + J2;
44 end
45
46 if abs(J1) < 1E-6; J1 = 0; elseif abs(J2) < 1E-6; J2 = 0;
    end
47
48 theta = atan2(abs(J2),abs(J1));
49 G      = abs(J1*cos(theta) + J2*sin(theta));
50
51
52 if abs(theta) < 1E-6; theta = 0; end
53 end
54
55
56 function [Sxx,Syy,Sxy,Exx,Eyy,Exy,dqdx,dqdy] =
        fsub_localstressesstrain(CRACK,IP,eval,dqdX,dqdY)
57 % Transformation of stresses and strain from global
        coordinate system to
58 % local crack tip coordinate system:
59 %
60 % y                                Y (global)
61 % ^                                ^
62 % | /                              |
63 % | /                              |
64 % + -----> x (crack)           + - - -> X

```



```

65
66
67 % Crack segment vector
68 Cv = [CRACK.X(end) - CRACK.X(end-1) , CRACK.Y(end) - CRACK.Y
        (end-1) , 0];
69 % Global X unit vector
70 Xv = [1 0 0];
71 % Angle between X axis and crack segment
72 CROSS = cross(Xv,Cv);
73 a      = atan2(CROSS(3),dot(Xv,Cv));
74 % Rotation matrix
75 R      = [cos(a) -sin(a);sin(a) cos(a)];
76
77 n_ips = length(IP(eval).EXX);
78 Exx = zeros(n_ips,1);
79 Eyy = zeros(n_ips,1);
80 Exy = zeros(n_ips,1);
81 for n1 = 1 : n_ips % Loop over integration points
82     EE = R'*[IP(eval).EXX(n1) IP(eval).EXY(n1);...
83             IP(eval).EXY(n1) IP(eval).EYY(n1)]*R;
84     Exx(n1) = EE(1,1);
85     Exy(n1) = EE(1,2);
86     Eyy(n1) = EE(2,2);
87 end
88
89 SXX = IP(eval).SXX;
90 SYX = IP(eval).SYX;
91 SXY = IP(eval).SXY;
92
93 Sxx = (SXX+SYX)/2 + (SXX-SYX)/2*cos(2*a) + SXY*sin(2*a);
94 Syy = (SXX+SYX)/2 - (SXX-SYX)/2*cos(2*a) - SXY*sin(2*a);
95 Sxy = SXY*cos(2*a) - (SXX-SYX)/2*sin(2*a);
96
97 dqdx = dqdX*cos(a) + dqdY*sin(a);
98 dqdy = -dqdX*sin(a) + dqdY*cos(a);
99 end

```

Code for element split

```

1 function [X,Y,conmat] = rgrid(x,y,div)
2 %rgrid(x,y,div) creates a rectangular grid in x, y based on
   divisions in x
3 xvec      = min(x):(max(x)-min(x))/div:max(x);
4 yvec      = min(y):(max(y)-min(y))/div:max(y);
5 [xmat,ymat] = meshgrid(xvec,yvec);
6 X          = reshape(xmat',[],1);
7 Y          = reshape(ymat',[],1);
8 ne         = div^2;
9 nn         = (div+1)^2;
10
11 conmat = zeros(ne,4);
12 c1      = -(div+1);
13 c2      = 0;
14 c3      = 0;
15 for n1 = 1 : div
16     c1 = c1 + (div+1);
17     c2 = c2 + (div+1);
18     for n2 = 1 : div
19         c3      = c3 + 1;
20         conmat(c3,1) = n2 + c1;
21         conmat(c3,2) = n2 + c1 + 1;
22         conmat(c3,3) = n2 + c2 + 1;
23         conmat(c3,4) = n2 + c2;
24     end
25 end
26 end

```

Inverse mapping of quadrilateral elements

```

1 function [xi,eta] = q4invmap(x,y,xq,yq)
2 %q4invmap: Perform inver mapping of integration points
3 %
4 % INPUTS:
5 %   x:  (4 x 1) column vector with x-coordinates
      quadrilateral nodes
6 %   y:  (4 x 1) column vector with y-coordinates
      quadrilateral nodes
7 %   xq: (n x 1) column vector with x-coordinates of IPs in
      REAL space
8 %   yq: (n x 1) column vector with y-coordinates of IPs in
      REAL space
9 %
10 % OUTPUT:
11 %   xi:  (n x 1) column vector with x-coordinates of IPs in
      REFERENCE space
12 %   eta: (n x 1) column vector with y-coordinates of IPs in
      REFERENCE space
13
14
15 % Check inputs
16 if isvector(x) == 0 || isvector(y) == 0 || isvector(xq) == 0
      || isvector(yq) == 0
17     error('Inputs are not vectors');
18 else
19     if size(x,1) == 1; x = x'; end
20     if size(y,1) == 1; y = y'; end
21 end
22
23 if length(x) ~= length(y) || length(xq) ~= length(yq)
24     error('x and y inputs have unequal length')
25 end
26
27 % Inverse map (according to Chongyu Hua "An inverse
      transformation for

```

```

28 % quadrilateralisoparametric elements: Analysis and
    application" Finite
29 % Elements in Analysis and Design, volume 7 (2) 1990 pp.
    159-166
30 % http://www.sciencedirect.com/science/article/pii/0168874
    X90900072?via%3Dihub
31 xi = zeros(length(xq),1);
32 eta = zeros(length(xq),1);
33 for i = 1 : length(xq) % loop over query points
34     d1 = 4*xq(i) - sum(x);
35     d2 = 4*yq(i) - sum(y);
36
37     XY = [x y];
38     I = [1 -1 1 -1;-1 1 1 -1;-1 -1 1 1];
39     ABC = I*XY;
40     a1 = ABC(1,1);
41     a2 = ABC(1,2);
42     b1 = ABC(2,1);
43     b2 = ABC(2,2);
44     c1 = ABC(3,1);
45     c2 = ABC(3,2);
46     ab = a1*b2 - a2*b1;
47     ac = a1*c2 - a2*c1;
48
49     if a1*a2*ab*ac ~= 0 || (a1 == 0 && a2*c1 ~= 0) || (a2 ==
        0 && a1*b2 ~= 0)
50         ad = a1*d2 - a2*d1;
51         ba = b1*a2 - b2*a1;
52         cb = c1*b2 - c2*b1;
53         da = d1*a2 - d2*a1;
54         dc = d1*c2 - d2*c1;
55         a = ab;
56         b = (cb+da);
57         c = dc;
58         xi1 = (-b+sqrt(b^2-4*a*c))/(2*a);
59         xi2 = (-b-sqrt(b^2-4*a*c))/(2*a);
60         if xi1 >= -1 && xi1 <= 1
61             xi(i) = xi1;

```

```
62         else
63             xi(i) = xi2;
64         end
65         eta(i) = (ad + ba*xi(i))/ac;
66
67     elseif a1*a2 ~= 0 && ab == 0
68         dc      = d1*c2 - d2*c1;
69         ad      = a1*d2 - a2*d1;
70         xi(i)   = a1*dc/(b1*ac + a1*ad);
71         eta(i)  = ad/ac;
72
73     elseif a1*a2 ~= 0 && ac == 0
74         ad      = a1*d2 - a2*d1;
75         db      = d1*b2 - d2*b1;
76         xi(i)   = ad/ab;
77         eta(i)  = a1*db/(c1*ab + a1*ad);
78
79     else
80         dc      = d1*c2 - d2*c1;
81         bc      = b1*c2 - b2*c1;
82         bd      = b1*d2 - b2*d1;
83         xi(i)   = dc/(a1*d2 + bc);
84         eta(i)  = bd/(a2*d1 + bc);
85     end
86 end
```

Legendre-Gauss quadrature in 2D

```
1 function [X,Y,WW] = gq2d()
2
3 % 1D integration point coordinates
4 x = [-0.577350269189626;0.577350269189626];
5 w = [1;1];
6 % Converting 1D coordinates into 2D space
7 X = repmat(x,2,1);
8 Y = reshape(repmat(x',2,1),[],1);
9 Wx = repmat(w,2,1);
10 Wy = reshape(repmat(w',2,1),[],1);
11 WW = Wx .* Wy;
12 end
```

Algorithm for derivative conversion

```

1 function [dfdo,dfdp] = ffem_dervconvert(mat,dfdX,dfdY)
2 %ffem_dervconvert maps derivatives functions to REFERENCE or
   REAL domain.
3 %   ffem_dervconvert(matJ,dfdX,dfdY,ni,nn) convert the
   derivatives of the
4 %   inputed functions to the REFERENCE or REAL domain. If
   the Jacobian
5 %   matrix is provided, the derivatives inputed must be with
   respect to the
6 %   REAL domain. If the inverse of the Jacobain is inputed,
   the inputed
7 %   derivatives must be with respect to the REFERENCE domain
   .
8 %       J       : (4,4,ni) Jacobian matrix or Jacobian inverse
9 %       dfdX    : (ni,4,1) function x-derivative in REAL/
   REFERENCE domain
10 %       dfdY    : (ni,4,1) function y-derivative in REAL/
   REFERENCE domain
11 % Note:
12 %       l       : number of functions
13 %       ni      : number of integration points
14 %       nn      : scalar, number of nodes
15 %
16
17 ni = size(dfdX,1);
18 nn = size(dfdX,2);
19 dfdo = zeros(size(dfdX));
20 dfdp = zeros(size(dfdY));
21 for n1 = 1 : size(dfdX,3) % Loop over functions
22     for n2 = 1 : ni % Loop over integration points
23         MAT = mat(:, :, n2);
24         for n3 = 1 : nn % Loop over nodes
25             VEC = [dfdX(n2,n3,n1); dfdY(n2,n3,n1)
26                 ];
27             LHS = MAT*VEC;
28             dfdo(n2,n3,n1) = LHS(1);

```

```
28         dfdp(n2,n3,n1) = LHS(2);
29     end
30 end
31 end
32 end
```


Jacobian

```

1  function [ varargout ] = ffem_jacobian( X,Y,dNido,dNidp,
    request )
2  %ffem_jacobian Computes the Jacobian values in 2D space
3  %   ffem_jacobian( X,Y,dNido,dNidp,request ) is capable of
    computing the
4  %   Jacobian matrix, its inverse and the determinant of both
    matrises.
5  %
6  % INPUTS:
7  %   X           : (n x 1) vector with x-coordinates of the
    element nodes
8  %   Y           : (n x 1) vector with y-coordinates of the
    element nodes
9  %   dNido       : (n x q) matrix of shape function derivatives
    with respect to
10 %               the abscissa coordinate (xi) coordinate of an
    integration
11 %               point in the REFERENCE space
12 %   dNidp       : (n x q) matrix of shape function derivatives
    with respect to
13 %               the ordinate coordinate (eta) coordinate of an
    integration
14 %               point in the REFERENCE space
15 %   request : argument to determine the output of the
    function
16 %               (1) = Jacobian matrix.
17 %               (2) = Jacobian matrix determinant.
18 %               (3) = Jacobian matrix and its determinant.
19 %               (4) = Jacobian inverse matrix.
20 %               (5) = Jacobian inverse matrix determinant.
21 %               (6) = Jacobian inverse matrix and its
    determinant.
22 %               (7) = Both Jacobian and inverse matrix and
    their
23 %               determinants.
24 %

```

```

25 % OUTPUT:
26 %   Variable output function; see 'request' input.
27
28
29
30 % PROCESS: Input check
31 if request < 1 || request > 7 || ceil(request) ~= floor(
    request)
32     error('ffem_jacobian : unsupported ''request'' value')
33 end
34
35 test(1) = sum(size(X) ~= size(Y));
36 test(2) = isvector(X);
37 test(3) = isvector(Y);
38
39 if sum(test) > 2
40     error('Error in input coordinates.')
41 end
42
43 if size(X,1) == 1
44     X = X';
45 end
46 if size(Y,1) == 1
47     Y = Y';
48 end
49
50 no_ip = size( dNido,1 );
51 if request == 1
52     J = fsub_matrix( X,Y,dNido,dNidp,no_ip );
53
54 elseif request == 2 || request == 3
55     J = fsub_matrix( X,Y,dNido,dNidp,no_ip );
56     detJ = fsub_determinant(J, no_ip);
57
58 elseif request == 4
59     J = fsub_matrix( X,Y,dNido,dNidp,no_ip );
60     detJ = fsub_determinant(J, no_ip);
61     invJ = fsub_inverse( J, detJ, no_ip );

```

```
62
63 elseif request >= 5
64     J      = fsub_matrix( X,Y,dNido,dNidp,no_ip );
65     detJ    = fsub_determinant(J, no_ip);
66     invJ    = fsub_inverse( J, detJ, no_ip );
67     detinvJ = fsub_determinant(invJ, no_ip);
68
69 end
70
71
72
73 switch request
74     case(1) % Returns only the Jacobian matrix
75         varargout{1} = J;
76
77     case(2) % Returns only the Jacobian determinant
78         varargout{1} = detJ;
79
80     case(3) % Returns the Jacobian matrix and its
81             determinant
82         varargout{1} = J;
83         varargout{2} = detJ;
84
85     case(4) % Returns only the Jacobian inverse
86         varargout{1} = invJ;
87
88     case(5) % Returns the Jacobian inverse matrix
89             determinant
90         varargout{1} = detinvJ;
91
92     case(6) % Returns both the Jacobian inverse and its
93             determinant
94         varargout{1} = invJ;
95         varargout{2} = detinvJ;
96
97     case(7) % Returns all values
98         varargout{1} = J;
99         varargout{2} = detJ;
```

```

97         varargout{3} = invJ;
98         varargout{4} = detinvJ;
99     end
100 end
101
102
103
104 function [ J ] = fsub_matrix( X,Y,dNido,dNidp,no_ip )
105
106 J = zeros( 2,2,no_ip);
107 for n1 = 1 : no_ip
108
109     % STEP: Calculate partial derivatives
110     J11 = dNido(n1,:) * X; % d(x)/d(xi)
111     J12 = dNido(n1,:) * Y; % d(x)/d(eta)
112     J21 = dNidp(n1,:) * X; % d(y)/d(xi)
113     J22 = dNidp(n1,:) * Y; % d(y)/d(eta)
114
115     % STEP: Generating Jacobian matrix
116     J(:, :, n1) = [J11 J12
117                   J21 J22];
118 end
119 end
120
121
122
123 function [ invJ ] = fsub_inverse( J, detJ, no_ip )
124
125 invJ = zeros( 2, 2, no_ip );
126 for n1 = 1 : no_ip
127     % PROCESS: Caculating Jacobian cofactors
128     cof_11 = J(2,2,n1);
129     cof_12 = -J(1,2,n1);
130     cof_21 = -J(2,1,n1);
131     cof_22 = J(1,1,n1);
132
133     % PROCESS: Caculating the Jacobian Adjoint matrix
134     AdjJ = [cof_11 cof_12

```

```
135         cof_21 cof_22];
136
137 % PROCESS: Calculating the Jacobian inverse matrix
138     invJ(:,:,n1) = AdjJ / detJ(n1);
139 end
140 end
141
142
143
144 function [ det ] = fsub_determinant( Mat, no_ip )
145
146 det = zeros( no_ip, 1);
147 for n1 = 1 : no_ip
148     Mat11      = Mat(1,1,n1);
149     Mat12      = Mat(1,2,n1);
150     Mat21      = Mat(2,1,n1);
151     Mat22      = Mat(2,2,n1);
152     det(n1) = Mat11 * Mat22 - Mat12 * Mat21;
153 end
154 end
```

Polar mapping of integration points

```

1 function [ radii,angle,alpha ] = fgeo_polarmap( xq,yq,xv,yv
    )
2 %fgeo_polarmap Computes the polar coordinate.
3 %   fgeo_polarmap( xq,yq,xv,yv ) computes the polar
    coordinates of a
4 %   given set of query points (xq,yq) for a polar radii
    direction vector in
5 %   (xv,yv) where xv(end) and yv(end) describe the polar
    space origin.
6 %
7 % INPUTS:
8 %   xq : (q x 1) vector of query point x-coordinates
9 %   yq : (q x 1) vector of query point y-coordinates
10 %   xv : (2 x 1) vector with x-coordinates of radial
    dimension vector of
11 %           the polar space
12 %   yv : (2 x 1) vector with y-coordinates of radial
    dimension vector of
13 %           the polar space
14 %
15 % OUTPUT:
16 %   angle : (q x 1) vector with angle coordinates of the
    query points
17 %   radii : (q x 1) vector with radii coordinates of the
    query points
18 %   alpha : (1) scalar with the oriented angle of rotation
    between the
19 %           radial dimension vector and a horizontal unit
    vector.
20
21 % STEP: Declaring radii function
22 r = @(x,y,xt,yt) sqrt((x-xt).^2+(y-yt).^2);
23
24 % STEP: Polar coordinate vector
25 cv = [xv(end) - xv(end-1)
26       yv(end) - yv(end-1)];

```

```

27
28 % STEP: Calculate angle between a regular Cartesian
    coordinate system and
29 %       vector of radial dimension in the polar space
30 av = [ 1 ; 0 ]; % x-coordinate unit vector
31 dot_product = cv' * av;
32 norm_product = norm( cv ) * norm( av );
33 cross_product = det( [cv av]' ); % Simplified to yield the z
    component
34 orientation = sign( cross_product );
35 alpha = acos( dot_product/norm_product ) *
    orientation;
36
37
38
39 % STEP: Declaring polar space origin
40 xt = xv(end);
41 yt = yv(end);
42
43 % STEP: Computing polar coordinates
44 radii = r( xq,yq,xt,yt );
45
46 angle = wrapTo2Pi( atan2( yq-yt,xq-xt ) + alpha - pi );
47 %angle = atan2( yq-yt,xq-xt ) + alpha;
48
49 %angle(angle<0) = angle(angle<0) + 2*pi;
50 %angle = wrapTo2Pi( atan2( yq-yt,xq-xt ) + alpha - pi ) + pi
    ; % Use this
51 %angle = atan2( yq-yt,xq-xt ) + alpha;
52 end

```

Level set definition

```

1 function [ Dn,Dt ] = fgeo_signed( xq,yq,xc,yc )
2 %fxfem_signed: Calculates the signed distance between a
   point and a line.
3 %   fgeo_signed(xq,yq,xc,yc) defines the signed distance
   function between
4 %   a query point and a line. Is based on a projection of
   the point to the
5 %   normal vector of the line defined 90 degrees
   counterclockwise.
6 %
7 % INPUTS:
8 %   xq : (n x 1) vector of x-coordinates of the query points
9 %   yq : (n x 1) vector of y-coordinates of the query points
10 %   xc : (2 x 1) vector of x-coordinates of the segment end
   points
11 %   yc : (2 x 1) vector of y-coordinates of the segment end
   points
12 %
13 % OUTPUT:
14 %   Dn : (n x 1) vector of the normal distances between the
   query points
15 %       and the segment
16 %   Dt : (n x 1) vector of the tangential distances between
   the query
17 %       points and the segment
18 %
19
20 % LOCAL NOTES:
21 %   Xo : Query points
22 %   X  : Segment edges
23 %   xn : Number of query points
24 %   Dn : Normal signed distance
25 %   Dt : Tangential signed distance
26 %   Nn : Unit normal vector
27 %   Nt : Unit tangential vector
28 %   Px : Projection points

```

```

29 %   Vx : Projection vectors
30
31 % PROCESS: Storing values in vectors
32 n_points = length(xq);
33 Xo       = [ xq yq ];
34 X        = [ xc(1) yc(1) ; xc(2) yc(2) ];
35
36 % PROCESS: Computing normal and tangential crack vectors
37 R  = X(2,:) - X(1,:);
38 Rn = [-R(2) R(1)];           % Normal vector
39 Nn = Rn/norm(Rn);           % Unit normal vector
40 Nt = R/norm(R);             % Unit tangent vector
41
42 % PROCESS: Computing normal and tangential distances
43 Dn = zeros(n_points,1);
44 Dt = zeros(n_points,1);
45 Px = zeros(n_points,2,2);
46 Vx = zeros(n_points,2,2);
47 for n1 = 1 : n_points
48     Rx = Xo(n1,:) - X(end,:);
49     Dn(n1) = Rx*Nn';         % Normal distance
50     Dt(n1) = Rx*Nt';         % Tangential distance
51     Vx(n1,:,1) = Dn(n1)*Nn;
52     Vx(n1,:,2) = Dt(n1)*Nt;
53     Px(n1,:,1) = Xo(n1,:) - Vx(n1,:,1);
54     Px(n1,:,2) = Xo(n1,:) - Vx(n1,:,2);
55 end
56 end

```

Script for plotting node enrichments

```

1 function [h0,h1,h2,h3] = fplot_nodes(NODE,ELEMENT,SIM)
2 %fplot_nodes: Plots the domain nodes.
3 %   The function classify and store the node coordinates for
4 %   plotting
5 %   purposes.
6 % INPUTS:
7 %   NODE : Node structure
8 % OUTPUT:
9 %   h0 : Plot handle for Standard nodes
10 %   h1 : Plot handle for Heavyside nodes
11 %   h2 : Plot handle for Near-Tip nodes
12 % NOTES:
13 %   - Node flag standard:
14 %       * (0) = Standard node
15 %       * (1) = Heavyside node
16 %       * (2) = Near-tip node
17 %       * (3) = Bimaterial node
18
19 % Plot nodes
20 blend_n = unique(SIM.CONMAT(sum(NODE.TYPE(SIM.CONMAT)==2,2)
21 <4 & sum(NODE.TYPE(SIM.CONMAT)==2,2)>0,:));
22 hold on
23 %h0 = plot(NODE.X(NODE.TYPE==0),NODE.Y(NODE.TYPE==0),'.k',
24 %   'LineWidth',1);
25 h1 = plot(NODE.X(NODE.TYPE==1),NODE.Y(NODE.TYPE==1),'ob',
26 %   'LineWidth',1,'MarkerFaceColor','b','MarkerSize',8);
27 h2 = plot(NODE.X(NODE.TYPE==2),NODE.Y(NODE.TYPE==2),'sb',
28 %   'LineWidth',1,'MarkerFaceColor','b','MarkerSize',8);
29 h3 = plot(NODE.X(blend_n),NODE.Y(blend_n),'xr','LineWidth'
30 %   ,1,'MarkerFaceColor','none','MarkerSize',8);
31 hold off
32 end

```

Script for plotting a rectangular mesh

```
1 function [ h ] = fplot_generalmesh( ConMat,X,Y,shadow )
2 %fplot_mesh: Plot the given mesh
3
4 % Creating coordinate matrix
5 Mesh = [X Y];
6
7 % Plotting the mesh grid
8 h = patch('Faces',ConMat,'Vertices',Mesh);
9 if shadow == 1
10     set(h,'FaceColor',[0.9 0.9 0.9])
11 else
12     set(h,'FaceColor','None','LineWidth',0.1,'EdgeColor',[1
13         1 1]*0.2,'EdgeAlpha',0.2)
14
15 end
```

Script to calculate stress and strain at integration points

```

1  function [IP]=fxfem_StrainStressIP(SIM,IP,ELEMENT,MATERIAL)
2  %fxfem_StrainStressIP calculates the stress and strain at
   the integration
3  %point.
4
5
6  for n1 = 1 : SIM.ELEMENTS % Loop over elements
7      n_ip = size(IP(n1).Ni,1);
8      exx = zeros(n_ip,1);
9      eyy = zeros(n_ip,1);
10     ezz = zeros(n_ip,1);
11     exy = zeros(n_ip,1);
12     sxx = zeros(n_ip,1);
13     syy = zeros(n_ip,1);
14     szz = zeros(n_ip,1);
15     sxy = zeros(n_ip,1);
16     svm = zeros(n_ip,1);
17     D    = MATERIAL(ELEMENT(n1).MATERIAL).D;
18
19     for n2 = 1 : n_ip % Loop over integration points
20
21         % DoFs vectors
22         u = SIM.d(ELEMENT(n1).DOF.U);
23         a = SIM.d(ELEMENT(n1).DOF.A);
24         b = SIM.d(ELEMENT(n1).DOF.B);
25
26         % Strain/displacement matrices
27         Bu = fxfem_Bmats(IP(n1).dNidX(n2,:),IP(n1).dNidY(
           n2,:), 'standard');
28         Ba = fxfem_Bmats(IP(n1).dM1dX(n2,:),IP(n1).dM1dY(
           n2,:), 'heaviside');
29         Bb = fxfem_Bmats(IP(n1).dM2dX(n2,:,:),IP(n1).dM2dY(
           n2,:,:), 'neartip');
30
31         % Strain and stress calculation

```

```

32     e  = Bu*u + Ba*a + Bb*b;
33     s  = D*e;
34
35     % Plane strain/plane stress
36     exx(n2) = e(1);
37     eyy(n2) = e(2);
38     exy(n2) = e(3);
39     sxx(n2) = s(1);
40     syy(n2) = s(2);
41     sxy(n2) = s(3);
42
43     switch MATERIAL(ELEMENT(n1).MATERIAL).TYPE
44     case('isotropic')
45         E = MATERIAL(ELEMENT(n1).MATERIAL).E;
46         v = MATERIAL(ELEMENT(n1).MATERIAL).V;
47         switch SIM.LOADING
48         case('plane stress')
49             szz(n2) = 0;
50             ezz(n2) = -v/E*(sxx(n2) + syy(n2));
51         case('plane strain')
52             szz(n2) = v*(sxx(n2) + syy(n2));
53             ezz(n2) = 0;
54         end
55     case('orthotropic')
56         Exx = MATERIAL(ELEMENT(n1).MATERIAL).E1;
57         Eyy = MATERIAL(ELEMENT(n1).MATERIAL).E2;
58         Ezz = MATERIAL(ELEMENT(n1).MATERIAL).E3;
59         vzx = MATERIAL(ELEMENT(n1).MATERIAL).V31;
60         vzy = MATERIAL(ELEMENT(n1).MATERIAL).V32;
61         switch SIM.LOADING
62         case('plane stress')
63             szz(n2) = 0;
64             ezz(n2) = -(vzx*sxx(n2)/Exx + vzy*
65                 syy(n2)/Eyy) ;
66         case('plane strain')
67             szz(n2) = Ezz*(vzx*sxx(n2)/Exx + vzy
68                 *syy(n2)/Eyy);
69             ezz(n2) = 0;

```

```
68             end
69         end
70
71         svm(n2) = sqrt(0.5*((sxx(n2)-syy(n2))^2+(syy(n2)-szz
72             (n2))^2+(szz(n2)-sxx(n2))^2)+3*sxy(n2)^2);
73     end
74     IP(n1).EXX = exx;
75     IP(n1).EYY = eyy;
76     IP(n1).EZZ = ezz;
77     IP(n1).EXY = exy;
78     IP(n1).SXX = sxx;
79     IP(n1).SYY = syy;
80     IP(n1).SZZ = szz;
81     IP(n1).SXY = sxy;
82     IP(n1).SVM = svm;
83     clear exx eyy ezz exy sxx syy szz sxy svm
84 end
85 end
```

Script for meshing

```
1 function [xn,yn,conmat] = mesher(x,y)
2 % Function creates conectivity matrix and nodal vectors
3
4
5 x = unique(x);
6 y = unique(y);
7 xn = repmat(x',length(y),1);
8 yn = reshape(repmat(y,length(x),1),[],1);
9
10 % Connectivity matrix
11 nn_x = length(x);
12 nn_y = length(y);
13 ne_x = nn_x - 1;
14 ne_y = nn_y - 1;
15 c1 = -nn_x;
16 c2 = 0;
17 i = 0;
18 nn = nn_x*nn_y; % number of nodes
19 ne = ne_x*ne_y; % number of elements
20 conmat = zeros(ne,4);
21 for n1 = 1 : ne_y
22     c1 = c1 + nn_x;
23     c2 = c2 + nn_x;
24     for n2 = 1 : ne_x
25         i = i + 1;
26         a = n2 + c1;
27         b = n2 + c1 + 1;
28         c = n2 + c2 + 1;
29         d = n2 + c2;
30         conmat(i,:) = [a b c d];
31     end
32 end
```

Script for enrichment assignment of nodes and elements

```

1  function [NODE,ELEMENT] = fxfem_enrich(SIM,NODE,ELEMENT,
    CRACK,varargin)
2  %fxfem_enrich      Node/Element enrichment assignment
3  %    fxfem_enrich(SIM,NODE,ELEMENT,CRACK) assigns the
4  %    element and node ID's as defined in SIM for XFEM
    processing.
5  %
6  % INPUTS:
7  %    SIM.ELEMENTS(i)
8  %    NODE.ID(i)
9  %    NODE.TYPE(i)
10 %    NODE.X(i)
11 %    NODE.Y(i)
12 %    NODE.PSI(i)
13 %    NODE.PHI(i)
14 %    ELEMENT(i).NODES(i)
15 %    ELEMENT(i).TYPE
16 %    CRACK.X(i)
17 %    CRACK.Y(i)
18 %
19 % OUTPUT:
20 %    NODE.TYPE(i)
21 %    ELEMENT(i).TYPE
22 %
23 % NOTES:
24 %    - Function limited to 4 node quadrilateral elements.
25
26 if strcmp(varargin{1},'radius') == 0 && strcmp(varargin{1},'
    periphery') == 0
27     error('Undefined condition.')
28 end
29 if strcmp(varargin{1},'radius') == 1 && length(varargin) ==
    1
30     error('Radius factor not specified.')
```



```

31 elseif strcmp(varargin{1},'radius') == 1 && length(varargin)
    == 2
32     rf = varargin{2};
33 else
34     rf = 1;
35 end
36
37 for n1 = 1 : SIM.ELEMENTS
38
39     x_e      = NODE.X( ELEMENT(n1).NODES );
40     y_e      = NODE.Y( ELEMENT(n1).NODES );
41     psi_e     = NODE.PSI( ELEMENT(n1).NODES );
                                     % normal level set
42     phi_e     = NODE.PHI( ELEMENT(n1).NODES );
                                     % tangential level set
43     x_vec     = [x_e(2:end) ; x_e(1)];
44     y_vec     = [y_e(2:end) ; y_e(1)];
45
46     % Test for crack influenced elements
47     if min(psi_e)*max(psi_e) <= 0 && max(phi_e) < 0
                                     % definite crack splited
                                     elements
48         NODE.TYPE(ELEMENT(n1).NODES) = 1;
49     elseif min(psi_e)*max(psi_e) <= 0 && min(phi_e)*max(
        phi_e) <= 0                 % possible crack tip elements
50         IN = inpolygon(CRACK.X(end),CRACK.Y(end),x_vec,y_vec
            );
51         if min(IN)*max(IN) == 1
                                     % crack
                                     tip inside element
52             NODE.TYPE(ELEMENT(n1).NODES) = 2;
53             area = polyarea([x_e ; x_e(1)],[y_e ; y_e(1)]);
54             r     = rf*sqrt(area); % Characteristic length
55         elseif isempty(polyxpoly(CRACK.X,CRACK.Y,x_vec,y_vec
            )) ~ = 0                 % crack passes trough element
56             NODE.TYPE(ELEMENT(n1).NODES) = 1;
57         end
58     end

```

```

59 end
60
61 switch varargin{1}
62     case('radius')
63         % *** Find nodes within characteristic radius
64         within = (NODE.X - CRACK.X(end)).^2 + (NODE.Y - CRACK.Y(
65             end)).^2 < r^2;
66         if sum(within) ~= 0
67             NODE.TYPE( within ) = 2;
68         end
69         for n1 = 1 : SIM.ELEMENTS
70             if sum(ismember(ELEMENT(n1).NODES,NODE.ID(within)))
71                 > 0
72                 NODE.TYPE( ELEMENT(n1).NODES ) = 2;
73             end
74         end
75         % ***
76         case('periphery')
77             for n1 = 1 : varargin{2} % loop over peripheries
78                 target_nodes = NODE.ID(NODE.TYPE == 2);
79                 for n2 = 1 : SIM.ELEMENTS
80                     test_nodes = ELEMENT(n2).NODES;
81                     if sum(ismember(test_nodes,target_nodes)) >
82                         0
83                         for n3 = 1 : 4
84                             if NODE.TYPE(ELEMENT(n2).NODES(n3))
85                                 ~= 1
86                                 NODE.TYPE(ELEMENT(n2).NODES(n3))
87                                     = 2;
88                             end
89                         end
90                     end
91                 end
92             end
93         case('none')
94         otherwise
95             error('Undefined definition for ''type''')
96     end

```

```
92
93 % Identifiying elements. The following convention for
    element
94 % identification is used:
95 % Standard element = 0
96 % Heaviside element = 1
97 % Near-tip element = 2
98 % Blending element = 3
99 for n1 = 1 : SIM.ELEMENTS
100     n_types = NODE.TYPE(ELEMENT(n1).NODES);
101     if sum(ismember(n_types,0)) == 4 % Standard
        element
102         ELEMENT(n1).TYPE = 0;
103     elseif sum(ismember(n_types,1)) == 4 % Heaviside
        element
104         ELEMENT(n1).TYPE = 1;
105     elseif sum(ismember(n_types,2)) == 4 % Near-tip
        element
106         ELEMENT(n1).TYPE = 2;
107     elseif sum(ismember(n_types,2)) >= 1 || sum(ismember(
        n_types,2)) < 4 % Blending element (on near-tips)
108         ELEMENT(n1).TYPE = 3;
109     end
110 end
111 end
```

Script for element degrees of freedom assignment

```

1  function [ dof,dof_x,dof_y ] = fxfem_dofs( nodes,no_nodes ,
      request )
2  % fxfem_dofs Computes the degrees of freedom for the given
      node indexes
3  %   fxfem_dofs(nodes,no_nodes,request) handles the Degrees
      of Freedom
4  %   indexes given the node indexes given in "nodes" row
      vector.
5  %
6  % INPUTS:
7  %   nodes      : (1 x n) vector with nodal indexes
8  %   no_nodes   : Total number of nodes in the simulation
9  %   request    : Character input to request type of degree of
      freedom
10 %           > 'standard'
11 %           > 'heaviside'
12 %           > 'neartip'
13 %
14 % OUTPUT:
15 %   dof       : (1 x q) vector with the degrees of freedom in
      the element
16 %   dof_x    : (1 x p) vector with x-coordinate degrees of
      freedom requested
17 %   dof_y    : (1 x p) vector with y-coordinate degrees of
      freedom requested
18
19
20
21 % Test inputs
22 if isvector( nodes ) == 0
23     error(' fxfem_dofs: input in "nodes" is not a vector');
24 elseif size( nodes,2 ) == 1
25     nodes = nodes';
26 end
27
28 switch request

```

```

29     case('standard')
30         [dof,dof_x,dof_y] = fsub_standard(nodes);
31
32     case('heaviside')
33         [dof,dof_x,dof_y] = fsub_heaviside(nodes,no_nodes);
34
35     case('neartip')
36         [dof,dof_x,dof_y] = fsub_neartip(nodes,no_nodes);
37         %dof_x = dof_x(:)';
38         %dof_y = dof_y(:)';
39         %dof    = dof(:)';
40
41     case('all')
42         [u,u_x,u_y] = fsub_standard(nodes);
43         [a,a_x,a_y] = fsub_heaviside(nodes,no_nodes);
44         [b,b_x,b_y] = fsub_neartip(nodes,no_nodes);
45
46         dof_x = [u_x a_x b_x(:)'];
47         dof_y = [u_y a_y b_y(:)'];
48         dof    = [u a b(:)'];
49
50     otherwise
51         error('fx fem_dofs : unknown request type')
52 end
53
54 end
55
56
57
58 function [u,u_x,u_y] = fsub_standard(nodes)
59 % Compute standard degrees of freedom
60 u_x = 2*nodes - 1;           % ux
61 u_y = 2*nodes;               % uy
62 u    = reshape( [ u_x ; u_y ] , 1 , [] );
63 end
64
65 function [a,a_x,a_y] = fsub_heaviside(nodes,no_nodes)
66 % Compute Heaviside degrees of freedom

```

```

67 a_x = 2*nodes + 2*no_nodes - 1;    % ax
68 a_y = 2*nodes + 2*no_nodes;        % ay
69 a    = reshape( [ a_x ; a_y ] , 1 , [] );
70 end
71
72 function [b,b_x,b_y] = fsub_neartip(nodes,no_nodes)
73 % Compute near tip enrichment degrees of freedom
74 b1x = 4*no_nodes + 2*nodes - 1;
75 b1y = 4*no_nodes + 2*nodes;
76 b1   = reshape( [ b1x ; b1y ] , 1 , [] );
77
78 b2x = 6*no_nodes + 2*nodes - 1;
79 b2y = 6*no_nodes + 2*nodes;
80 b2   = reshape( [ b2x ; b2y ] , 1 , [] );
81
82 b3x = 8*no_nodes + 2*nodes - 1;
83 b3y = 8*no_nodes + 2*nodes;
84 b3   = reshape( [ b3x ; b3y ] , 1 , [] );
85
86 b4x = 10*no_nodes + 2*nodes - 1;
87 b4y = 10*no_nodes + 2*nodes;
88 b4   = reshape( [ b4x ; b4y ] , 1 , [] );
89
90 b5x = 12*no_nodes + 2*nodes - 1;
91 b5y = 12*no_nodes + 2*nodes;
92 b5   = reshape( [ b5x ; b5y ] , 1 , [] );
93
94 b6x = 14*no_nodes + 2*nodes - 1;
95 b6y = 14*no_nodes + 2*nodes;
96 b6   = reshape( [ b6x ; b6y ] , 1 , [] );
97
98 b_x = [b1x b2x b3x b4x b5x b6x];
99 b_y = [b1y b2y b3y b4y b5y b6y];
100 b    = [b1  b2  b3  b4  b5  b6 ];
101 end

```

Script for element stiffness integration

```

1  function [ELEMENT,IP] = stiffness(SIM,ELEMENT,NODE,CRACK,
    MATERIAL)
2  %stiffness Performs element integrations.
3
4  % INPUTS:
5  %   ELEM      : current element structure
6  %   IPOINT    : integration point structure
7  %   D         : material matrix
8  %   etype     : element type
9  % OUTPUT:
10 %   ELEM      : current element structure
11 % NOTES:
12 %   - The structure IPOINT is changed at enriched element
    subroutine output
13 %   - Only 'quad' elements supported.
14 %
15 % Numbering convention supported
16 %
17 %   Quadrilateral
18 %   (4)         (3)
19 %   o - - - o   y
20 %   |         |   ^
21 %   |         |   |
22 %   o - - - o   + - > x
23 %   (1)         (2)
24
25 % Extracting crack quantities
26 Xc = CRACK.X( [1 end] );
27 Yc = CRACK.Y( [1 end] );
28 Cv = [ CRACK.X(end) - CRACK.X(end-1) ; CRACK.Y(end) - CRACK.
    Y(end-1) ];
29
30 IP(SIM.ELEMENTS) = struct('X',[],'Y',[]);
31 for n1 = 1 : SIM.ELEMENTS % Loops over elements
32
33     if ELEMENT(n1).TYPE == 0 % Standard element

```

```

34     [o,p,ww] = gq2d();
35 else
36     [o,p,ww] = subcells(NODE.X(SIM.CONMAT(n1,:)),...
37                         NODE.Y(SIM.CONMAT(n1,:)),SIM.SUBCELLS);
38 end
39 [Ni,dNido,dNidp,~,~,dN5do,dN5dp,dN6do,dN6dp] = q6elem(o,
40     p);
41 [~,detJ,invJ] = ffem_jacobian(NODE.X(SIM.CONMAT(n1,:))
42     ),...
43     NODE.Y(SIM.CONMAT(n1,:)),dNido,dNidp
44     ,7);
45 [dNidX,dNidY] = ffem_dervconvert(invJ,dNido,dNidp);
46 [dN5dX,dN5dY] = ffem_dervconvert(invJ,dN5do,dN5dp);
47 [dN6dX,dN6dY] = ffem_dervconvert(invJ,dN6do,dN6dp);
48 xip = Ni*NODE.X(SIM.CONMAT(n1,:));
49 yip = Ni*NODE.Y(SIM.CONMAT(n1,:));
50 [rip,oip] = fgeo_polarmap(xip,yip,Xc,Yc);
51 D = MATERIAL(ELEMENT(n1).MATERIAL).D;
52
53 % Computing Heaviside enrichment functions
54 [~,dM1dX,dM1dY] = fxfem_heaviside('signed','all',...
55     NODE.PSI(SIM.CONMAT(n1,:)),Ni,dNidX,
56     dNidY);
57
58 if ELEMENT(n1).TYPE == 0 || ELEMENT(n1).TYPE == 1
59     dM2dX = zeros(size(Ni,1),length(SIM.CONMAT(n1,:)),6)
60     ;
61     dM2dY = zeros(size(Ni,1),length(SIM.CONMAT(n1,:)),6)
62     ;
63 elseif ELEMENT(n1).TYPE == 2 || ELEMENT(n1).TYPE == 3
64     nt_vec = double(NODE.TYPE(ELEMENT(n1).NODES) == 2);
65     [rn,on]= fgeo_polarmap(NODE.X(SIM.CONMAT(n1,:)),...
66         NODE.Y(SIM.CONMAT(n1,:)),...
67         CRACK.X([1 end]),CRACK.Y([1 end]));
68     [~,dM2dX,dM2dY] = fxfem_neartip(Ni,dNidX,dNidY,...
69         Cv,on,rn,oip,rip,nt_vec);
70 end

```



```

66     % Element integration
67     K_CC = zeros(2*4+2*4+2*4*6, 2*4+2*4+2*4*6);
68     K_IC = zeros(4, 2*4+2*4+2*4*6);
69     K_CI = zeros(2*4+2*4+2*4*6, 4);
70     K_II = zeros(4, 4);
71     B_Ii = zeros(3, 4);
72     for n2 = 1 : size(Ni, 1) % Loop over integration points
73         B_I = [dN5dX(n2)    0          dN6dX(n2)    0
74                0          dN5dY(n2)    0          dN6dY(n2)
75                dN5dY(n2) dN5dX(n2) dN6dY(n2) dN6dX(n2)];
76         B_Ii = B_Ii + ww(n2) * B_I * detJ(n2);
77     end
78     V = ww' * detJ * SIM.THICKNESS;
79     B_IC = -B_Ii / V;
80     for n2 = 1 : size(Ni, 1) % Loop over integration points
81         Bu = fxfem_Bmats(dNidX(n2,:), dNidY(n2,:), '
            standard');
82         Ba = fxfem_Bmats(dM1dX(n2,:), dM1dY(n2,:), '
            heaviside');
83         Bb = fxfem_Bmats(dM2dX(n2, :, :), dM2dY(n2, :, :), '
            neartip');
84         B_I = [dN5dX(n2)    0          dN6dX(n2)    0
85                0          dN5dY(n2)    0          dN6dY(n2)
86                dN5dY(n2) dN5dX(n2) dN6dY(n2) dN6dX(n2)];
87
88         B_C = [Bu Ba Bb];
89         B_Ibar = B_I + B_IC;
90         K_CC = K_CC + ww(n2) * B_C' * D * B_C * detJ
            (n2);
91
92         if strcmp(SIM.ELEMENT_TYPE, 'Q6')
93             K_CI = K_CI + ww(n2) * B_C' * D * B_Ibar *
                detJ(n2);
94             K_IC = K_IC + ww(n2) * B_Ibar' * D * B_C *
                detJ(n2);
95             K_II = K_II + ww(n2) * B_Ibar' * D * B_Ibar *
                detJ(n2);
96         else

```

```
97         K_IC = zeros(4,48);
98         K_CI = zeros(48,4);
99         K_II = zeros(4,4);
100     end
101 end
102 if strcmp(SIM.ELEMENT_TYPE,'Q6')
103     ELEMENT(n1).STIFFNESS = (K_CC - K_CI*inv(K_II)*K_IC)
104         * SIM.THICKNESS;
105 else
106     ELEMENT(n1).STIFFNESS = K_CC * SIM.THICKNESS;
107 end
108 % Storing values
109 IP(n1).X      = xip;
110 IP(n1).Y      = yip;
111 IP(n1).W      = ww;
112 IP(n1).R      = rip;
113 IP(n1).O      = oip;
114 IP(n1).Ni     = Ni;
115 IP(n1).dNidX  = dNidX;
116 IP(n1).dNidY  = dNidY;
117 IP(n1).dM1dX  = dM1dX;
118 IP(n1).dM1dY  = dM1dY;
119 IP(n1).dM2dX  = dM2dX;
120 IP(n1).dM2dY  = dM2dY;
121 IP(n1).detJ   = detJ;
122 end
123 end
```

Heaviside enrichment definition

```

1 function [varargout] = fxfem_heaviside(type,request,Psi,
    varargin)
2 % fxfem_heaviside: Heaviside enrichment definition.
3 %   fxfem_heaviside(dNidX,dNidY,Ni,Psi,type) computes the
    Heaviside shape
4 %   functions and their derivatives.
5 %
6 % INPUTS: (variable input)
7 %   (1) type       : defines the Heaviside definition to be
    used:
8 %               > 'standard' for standard definition of
    the Heaviside
9 %               function, it has a value of 0 below and
    at the crack and
10 %               1 above the crack
11 %               > 'signed' = uses the sign of the psi
    level set function
12 %               so that the Heaviside function can have
    values of -1, 0
13 %               and 1
14 %   (2) request    : defines the requested outputs.
15 %               > 'function' request to return the M1
    enrichment
16 %               function at integration points (M1i)
17 %               > 'derivatives' request to return the
    derivatives of the
18 %               M1 function (dM1dX & dM1dY) at
    integration points
19 %               (dM1idX & dM1idY)
20 %               > 'all' request to return the enrichment
    function M1 and
21 %               its derivatives of the.
22 %   (3) Psi        : (q x 1) matrix of nodal Psi level set
    values.
23 %   (4) Ni         : (n x q) matrix of shape function values.

```

```

24 % (5) dNidX : (n x q) matrix of shape function
    derivatives with respect
25 %           to x-coordinate.
26 % (6) dNidY : (n x q) matrix of shape function
    derivatives with respect
27 %           to y-coordinate.
28 % OUTPUT: (variable output)
29 % (1) Mi : (n x q) matrix of Heaviside enriched shape
    function values
30 % (2) dMidX : (n x q) matrix of Heaviside enriched shape
    function
31 %           derivatives with respect to the x-coordinate
32 % (3) dMidY : (n x q) matrix of Heaviside enriched shape
    function
33 %           derivatives with respect to the y-coordinate
34 % (4) Hi : (n x q) matrix of shifted Heaviside function
    value
35 % (5) H : (n x 1) matrix of Heaviside enrichment
    functions at the
36 %           integration points
37 % NOTES:
38 % (1) Request for 'function' will require inputs 1, 2, 3,
    and 4. The
39 %     function will output 1, 4 and 5.
40 % (2) Request for 'derivatives' will require inputs 1, 2,
    3, 5 and 6. The
41 %     function will output 2 & 3.
42 % (3) Request for 'all' will require inputs 1, 2, 3, 4, 5
    and 6. The
43 %     function will output 1, 2, 3 & 4.
44
45 % Read requested output
46 if strcmp( request, 'function' ) == 1 && length( varargin )
    == 1
47     goto1 = 1;
48     Ni = varargin{1};
49 elseif strcmp( request, 'derivatives' ) == 1 && length(
    varargin ) == 3

```

```

50     goto1 = 2;
51     Ni     = varargin{1};
52     dNidX  = varargin{2};
53     dNidY  = varargin{3};
54     elseif strcmp( request, 'all' ) == 1 && length( varargin ) ==
55         3
56         goto1 = 3;
57         Ni     = varargin{1};
58         dNidX  = varargin{2};
59         dNidY  = varargin{3};
60     else
61         error('Unknown request or inconsistent inputs.')
62     end
63     % Test for heaviside function selection
64     if strcmp(type, 'standard') == 0 && strcmp(type, 'signed') ==
65         0
66         error('Unknown function definition')
67     end
68     % Interpolation of PSI level set values to integration
69     points
70     Psi_ip = Ni * Psi;
71     % Heaviside values
72     switch type
73         case('standard')
74             H_node = fsub_Hstandard( Psi );
75             H_ip = fsub_Hstandard( Psi_ip )';
76         case('signed')
77             H_node = fsub_Hsigned( Psi );
78             H_ip = fsub_Hsigned( Psi_ip )';
79     end
80
81     % Computing totals
82     no_ip = length( Psi_ip );
83     no_nodes = length( Psi );
84

```

```

85 % Rearranging Heaviside values
86 H_ip_rep = repmat( H_ip,1,no_nodes );
87 H_node_rep = repmat( H_node,no_ip,1 );
88
89 % Computing shifted Heaviside matrix
90 Hmod = H_ip_rep - H_node_rep;
91
92 % Computing Heaviside enriched matrices
93 switch gotol
94     case(1)
95         varargout{1} = Ni .* Hmod; % M1i
96         varargout{2} = Hmod;
97         varargout{3} = H_ip;
98     case(2)
99         varargout{1} = dNidX .* Hmod; % dM1idX
100        varargout{2} = dNidY .* Hmod; % dM1idY
101    case(3)
102        varargout{1} = Ni .* Hmod; % M1i
103        varargout{2} = dNidX .* Hmod; % dM1idX
104        varargout{3} = dNidY .* Hmod; % dM1idY
105        varargout{4} = Hmod; % H-Hk
106 end
107 end
108 function [H] = fsub_Hstandard(psi)
109 %fsub_Hstandard: Standard Heaviside function.
110 % Computation of the Heaviside function using the standard
    definition.
111
112 % Calculating the number of points for evaluation
113 no_points = length( psi );
114
115 % Setting all values to 0
116 H = zeros( 1, no_points );
117
118 % Assigning a value of 1 to points above the crack
119 H( sign( psi ) > 0 ) = 1;
120 end
121 function [ H ] = fsub_Hsigned( psi )

```

```
122 %fsub_Hsigned: Standard Heaviside function.  
123 %    Computation of the Heaviside function using the signed  
    definition.  
124  
125 % Applying sign values to points  
126 H = sign( psi )';  
127 end
```

Script for strain/displacement matrix assembly

```

1 function [B] = fxfem_Bmats(dfdx,dfdy,eval)
2 % fxfem_Bmats Sub-function to construct the strain/
   displacement matrices.
3 %
4 %   Evaluation types:
5 %       'standard'
6 %           dfdx = size(1,4)
7 %           dfdy = size(1,4)
8 %       'heaviside'
9 %           dfdx = size(1,4)
10 %          dfdy = size(1,4)
11 %       'neartip'
12 %           dfdx = size(1,4,6)
13 %           dfdy = size(1,4,6)
14
15 switch eval
16     case('standard')
17         B = zeros(3,8);
18         B(1,1:2:end) = dfdx;
19         B(2,2:2:end) = dfdy;
20         B(3,1:2:end) = dfdy;
21         B(3,2:2:end) = dfdx;
22     case('heaviside')
23         B = zeros(3,8);
24         B(1,1:2:end) = dfdx;
25         B(2,2:2:end) = dfdy;
26         B(3,1:2:end) = dfdy;
27         B(3,2:2:end) = dfdx;
28     case('neartip')
29         Bbi = zeros(3,8,6);
30         for n1 = 1 : 6 % Loop over isotropic near-tip
            functions
31             Bbi(1,1:2:end,n1) = dfdx(:, :, n1);
32             Bbi(2,2:2:end,n1) = dfdy(:, :, n1);
33             Bbi(3,1:2:end,n1) = dfdy(:, :, n1);
34             Bbi(3,2:2:end,n1) = dfdx(:, :, n1);

```

```
35         end
36         B = [ Bbi(:, :, 1) Bbi(:, :, 2) Bbi(:, :, 3) Bbi(:, :, 4)
               Bbi(:, :, 5) Bbi(:, :, 6)];
37     otherwise
38         error('Undefined evaluation')
39 end
40
41 end
```

```
function [xi,eta,WW,Xip,Yip] = subcells(x,y,div)
%subcells: tessellates a Q4 element and assign integration
    points in
%REFERENCE space
%
% INPUTS:
%   x:   (4 x 1) column vector with x-coordinates
        quadrilateral nodes
%   y:   (4 x 1) column vector with y-coordinates
        quadrilateral nodes
%   div: number of divisions in x or y coordinates.
%
% OUTPUT:
%   xi:  (n x 1) column vector with x-coordinates of IPs in
        REFERENCE space
%   eta: (n x 1) column vector with y-coordinates of IPs in
        REFERENCE space
%   WW:  (n x 1) transformed integration weights
%   Xip: (n x 1) column vector with y-coordinates of IPs in
        REAL space
%   Yip: (n x 1) column vector with y-coordinates of IPs in
        REAL space
% Check inputs
if isvector(x) == 0 || isvector(y) == 0
    error('Inputs are not vectors');
else
    if size(x,2) == 1; x = x'; end
    if size(y,2) == 1; y = y'; end
end

if length(x) ~= length(y)
    error('Inputs have unequal length')
end

dop = 2; % Limited to 4 IP per cell!
```

```

30 [X,Y,conmat]      = rgrid(x,y,div);
31 [o,p,ww]         = gq2d();
32 [Ni,dNido,dNidp] = q6elem(o,p);
33 Xip              = zeros(dop^2*div^2,1);
34 Yip              = zeros(dop^2*div^2,1);
35 detJ             = zeros(dop^2*div^2,1);
36 WW              = zeros(dop^2*div^2,1);
37 c1               = 1;
38
39 for i = 1 : size(conmat,1) % loop over cells
40     detJ(c1:c1+dop^2-1) = ffem_jacobian(X(conmat(i,:)),Y(
        conmat(i,:)),dNido,dNidp,2);
41     Xip(c1:c1+dop^2-1)  = Ni*X(conmat(i,:));
42     Yip(c1:c1+dop^2-1)  = Ni*Y(conmat(i,:));
43     WW(c1:c1+dop^2-1)   = ww.*detJ(c1:c1+dop^2-1);
44     c1                  = c1 + dop^2;
45 end
46 [xi,eta]            = q4invmap(x,y,Xip,Yip);
47 end

```

Near-tip enrichment definition

```

1 function [M2,dM2dX,dM2dY] = fxfem_neartip(Ni,dNidX,dNidY,Vc,
    On,Rn,Oip,Rip,n_type)
2
3 % Angle between X axis and crack segment
4 CROSS = cross([1 0 0],[Vc ; 0]);
5 a      = atan2(CROSS(3),dot([1 0 0],[Vc ; 0]));
6 n_ip   = length(Oip);
7
8 f1      = @(r,o) sqrt(r) * sin(o/2);
9 f2      = @(r,o) sqrt(r) * cos(o/2);
10 f3      = @(r,o) sqrt(r) * sin(o/2) * sin(o);
11 f4      = @(r,o) sqrt(r) * cos(o/2) * sin(o);
12 f5      = @(r,o) r*sin(o);
13 f6      = @(r,o) r*cos(o);
14
15 df1dr   = @(r,o) 1/(2*sqrt(r)) * sin(o/2);
16 df1do   = @(r,o) sqrt(r)/2      * cos(o/2);
17
18 df2dr   = @(r,o) 1/(2*sqrt(r)) * cos(o/2);
19 df2do   = @(r,o) -sqrt(r)/2     * sin(o/2);
20
21 df3dr   = @(r,o) 1/(2*sqrt(r)) * sin(o/2) * sin(o);
22 df3do   = @(r,o) sqrt(r)        * (0.5*cos(o/2)*sin(o)+sin(o/2)
    *cos(o));
23
24 df4dr   = @(r,o) 1/(2*sqrt(r)) * cos(o/2) * sin(o);
25 df4do   = @(r,o) sqrt(r)        * (cos(o/2)*cos(o) - 0.5*sin(o)
    /2)*sin(o));
26
27 df5dr   = @(r,o) sin(o);
28 df5do   = @(r,o) r*cos(o);
29
30 df6dr   = @(r,o) cos(o);
31 df6do   = @(r,o) -r*sin(o);
32
33 T       = @(r,o) [cos(o) -sin(o)/r ; sin(o) cos(o)/r];

```

```

34 Q      = @(a)      [cos(a) -sin(a)      ; sin(a) cos(a)      ];
35
36
37 F11 = f1(Rn(1),On(1));
38 F12 = f1(Rn(2),On(2));
39 F13 = f1(Rn(3),On(3));
40 F14 = f1(Rn(4),On(4));
41 F21 = f2(Rn(1),On(1));
42 F22 = f2(Rn(2),On(2));
43 F23 = f2(Rn(3),On(3));
44 F24 = f2(Rn(4),On(4));
45 F31 = f3(Rn(1),On(1));
46 F32 = f3(Rn(2),On(2));
47 F33 = f3(Rn(3),On(3));
48 F34 = f3(Rn(4),On(4));
49 F41 = f4(Rn(1),On(1));
50 F42 = f4(Rn(2),On(2));
51 F43 = f4(Rn(3),On(3));
52 F44 = f4(Rn(4),On(4));
53 F51 = f5(Rn(1),On(1));
54 F52 = f5(Rn(2),On(2));
55 F53 = f5(Rn(3),On(3));
56 F54 = f5(Rn(4),On(4));
57 F61 = f6(Rn(1),On(1));
58 F62 = f6(Rn(2),On(2));
59 F63 = f6(Rn(3),On(3));
60 F64 = f6(Rn(4),On(4));
61
62
63 C1 = T(Rn(1),On(1)) * Q(a);
64 C2 = T(Rn(2),On(2)) * Q(a);
65 C3 = T(Rn(3),On(3)) * Q(a);
66 C4 = T(Rn(4),On(4)) * Q(a);
67
68 df1dX1 = C1(1,:) * [df1dr(Rn(1),On(1)) df1do(Rn(1),On(1))]';
69 df1dY1 = C1(2,:) * [df1dr(Rn(1),On(1)) df1do(Rn(1),On(1))]';
70 df2dX1 = C1(1,:) * [df2dr(Rn(1),On(1)) df2do(Rn(1),On(1))]';
71 df2dY1 = C1(2,:) * [df2dr(Rn(1),On(1)) df2do(Rn(1),On(1))]';

```

```

72 df3dX1 = C1(1,:) * [df3dr(Rn(1),On(1)) df3do(Rn(1),On(1))]';
73 df3dY1 = C1(2,:) * [df3dr(Rn(1),On(1)) df3do(Rn(1),On(1))]';
74 df4dX1 = C1(1,:) * [df4dr(Rn(1),On(1)) df4do(Rn(1),On(1))]';
75 df4dY1 = C1(2,:) * [df4dr(Rn(1),On(1)) df4do(Rn(1),On(1))]';
76 df5dX1 = C1(1,:) * [df5dr(Rn(1),On(1)) df5do(Rn(1),On(1))]';
77 df5dY1 = C1(2,:) * [df5dr(Rn(1),On(1)) df5do(Rn(1),On(1))]';
78 df6dX1 = C1(1,:) * [df6dr(Rn(1),On(1)) df6do(Rn(1),On(1))]';
79 df6dY1 = C1(2,:) * [df6dr(Rn(1),On(1)) df6do(Rn(1),On(1))]';
80
81 df1dX2 = C2(1,:) * [df1dr(Rn(2),On(2)) df1do(Rn(2),On(2))]';
82 df1dY2 = C2(2,:) * [df1dr(Rn(2),On(2)) df1do(Rn(2),On(2))]';
83 df2dX2 = C2(1,:) * [df2dr(Rn(2),On(2)) df2do(Rn(2),On(2))]';
84 df2dY2 = C2(2,:) * [df2dr(Rn(2),On(2)) df2do(Rn(2),On(2))]';
85 df3dX2 = C2(1,:) * [df3dr(Rn(2),On(2)) df3do(Rn(2),On(2))]';
86 df3dY2 = C2(2,:) * [df3dr(Rn(2),On(2)) df3do(Rn(2),On(2))]';
87 df4dX2 = C2(1,:) * [df4dr(Rn(2),On(2)) df4do(Rn(2),On(2))]';
88 df4dY2 = C2(2,:) * [df4dr(Rn(2),On(2)) df4do(Rn(2),On(2))]';
89 df5dX2 = C2(1,:) * [df5dr(Rn(2),On(2)) df5do(Rn(2),On(2))]';
90 df5dY2 = C2(2,:) * [df5dr(Rn(2),On(2)) df5do(Rn(2),On(2))]';
91 df6dX2 = C2(1,:) * [df6dr(Rn(2),On(2)) df6do(Rn(2),On(2))]';
92 df6dY2 = C2(2,:) * [df6dr(Rn(2),On(2)) df6do(Rn(2),On(2))]';
93
94 df1dX3 = C3(1,:) * [df1dr(Rn(3),On(3)) df1do(Rn(3),On(3))]';
95 df1dY3 = C3(2,:) * [df1dr(Rn(3),On(3)) df1do(Rn(3),On(3))]';
96 df2dX3 = C3(1,:) * [df2dr(Rn(3),On(3)) df2do(Rn(3),On(3))]';
97 df2dY3 = C3(2,:) * [df2dr(Rn(3),On(3)) df2do(Rn(3),On(3))]';
98 df3dX3 = C3(1,:) * [df3dr(Rn(3),On(3)) df3do(Rn(3),On(3))]';
99 df3dY3 = C3(2,:) * [df3dr(Rn(3),On(3)) df3do(Rn(3),On(3))]';
100 df4dX3 = C3(1,:) * [df4dr(Rn(3),On(3)) df4do(Rn(3),On(3))]';
101 df4dY3 = C3(2,:) * [df4dr(Rn(3),On(3)) df4do(Rn(3),On(3))]';
102 df5dX3 = C3(1,:) * [df5dr(Rn(3),On(3)) df5do(Rn(3),On(3))]';
103 df5dY3 = C3(2,:) * [df5dr(Rn(3),On(3)) df5do(Rn(3),On(3))]';
104 df6dX3 = C3(1,:) * [df6dr(Rn(3),On(3)) df6do(Rn(3),On(3))]';
105 df6dY3 = C3(2,:) * [df6dr(Rn(3),On(3)) df6do(Rn(3),On(3))]';
106
107 df1dX4 = C4(1,:) * [df1dr(Rn(4),On(4)) df1do(Rn(4),On(4))]';
108 df1dY4 = C4(2,:) * [df1dr(Rn(4),On(4)) df1do(Rn(4),On(4))]';
109 df2dX4 = C4(1,:) * [df2dr(Rn(4),On(4)) df2do(Rn(4),On(4))]';

```

```

110 df2dY4 = C4(2,:) * [df2dr(Rn(4),On(4)) df2do(Rn(4),On(4))]';
111 df3dX4 = C4(1,:) * [df3dr(Rn(4),On(4)) df3do(Rn(4),On(4))]';
112 df3dY4 = C4(2,:) * [df3dr(Rn(4),On(4)) df3do(Rn(4),On(4))]';
113 df4dX4 = C4(1,:) * [df4dr(Rn(4),On(4)) df4do(Rn(4),On(4))]';
114 df4dY4 = C4(2,:) * [df4dr(Rn(4),On(4)) df4do(Rn(4),On(4))]';
115 df5dX4 = C4(1,:) * [df5dr(Rn(4),On(4)) df5do(Rn(4),On(4))]';
116 df5dY4 = C4(2,:) * [df5dr(Rn(4),On(4)) df5do(Rn(4),On(4))]';
117 df6dX4 = C4(1,:) * [df6dr(Rn(4),On(4)) df6do(Rn(4),On(4))]';
118 df6dY4 = C4(2,:) * [df6dr(Rn(4),On(4)) df6do(Rn(4),On(4))]';
119
120 dM2dX = zeros(n_ip,4,6);
121 dM2dY = zeros(n_ip,4,6);
122 M2     = zeros(n_ip,4,6);
123 for i = 1 : n_ip
124     N1 = Ni(i,1);
125     N2 = Ni(i,2);
126     N3 = Ni(i,3);
127     N4 = Ni(i,4);
128
129     dN1dX = dNidX(i,1);
130     dN2dX = dNidX(i,2);
131     dN3dX = dNidX(i,3);
132     dN4dX = dNidX(i,4);
133
134     dN1dY = dNidY(i,1);
135     dN2dY = dNidY(i,2);
136     dN3dY = dNidY(i,3);
137     dN4dY = dNidY(i,4);
138
139     R = sum(N1*n_type(1) + N2*n_type(2) + N3*n_type
140             (3) + N4*n_type(4),2);
141     dRdX = sum(dN1dX*n_type(1) + dN2dX*n_type(2) + dN3dX*
142             n_type(3) + dN4dX*n_type(4),2);
143     dRdY = sum(dN1dY*n_type(1) + dN2dY*n_type(2) + dN3dY*
144             n_type(3) + dN4dY*n_type(4),2);
145
146     r = Rip(i);
147     o = Oip(i);

```

```

145      C = T(r,o) * Q(a);
146
147      F1      = f1(r,o);
148      F2      = f2(r,o);
149      F3      = f3(r,o);
150      F4      = f4(r,o);
151      F5      = f5(r,o);
152      F6      = f6(r,o);
153
154      M2(i,1,1) = N1 * (F1 - F11) * R;
155      M2(i,2,1) = N2 * (F1 - F12) * R;
156      M2(i,3,1) = N3 * (F1 - F13) * R;
157      M2(i,4,1) = N4 * (F1 - F14) * R;
158
159      M2(i,1,2) = N1 * (F2 - F21) * R;
160      M2(i,2,2) = N2 * (F2 - F22) * R;
161      M2(i,3,2) = N3 * (F2 - F23) * R;
162      M2(i,4,2) = N4 * (F2 - F24) * R;
163
164      M2(i,1,3) = N1 * (F3 - F31) * R;
165      M2(i,2,3) = N2 * (F3 - F32) * R;
166      M2(i,3,3) = N3 * (F3 - F33) * R;
167      M2(i,4,3) = N4 * (F3 - F34) * R;
168
169      M2(i,1,4) = N1 * (F4 - F41) * R;
170      M2(i,2,4) = N2 * (F4 - F42) * R;
171      M2(i,3,4) = N3 * (F4 - F43) * R;
172      M2(i,4,4) = N4 * (F4 - F44) * R;
173
174      M2(i,1,5) = N1 * (F5 - F51) * R;
175      M2(i,2,5) = N2 * (F5 - F52) * R;
176      M2(i,3,5) = N3 * (F5 - F53) * R;
177      M2(i,4,5) = N4 * (F5 - F54) * R;
178
179      M2(i,1,6) = N1 * (F6 - F61) * R;
180      M2(i,2,6) = N2 * (F6 - F62) * R;
181      M2(i,3,6) = N3 * (F6 - F63) * R;
182      M2(i,4,6) = N4 * (F6 - F64) * R;

```



```

183
184     df1dX = C(1,:) * [df1dr(r,o) df1do(r,o)]';
185     df1dY = C(2,:) * [df1dr(r,o) df1do(r,o)]';
186
187     df2dX = C(1,:) * [df2dr(r,o) df2do(r,o)]';
188     df2dY = C(2,:) * [df2dr(r,o) df2do(r,o)]';
189
190     df3dX = C(1,:) * [df3dr(r,o) df3do(r,o)]';
191     df3dY = C(2,:) * [df3dr(r,o) df3do(r,o)]';
192
193     df4dX = C(1,:) * [df4dr(r,o) df4do(r,o)]';
194     df4dY = C(2,:) * [df4dr(r,o) df4do(r,o)]';
195
196     df5dX = C(1,:) * [df5dr(r,o) df5do(r,o)]';
197     df5dY = C(2,:) * [df5dr(r,o) df5do(r,o)]';
198
199     df6dX = C(1,:) * [df6dr(r,o) df6do(r,o)]';
200     df6dY = C(2,:) * [df6dr(r,o) df6do(r,o)]';
201
202     dF1dX1 = dN1dX*(F1-F11)*R + N1*(df1dX-df1dX1)*R + N1*(F1
203             -F11)*dRdX;
204     dF1dX2 = dN2dX*(F1-F12)*R + N2*(df1dX-df1dX2)*R + N2*(F1
205             -F12)*dRdX;
206     dF1dX3 = dN3dX*(F1-F13)*R + N3*(df1dX-df1dX3)*R + N3*(F1
207             -F13)*dRdX;
208     dF1dX4 = dN4dX*(F1-F14)*R + N4*(df1dX-df1dX4)*R + N4*(F1
209             -F14)*dRdX;
210
211     dF1dY1 = dN1dY*(F1-F11)*R + N1*(df1dY-df1dY1)*R + N1*(F1
212             -F11)*dRdY;
213     dF1dY2 = dN2dY*(F1-F12)*R + N2*(df1dY-df1dY2)*R + N2*(F1
214             -F12)*dRdY;
215     dF1dY3 = dN3dY*(F1-F13)*R + N3*(df1dY-df1dY3)*R + N3*(F1
216             -F13)*dRdY;
217     dF1dY4 = dN4dY*(F1-F14)*R + N4*(df1dY-df1dY4)*R + N4*(F1
218             -F14)*dRdY;
219
220     dF2dX1 = dN1dX*(F2-F21)*R + N1*(df2dX-df2dX1)*R + N1*(F2
221             -F21)*dRdX;

```

$$\begin{aligned}
212 \quad dF2dX2 &= dN2dX*(F2-F22)*R + N2*(df2dX-df2dX2)*R + N2*(F2 \\
&\quad -F22)*dRdX; \\
213 \quad dF2dX3 &= dN3dX*(F2-F23)*R + N3*(df2dX-df2dX3)*R + N3*(F2 \\
&\quad -F23)*dRdX; \\
214 \quad dF2dX4 &= dN4dX*(F2-F24)*R + N4*(df2dX-df2dX4)*R + N4*(F2 \\
&\quad -F24)*dRdX; \\
215 \quad dF2dY1 &= dN1dY*(F2-F21)*R + N1*(df2dY-df2dY1)*R + N1*(F2 \\
&\quad -F21)*dRdY; \\
216 \quad dF2dY2 &= dN2dY*(F2-F22)*R + N2*(df2dY-df2dY2)*R + N2*(F2 \\
&\quad -F22)*dRdY; \\
217 \quad dF2dY3 &= dN3dY*(F2-F23)*R + N3*(df2dY-df2dY3)*R + N3*(F2 \\
&\quad -F23)*dRdY; \\
218 \quad dF2dY4 &= dN4dY*(F2-F24)*R + N4*(df2dY-df2dY4)*R + N4*(F2 \\
&\quad -F24)*dRdY; \\
219 \quad & \\
220 \quad dF3dX1 &= dN1dX*(F3-F31)*R + N1*(df3dX-df3dX1)*R + N1*(F3 \\
&\quad -F31)*dRdX; \\
221 \quad dF3dX2 &= dN2dX*(F3-F32)*R + N2*(df3dX-df3dX2)*R + N2*(F3 \\
&\quad -F32)*dRdX; \\
222 \quad dF3dX3 &= dN3dX*(F3-F33)*R + N3*(df3dX-df3dX3)*R + N3*(F3 \\
&\quad -F33)*dRdX; \\
223 \quad dF3dX4 &= dN4dX*(F3-F34)*R + N4*(df3dX-df3dX4)*R + N4*(F3 \\
&\quad -F34)*dRdX; \\
224 \quad dF3dY1 &= dN1dY*(F3-F31)*R + N1*(df3dY-df3dY1)*R + N1*(F3 \\
&\quad -F31)*dRdY; \\
225 \quad dF3dY2 &= dN2dY*(F3-F32)*R + N2*(df3dY-df3dY2)*R + N2*(F3 \\
&\quad -F32)*dRdY; \\
226 \quad dF3dY3 &= dN3dY*(F3-F33)*R + N3*(df3dY-df3dY3)*R + N3*(F3 \\
&\quad -F33)*dRdY; \\
227 \quad dF3dY4 &= dN4dY*(F3-F34)*R + N4*(df3dY-df3dY4)*R + N4*(F3 \\
&\quad -F34)*dRdY; \\
228 \quad & \\
229 \quad dF4dX1 &= dN1dX*(F4-F41)*R + N1*(df4dX-df4dX1)*R + N1*(F4 \\
&\quad -F41)*dRdX; \\
230 \quad dF4dX2 &= dN2dX*(F4-F42)*R + N2*(df4dX-df4dX2)*R + N2*(F4 \\
&\quad -F42)*dRdX; \\
231 \quad dF4dX3 &= dN3dX*(F4-F43)*R + N3*(df4dX-df4dX3)*R + N3*(F4 \\
&\quad -F43)*dRdX;
\end{aligned}$$

$$\begin{aligned}
232 \quad dF4dX4 &= dN4dX*(F4-F44)*R + N4*(df4dX-df4dX4)*R + N4*(F4 \\
&\quad -F44)*dRdX; \\
233 \quad dF4dY1 &= dN1dY*(F4-F41)*R + N1*(df4dY-df4dY1)*R + N1*(F4 \\
&\quad -F41)*dRdY; \\
234 \quad dF4dY2 &= dN2dY*(F4-F42)*R + N2*(df4dY-df4dY2)*R + N2*(F4 \\
&\quad -F42)*dRdY; \\
235 \quad dF4dY3 &= dN3dY*(F4-F43)*R + N3*(df4dY-df4dY3)*R + N3*(F4 \\
&\quad -F43)*dRdY; \\
236 \quad dF4dY4 &= dN4dY*(F4-F44)*R + N4*(df4dY-df4dY4)*R + N4*(F4 \\
&\quad -F44)*dRdY; \\
237 \quad & \\
238 \quad dF5dX1 &= dN1dX*(F5-F51)*R + N1*(df5dX-df5dX1)*R + N1*(F5 \\
&\quad -F51)*dRdX; \\
239 \quad dF5dX2 &= dN2dX*(F5-F52)*R + N2*(df5dX-df5dX2)*R + N2*(F5 \\
&\quad -F52)*dRdX; \\
240 \quad dF5dX3 &= dN3dX*(F5-F53)*R + N3*(df5dX-df5dX3)*R + N3*(F5 \\
&\quad -F53)*dRdX; \\
241 \quad dF5dX4 &= dN4dX*(F5-F54)*R + N4*(df5dX-df5dX4)*R + N4*(F5 \\
&\quad -F54)*dRdX; \\
242 \quad dF5dY1 &= dN1dY*(F5-F51)*R + N1*(df5dY-df5dY1)*R + N1*(F5 \\
&\quad -F51)*dRdY; \\
243 \quad dF5dY2 &= dN2dY*(F5-F52)*R + N2*(df5dY-df5dY2)*R + N2*(F5 \\
&\quad -F52)*dRdY; \\
244 \quad dF5dY3 &= dN3dY*(F5-F53)*R + N3*(df5dY-df5dY3)*R + N3*(F5 \\
&\quad -F53)*dRdY; \\
245 \quad dF5dY4 &= dN4dY*(F5-F54)*R + N4*(df5dY-df5dY4)*R + N4*(F5 \\
&\quad -F54)*dRdY; \\
246 \quad & \\
247 \quad dF6dX1 &= dN1dX*(F6-F61)*R + N1*(df6dX-df6dX1)*R + N1*(F6 \\
&\quad -F61)*dRdX; \\
248 \quad dF6dX2 &= dN2dX*(F6-F62)*R + N2*(df6dX-df6dX2)*R + N2*(F6 \\
&\quad -F62)*dRdX; \\
249 \quad dF6dX3 &= dN3dX*(F6-F63)*R + N3*(df6dX-df6dX3)*R + N3*(F6 \\
&\quad -F63)*dRdX; \\
250 \quad dF6dX4 &= dN4dX*(F6-F64)*R + N4*(df6dX-df6dX4)*R + N4*(F6 \\
&\quad -F64)*dRdX; \\
251 \quad dF6dY1 &= dN1dY*(F6-F61)*R + N1*(df6dY-df6dY1)*R + N1*(F6 \\
&\quad -F61)*dRdY;
\end{aligned}$$

```

252     dF6dY2 = dN2dY*(F6-F62)*R + N2*(df6dY-df6dY2)*R + N2*(F6
        -F62)*dRdY;
253     dF6dY3 = dN3dY*(F6-F63)*R + N3*(df6dY-df6dY3)*R + N3*(F6
        -F63)*dRdY;
254     dF6dY4 = dN4dY*(F6-F64)*R + N4*(df6dY-df6dY4)*R + N4*(F6
        -F64)*dRdY;
255
256     dM2dX(i, :, 1) = [dF1dX1 dF1dX2 dF1dX3 dF1dX4];
257     dM2dX(i, :, 2) = [dF2dX1 dF2dX2 dF2dX3 dF2dX4];
258     dM2dX(i, :, 3) = [dF3dX1 dF3dX2 dF3dX3 dF3dX4];
259     dM2dX(i, :, 4) = [dF4dX1 dF4dX2 dF4dX3 dF4dX4];
260     dM2dX(i, :, 5) = [dF5dX1 dF5dX2 dF5dX3 dF5dX4];
261     dM2dX(i, :, 6) = [dF6dX1 dF6dX2 dF6dX3 dF6dX4];
262
263     dM2dY(i, :, 1) = [dF1dY1 dF1dY2 dF1dY3 dF1dY4];
264     dM2dY(i, :, 2) = [dF2dY1 dF2dY2 dF2dY3 dF2dY4];
265     dM2dY(i, :, 3) = [dF3dY1 dF3dY2 dF3dY3 dF3dY4];
266     dM2dY(i, :, 4) = [dF4dY1 dF4dY2 dF4dY3 dF4dY4];
267     dM2dY(i, :, 5) = [dF5dY1 dF5dY2 dF5dY3 dF5dY4];
268     dM2dY(i, :, 6) = [dF6dY1 dF6dY2 dF6dY3 dF6dY4];
269 end
270 end

```

Script for XFEM solution

```

1 function [ SIM ] = fxfem_solver( ELEMENT,NODE,BC,SIM )
2 % fxfem_solver Solves the FEM problem
3
4 % Defining the total Degrees of Freedom (DOF) of the system
   and the DOF
5 % vector
6 no_nodes = SIM.NODES;
7 dof      = 2*SIM.NODES + 2*SIM.NODES + 2*SIM.NODES*6;
8 Vdof     = ( 1 : dof )';
9
10 % Initializing displacements
11 NODE.UX = zeros(SIM.NODES,1);
12 NODE.UY = NODE.UX;
13 NODE.AX = zeros(SIM.NODES,1);
14 NODE.AY = NODE.AX;
15 NODE.BX = zeros(SIM.NODES,4);
16 NODE.BY = NODE.BX;
17
18 % Assembly of global stiffness matrix
19 Kg = zeros( dof,dof );
20 for n1 = 1 : SIM.ELEMENTS % Loop over elements
21     i      = ELEMENT(n1).CONVEC;
22     Kg(i,i) = Kg(i,i) + ELEMENT(n1).STIFFNESS;
23
24 end
25
26 % Activating node Standard DOF
27 U_nodes = NODE.ID(NODE.STATE == 1);
28 [U_act] = fxfem_dofs(U_nodes,no_nodes,'standard');
29
30 % Activating node Heaviside DOF
31 A_nodes = NODE.ID(NODE.TYPE == 1);
32 [A_act] = fxfem_dofs(A_nodes,no_nodes,'heaviside');
33
34 % Activating node Isotropic Near-tip DOF

```

```

35 B_nodes = unique(SIM.CONMAT(sum(NODE.TYPE(SIM.CONMAT)==2,2)
    <4 & ...
36         sum(NODE.TYPE(SIM.CONMAT)==2,2)>0,:));
37 [B_act] = fxfem_dofs(B_nodes,no_nodes,'neartip');
38
39 % Declaring the 'active' and 'constrained' DOF
40 act_nodes = sort([U_act A_act B_act]);
41 con_nodes = setdiff(Vdof,act_nodes);
42
43 Kcc = sparse( Kg( con_nodes , con_nodes ) );
44 Kca = sparse( Kg( con_nodes , act_nodes ) );
45 Kac = sparse( Kg( act_nodes , con_nodes ) );
46 Kaa = sparse( Kg( act_nodes , act_nodes ) );
47
48 % Initializing variables
49 f = zeros( dof , 1 );
50 U = zeros( dof , 1 );
51
52 % Determining node DOF with boundary conditions
53 U_nodes = BC.DISP.NODE;
54 [~,Ux_con,Uy_con] = fxfem_dofs(U_nodes,no_nodes,'standard');
55
56 % Adding Dirichlet boundary conditions to the system
    displacement
57 % vector
58 U(Ux_con) = BC.DISP.UX;
59 U(Uy_con) = BC.DISP.UY;
60
61 % Adding Newman boundary conditions to the system Force
    vector
62 if isempty(BC.FORCE.NODE) == 0
63     [~,Ux_con,Uy_con] = fxfem_dofs(BC.FORCE.NODE,no_nodes,'
        standard');
64     f(Ux_con) = BC.FORCE.FX;
65     f(Uy_con) = BC.FORCE.FY;
66 end
67
68 % Reducing the system of equations

```

```
69 Fa = f(act_nodes);
70 Uc = U(con_nodes);
71
72 % Solving the system of equations
73 Ua = Kaa\'(Fa-Kac*Uc);
74 Fc = Kcc*Uc + Kca*Ua;
75
76 % Store solution values
77 U(act_nodes) = Ua;
78 f(con_nodes) = Fc;
79
80 SIM.d = U;
81 SIM.K = Kg;
82 SIM.f = f;
83 end
```

Script for XFEM displacement solution

```

1 function [NODE]=xfem_disp(SIM,NODE,ELEMENT,CRACK,MATERIAL)
2 %xfem_disp calculates the displacements of the nodes
3
4 Cv = [ CRACK.X(end) - CRACK.X(end-1) ; CRACK.Y(end) - CRACK.
      Y(end-1) ];
5 o = [-1 1 1 -1];
6 p = [-1 -1 1 1];
7 [Ni,dNido,dNidp] = ffem_q4elem(o,p);
8
9 MAT = zeros(SIM.NODES,9);
10
11
12 for n1 = 1 : SIM.ELEMENTS % Loop over elements
13
14     X = NODE.X(ELEMENT(n1).NODES);
15     Y = NODE.Y(ELEMENT(n1).NODES);
16     invJ = ffem_jacobian(X,Y,dNido,dNidp,4);
17     matJ = ffem_jacobian(X,Y,dNido,dNidp,1);
18     di = SIM.d(ELEMENT(n1).CONVEC);
19     Psi = NODE.PSI(ELEMENT(n1).NODES);
20     r = NODE.R(ELEMENT(n1).NODES);
21     o = NODE.O(ELEMENT(n1).NODES);
22
23     [dNidX,dNidY] = ffem_dervconvert(invJ,dNido,dNidp);
24
25     [M1,dM1dX,dM1dY] = fxfem_heaviside('signed','all',Psi,
      Ni,dNidX,dNidY);
26     nt_vec = double(NODE.TYPE(ELEMENT(n1).NODES) == 2);
27     [M2,dM2dX,dM2dY] = fxfem_neartip(Ni,dNidX,dNidY,Cv,o,r
      ,o,r,nt_vec);
28
29     [dM1do,dM1dp] = ffem_dervconvert(matJ,dM1dX,dM1dY);
30     [dM2do,dM2dp] = ffem_dervconvert(matJ,dM2dX,dM2dY);
31
32     for i = 1 : 4 % Loop over nodes
33         Bu = fxfem_Bmats(dNido(i,:),dNidp(i,:), 'standard');

```



```

34     Ba = fxfem_Bmats(dM1do(i,:),dM1dp(i,:), 'heaviside');
35     Bb = fxfem_Bmats(dM2do(i,:,:),dM2dp(i,:,:), 'neartip'
36         );
37
38     u = fxfem_dofs(ELEMENT(n1).NODES,SIM.NODES, 'standard
39         ')';
40
41     a = fxfem_dofs(ELEMENT(n1).NODES,SIM.NODES, '
42         heaviside')';
43
44     b = fxfem_dofs(ELEMENT(n1).NODES,SIM.NODES, 'neartip'
45         )';
46
47     D = MATERIAL(ELEMENT(n1).MATERIAL).D;
48
49     e = Bu*u + Ba*a + Bb*b;
50
51     s = D*e;
52
53
54     Mi = [Ni(i,:) M1(i,:) M2(i,:,1) M2(i,:,2) M2(i,:,3)
55         M2(i,:,4) M2(i,:,5) M2(i,:,6)];
56
57     dx = Mi * di(1:2:end);
58     dy = Mi * di(2:2:end);
59
60
61     MAT(ELEMENT(n1).NODES(i),1) = MAT(ELEMENT(n1).NODES(
62         i),1) + dx;    % X displacement
63
64     MAT(ELEMENT(n1).NODES(i),2) = MAT(ELEMENT(n1).NODES(
65         i),2) + dy;    % Y displacement
66
67     MAT(ELEMENT(n1).NODES(i),3) = MAT(ELEMENT(n1).NODES(
68         i),3) + e(1);  % Strain in XX
69
70     MAT(ELEMENT(n1).NODES(i),4) = MAT(ELEMENT(n1).NODES(
71         i),4) + e(2);  % Strain in YY
72
73     MAT(ELEMENT(n1).NODES(i),5) = MAT(ELEMENT(n1).NODES(
74         i),5) + e(3);  % Strain in XY
75
76     MAT(ELEMENT(n1).NODES(i),6) = MAT(ELEMENT(n1).NODES(
77         i),6) + s(1);  % Stress in XX
78
79     MAT(ELEMENT(n1).NODES(i),7) = MAT(ELEMENT(n1).NODES(
80         i),7) + s(2);  % Stress in YY
81
82     MAT(ELEMENT(n1).NODES(i),8) = MAT(ELEMENT(n1).NODES(
83         i),8) + s(3);  % Stress in XY

```

```
58         MAT(ELEMENT(n1).NODES(i),9) = MAT(ELEMENT(n1).NODES(  
           i),9) + 1;           % Counter  
59     end  
60 end  
61  
62 NODE.dx = MAT(:,1) ./ MAT(:,9);  
63 NODE.dy = MAT(:,2) ./ MAT(:,9);  
64 NODE.exx = MAT(:,3) ./ MAT(:,9);  
65 NODE.eyy = MAT(:,4) ./ MAT(:,9);  
66 NODE.exy = MAT(:,5) ./ MAT(:,9);  
67 NODE.sxx = MAT(:,6) ./ MAT(:,9);  
68 NODE.syy = MAT(:,7) ./ MAT(:,9);  
69 NODE.sxy = MAT(:,8) ./ MAT(:,9);  
70  
71 end
```

Script for XFEM stress/strain extrapolation to nodes

```

1  function [NODE]=fx fem_StrainStressNODE(IP,ELEMENT,NODE,SIM,
    request)
2  SOL = struct('CUM',zeros(SIM.NODES,1),'COUNT',zeros(SIM.
    NODES,1));
3  for n1 = 1 : SIM.ELEMENTS % loop over elements
4      x = IP(n1).X;
5      y = IP(n1).Y;
6      z = IP(n1).(request);
7
8      X = NODE.X(ELEMENT(n1).NODES);
9      Y = NODE.Y(ELEMENT(n1).NODES);
10     Z = griddata(x,y,z,X,Y,'v4');
11
12     SOL.CUM(ELEMENT(n1).NODES) = SOL.CUM(ELEMENT(n1).NODES
        ) + Z;
13     SOL.COUNT(ELEMENT(n1).NODES) = SOL.COUNT(ELEMENT(n1).
        NODES) + 1;
14 end
15     SOL.AVG = SOL.CUM ./ SOL.COUNT;
16
17     NODE.(request) = SOL.AVG;
18 end

```

Stochastic fatigue model

```
1 function [xp] = stochastic()
2 % This function applies a stochastic correlation time
   parameter to the
3 % Paris-Erdogan model. It is based on the Yang-Manning's
   model.
4 Sz      = 0.932334471; % Estimated from experimental data
5 n       = 23;          % Data points
6 lambda  = sqrt((n-1)/2)*gamma((n-1)/2)/gamma(n/2);
7 mu_p    = normrnd(0,1);
8 Zp      = -lambda*mu_p*Sz;
9 xp      = 2^Zp;
10 end
```

Script to find intersection between two line segments

```

1 function [ Ipoint ] = fsintersect( Line1,Line2 )
2 %fsintersect: Intersection between two finite line segments.
3 %   fgeo_sintersect detects intersection points between two
   finite line
4 %   segments and returns the (x,y) coordinate of the
   intersection point. If
5 %   no intersection exists, the function returns a (NaN,NaN)
   point
6 %   coordinate.
7 %
8 % INPUTS:
9 %   Line1   : first line segment [xi,yi]
10 %   Line2   : second line segment [xi,yi]
11 %
12 % OUTPUT:
13 %   Ipoint : intersection point value [NaN,NaN] if none, [x,
   y] otherwise
14 %
15 % NOTES:
16 %   - Only 2D space lines supported.
17
18 % PROCESS: Defining segment starting points
19 p = Line1(1,:);
20 q = Line2(1,:);
21
22 % PROCESS: Defining segment vectors
23 r = Line1(2,:) - p;
24 s = Line2(2,:) - q;
25
26 % PROCESS: Defining distance vector
27 qp = q-p;
28
29 % PROCESS: Calculating parameters
30 t = (qp(1)*s(2)-qp(2)*s(1))/(r(1)*s(2)-r(2)*s(1));
31 u = (qp(1)*r(2)-qp(2)*r(1))/(r(1)*s(2)-r(2)*s(1));
32

```

```
33 % PROCESS: Finds if intersection exists
34 if (t >= 0) && (t <= 1) && (u >= 0) && (u <= 1)
35
36     Ipoint = p + t*r;
37     %Ipoint = q + u*s;
38
39 else
40
41     Ipoint = [NaN NaN];
42
43 end
44 end
```

Bibliography

- (2008). "Meshless methods: A review and computer implementation aspects". *Mathematics and Computers in Simulation* 79(3), 763–813. [Cited on page(s): [27](#)]
- Ahmed, A. (2009, April). "eXtended Finite Element Method(XFEM)-Modeling arbitrary discontinuities and Failure analysis". [Cited on page(s): [23](#)]
- Anderson, T. (2005). *"Fracture Mechanics: Fundamentals and Applications, Third Edition"*. Taylor & Francis. [Cited on page(s): [30](#), [33](#), [45](#), [47](#)]
- Asadpoure, A. and S. Mohammadi (2007). "Developing new enrichment functions for crack simulation in orthotropic media by the extended finite element method". *International Journal for Numerical Methods in Engineering* 69(10), 2150–2172. [Cited on page(s): [14](#), [19](#)]
- Banea, M. D. and L. F. M. da Silva (2009). "Adhesively bonded joints in composite materials: An overview". *Proceedings of the Institution of Mechanical Engineers, Part L: Journal of Materials: Design and Applications* 223(1), 1–18. [Cited on page(s): [9](#), [32](#), [57](#)]
- Belytschko, T. and T. Black (1999). "Elastic crack growth in finite elements with minimal remeshing". *International journal for numerical methods in engineering* 45(5), 601–620. [Cited on page(s): [4](#), [18](#), [20](#)]
- Biel, A. and U. Stigh (2007). "An analysis of the evaluation of the fracture energy using the DCB-specimen". *Archives of Mechanics* 59(4-5), 311–327. [Cited on page(s): [9](#), [32](#), [57](#)]
- Boeing Commercial Airplanes (2006). *AERO magazine* (24). [Cited on page(s): [2](#)]
- Cahill, L., S. Natarajan, S. Bordas, R. O'Higgins, and C. McCarthy (2014). "An experimental/numerical investigation into the main driving force for crack propagation in uni-directional fibre-reinforced composite laminae". *Composite Structures* 107, 119–130. [Cited on page(s): [13](#), [14](#)]
- Campilho, R., M. Banea, F. Chaves, and L. da Silva (2011). "eXtended Finite Element Method for fracture characterization of adhesive joints in pure mode I". *Computational Materials Science* 50(4), 1543 – 1549. Proceedings of the 19th International Workshop on Computational Mechanics of Materials. [Cited on page(s): [3](#), [13](#)]

- Campilho, R., M. Banea, A. Pinto, L. da Silva, and A. de Jesus (2011). "Strength prediction of single- and double-lap joints by standard and extended finite element modelling". *International Journal of Adhesion and Adhesives* 31(5), 363–372. [Cited on page(s): 2, 3, 13]
- Car, E., S. Oller, and E. Oñate (2000). "An anisotropic elastoplastic constitutive model for large strain analysis of fiber reinforced composite materials". *Computer Methods in Applied Mechanics and Engineering* 185(2), 245–277. [Cited on page(s): 42]
- Cherepanov, G. P. (1967). "Crack propagation in continuous media: PMM vol. 31, no. 3, 1967, pp. 476–488". *Journal of Applied Mathematics and Mechanics* 31(3), 503–512. [Cited on page(s): 28]
- de Borst, R. and J. J. Remmers (2006). "Computational modelling of delamination". *Composites Science and Technology* 66(6), 713–722. [Cited on page(s): 3, 13]
- Dolbow, J. (1999, December). "An Extended Finite Element Method with Discontinuous Enrichment for Applied Mechanics". Ph. D. thesis, Northwestern University. Field of Theoretical and Applied Mechanics. [Cited on page(s): 26]
- Dolbow, J., N. Moës, and T. Belytschko (2000). "Discontinuous enrichment in finite elements with a partition of unity method". *Finite Elements in Analysis and Design* 36(3–4), 235–260. Robert J. Melosh Medal Competition, Duke University, Durham NC, USA, March 1999. [Cited on page(s): 4]
- Fleming, M. (1997, June). "The Element-Free Galerkin Method for Fatigue and Quasi-static Fracture". Ph. D. thesis, Northwestern University. Field of Theoretical and Applied Mechanics. [Cited on page(s): 20]
- Fleming, M., Y. Chu, B. Moran, T. Belytschko, Y. Lu, and L. Gu (1997). "Enriched element-free Galerkin methods for crack tip fields". *International journal for numerical methods in engineering* 40(8), 1483–1504. [Cited on page(s): 20]
- Fries, T.-P. (2008). "A Corrected XFEM Approximation Without Problems in Blending Elements". *International Journal for Numerical Methods in Engineering* 75(10), 503–532. [Cited on page(s): 21, 24]
- Frostig, Y., O. T. Thomsen, and F. Mortensen (1999). "Analysis of adhesive-bonded joints, square-end, and spew-fillet-high-order theory approach". *Journal of engineering mechanics* 125(11), 1298–1307. [Cited on page(s): 2]
- Griffith, A. A. (1921, January). "The Phenomena of Rupture and Flow in Solids". *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 221(582-593), 163–198. [Cited on page(s): 28]

- Hart-Smith, L. (2002). "Adhesive bonding of composite structures: progress to date and some remaining challenges". *Journal of composites technology & research* 24(3), 133–151. [Cited on page(s): 2]
- Hua, C. (1990). "An inverse transformation for quadrilateral isoparametric elements: Analysis and application". *Finite Elements in Analysis and Design* 7(2), 159–166. [Cited on page(s): 27]
- Irwin, G. R. (1957). "Analysis of stresses and strains near the end of a crack traversing a plate". *Journal of Applied Mechanics* 24, 361–364. [Cited on page(s): 28]
- König, M., R. Krüger, K. Kussmaul, M. Von Alberti, and M. Gädke (1997). "Characterizing static and fatigue interlaminar fracture behavior of a first generation graphite/epoxy composite". In *Composite Materials: Testing and Design, Thirteenth Volume*. ASTM International. [Cited on page(s): 38, 42, 43, 44, 57]
- Krueger, R. (2010). "Development of a benchmark example for delamination fatigue growth prediction". Technical Report NIA Report No. 2010-04, National Institute of Aerospace. [Cited on page(s): 33, 42, 43, 46, 57]
- Kuna, M. (2013). *"Finite Elements in Fracture Mechanics"*, Volume 201. Springer. [Cited on page(s): 6, 14, 29]
- Li, W., T. Sakai, Q. Li, and P. Wang (2011). Statistical analysis of fatigue crack growth behavior for grade b cast steel. *Materials & Design* 32(3), 1262–1272. [Cited on page(s): 37]
- Logan, D. L. (2011). *"A first course in the finite element method"*. Cengage Learning. [Cited on page(s): 17, 18]
- Lord, S. J. and M. F. Ngah (2005). "On the Modelling of Impact Damage Growth in Composite Structures". In *European Conference for Aerospace Sciences (EUCASS)*. [Cited on page(s): 1]
- Moës, N., J. Dolbow, and T. Belytschko (1999). "A finite element method for crack growth without remeshing". *International Journal for Numerical Methods in Engineering* 46(1), 131–150. [Cited on page(s): 4, 5]
- Mohammadi, S. (2008). *"Extended finite element method: for fracture analysis of structures"*. John Wiley & Sons, Ltd. [Cited on page(s): 18, 19]
- Mohammadi, S. (2012). *"XFEM Fracture Analysis of Composites"*. John Wiley & Sons, Ltd. [Cited on page(s): 18]
- Motamedi, D., A. Milani, M. Komeili, M. Bureau, F. Thibault, and D. Trudel-Boucher (2014). "A stochastic XFEM model to study delamination in PPS/glass UD composites: effect of uncertain fracture properties". *Applied Composite Materials* 21(2),

- 341–358. [Cited on page(s): [3](#)]
- Motamedi, D., A. S. Milani, et al. (2013). "3D Nonlinear XFEM Simulation of Delamination in Unidirectional Composite Laminates: A Sensitivity Analysis of Modeling Parameters". *Open Journal of Composite Materials* 3(04), 113. [Cited on page(s): [3](#), [13](#)]
- Nairn, J. A. (2000). "Energy release rate analysis for adhesive and laminate double cantilever beam specimens emphasizing the effect of residual stresses". *International Journal of Adhesion and Adhesives* 20(1), 59–70. [Cited on page(s): [33](#)]
- Nishioka, T. and S. Atluri (1983). "A numerical study of the use of path independent integrals in elasto-dynamic crack propagation". *Engineering Fracture Mechanics* 18(1), 23–33. [Cited on page(s): [36](#)]
- Nuismer, R. (1975). "An energy release rate criterion for mixed mode fracture". *International journal of fracture* 11(2), 245–250. [Cited on page(s): [36](#)]
- Osher, S. and J. A. Sethian (1988, November). "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations". *Journal of Computational Physics* 79(1), 12–49. [Cited on page(s): [22](#)]
- Paris, P. and F. Erdogan (1960). "A Critical Analysis of Crack Propagation Laws". *Journal of Basic Engineering* 85(4), 528–533. [Cited on page(s): [37](#)]
- Paris, P. C., M. P. Gomez, and W. E. Anderson (1961). "A rational analytic theory of fatigue". *The trend in engineering* 13(1), 9–14. [Cited on page(s): [37](#)]
- Pascoe, J., R. Alderliesten, and R. Benedictus (2013). "Methods for the prediction of fatigue delamination growth in composites and adhesive bonds – A critical review". *Engineering Fracture Mechanics* 112–113, 72–96. [Cited on page(s): [13](#), [37](#)]
- Pascoe, J., R. Alderliesten, and R. Benedictus (2016). "Characterising resistance to fatigue crack growth in adhesive bonds by measuring release of strain energy". *Procedia Structural Integrity* 2, 80–87. [Cited on page(s): [37](#)]
- Perez, N. (2004). *"Fracture Mechanics"*. Mathematics and Its Applications Series. Springer. [Cited on page(s): [30](#), [31](#)]
- Phu, N. V. (2005, November). "An Object Oriented Approach to the Extended Finite Element Method with Application to Fracture Mechanics". [Cited on page(s): [26](#)]
- Prasad, M. S., C. Venkatesha, T. Jayaraju, et al. (2011). "Experimental methods of determining fracture toughness of fiber reinforced polymer composites under various loading conditions". *Journal of Minerals and Materials Characterization and Engineering* 10(13), 1263–1275. [Cited on page(s): [33](#)]

- Rice, J. (1968). "A path independent integral and the approximate analysis of strain concentration by notches and cracks". *Journal of applied mechanics* 35(2), 379–386. [Cited on page(s): 28]
- Roderick, G., R. Everett, and J. Crews (1975). "Debond propagation in composite-reinforced metals". In *Fatigue of Composite Materials*. ASTM International. [Cited on page(s): 37]
- Shih, C. F., B. Moran, and T. Nakamura (1986). "Energy release rate along a three-dimensional crack front in a thermally stressed body". *International Journal of Fracture* 30(2), 79–102. [Cited on page(s): 29]
- Sosa, J. C. and N. Karapurath (2012). "Delamination modelling of {GLARE} using the extended finite element method". *Composites Science and Technology* 72(7), 788–791. [Cited on page(s): 3, 14]
- Stolarska, M., D. L. Chopp, N. MoÑs, and T. Belytschko (2001). "Modelling crack growth by level sets in the extended finite element method". *International Journal for Numerical Methods in Engineering* 51(8), 943–960. [Cited on page(s): 22, 23]
- Sukumar, N., D. Chopp, N. Moës, and T. Belytschko (2001). "Modeling holes and inclusions by level sets in the extended finite-element method". *Computer Methods in Applied Mechanics and Engineering* 190(46–47), 6183–6200. [Cited on page(s): 22]
- Sukumar, N., Z. Y. Huang, J.-H. Prévost, and Z. Suo (2004). "Partition of unity enrichment for bimaterial interface cracks". *International Journal for Numerical Methods in Engineering* 59(8), 1075–1102. [Cited on page(s): 14, 18]
- Sukumar, N. and J.-H. Prévost (2003). "Modeling quasi-static crack growth with the extended finite element method Part I: Computer implementation". *International journal of solids and structures* 40(26), 7513–7537. [Cited on page(s): 22, 27]
- Tong, L. and G. Steven (1999, Apr). *Analysis and design of structural bonded joints*. Kluwer Academic Publishers, Hingham, MA (US). [Cited on page(s): 1]
- Williams, M. L. (1957). "On the stress distribution at the base of a stationary crack". *Journal of Applied Mechanics* 24, 109–114. [Cited on page(s): 20, 31]
- Wilson, E., R. Taylor, W. Doherty, and J. Ghaboussi (1973). "Incompatible displacement models". In *Numerical and computer methods in structural mechanics*, Volume 43. Academic Press. [Cited on page(s): 18, 57]
- Wu, W. and C. Ni (2003). "A study of stochastic fatigue crack growth modeling through experimental data". *Probabilistic Engineering Mechanics* 18(2), 107–118. [Cited on page(s): 3]

- Xiao, Q., B. Karihaloo, and X. Liu (2004). "Direct determination of SIF and higher order terms of mixed mode cracks by a hybrid crack element". *International Journal of Fracture* 125(3), 207–225. [Cited on page(s): 31]
- Xiao, Q.-Z. and B. L. Karihaloo (2006). "Improving the accuracy of XFEM crack tip fields using higher order quadrature and statically admissible stress recovery". *International Journal for Numerical Methods in Engineering* 66(9), 1378–1410. [Cited on page(s): 20]
- Yang, J. and S. Manning (1990). Stochastic crack growth analysis methodologies for metallic structures. *Engineering Fracture Mechanics* 37(5), 1105–1124. [Cited on page(s): 37]
- Ye, C., J. Shi, and G. J. Cheng (2012). "An eXtended Finite Element Method (XFEM) study on the effect of reinforcing particles on the crack propagation behavior in a metal-matrix composite". *International Journal of Fatigue* 44, 151–156. [Cited on page(s): 14]

Vita

Sergio Candelario

Sergio Candelario was born in San Juan, Puerto Rico on June 24, 1987 but raised in Caguas, Puerto Rico. After completing his high school education, he was accepted at University of Puerto Rico Mayagüez campus (UPRM) mechanical engineering program.

In May of 2011, he graduated obtaining a Bachelor Science degree in Mechanical Engineering with Magna Cum Laude. He is a member of the Engineering Society Tau-Beta-Pi Alpha Chapter of Puerto Rico.

His interest in structural mechanics motivated him to join Dr. Goyal's research group in 2011 pursuing the masters degree in Mechanical Engineering at UPRM under the guidance of Dr. Vijay K. Goyal but later moved to work with Dr. David Serrano for the completion of his dissertation.

After two years into the master's program, he moved to Caguas to work as a validation scientist contractor at Janssen in Gurabo, Puerto Rico, a pharmaceutical manufacturing company. He has continuously worked in the technical services area focused in new product introduction and has worked in the characterization, registration and validation manufacturing process of several products.

He has been the recipient of the Departmental's Graduate Teaching Assistant for Mechanical Engineering undergraduate laboratories.

He expects to complete his M.S. degree on May of 2017.