An automated method for the FE analysis of 3D cracks under mixed mode loading

Supervisor: Prof. Mario Guagliano
Student: Dimitar Danov
Matr.: 722574

Academic Year 2010 – 2011
I would like to dedicate this work to my parents,

Neda and Atanas

for their boundless patience and incredible support!
Acknowledgements

I would like to thank my supervisor Prof. Guaglinao, for his trust and for giving me both freedom and opportunity to explore fields, which I am interested in.
## Contents

1 Introduction

1.1 Previous development and foundation of the present work

1.1.1 Background

1.1.2 Present work

1.2 Programming side

1.2.1 Abaqus structure and script execution

1.2.2 Python and Abaqus

1.2.3 On programming style

1.3 Motivation

1.3.1 Automation

1.3.2 Post-processing

1.3.3 Visualization

1.4 Significance of the work

1.4.1 Modeling

1.4.2 Automation

1.4.3 Post-processing

1.4.4 Scalability

1.4.5 Knowledge base

2 Theoretical overview

2.1 Introduction

2.2 Fracture mechanics overview

2.2.1 LEFM scope

2.2.2 Stress intensity factors $K$

2.2.3 Energy approach to Fracture Mechanics

2.2.4 Elliptic cracks

2.3 LEFM analysis with FEM

2.3.1 Special crack tip elements

2.3.2 Calculation of the stress intensity factors

2.4 XFEM

2.4.1 Introduction

2.4.2 XFEM concepts

2.4.3 Partition of unity based methods

2.4.4 XFEM formulation

2.4.5 XFEM crack definition in Abaqus

2.4.6 Crack analysis with XFEM

2.5 Fracture mechanics software

2.5.1 Zencrack
## Methodology

### 3.1 Introduction

### 3.2 Description of the model types

- **3.2.1 FEM – crackNormal model type**
- **3.2.2 Model types for XFEM analysis**
- **3.2.3 XFEM simple model type**
- **3.2.4 XFEM crackPartition model type**
- **3.2.5 XFEM multiplePartitions model type**

### 3.3 Visualization Odb

- **3.3.1 Node data**
- **3.3.2 Element data**
- **3.3.3 Field output data**

### 3.4 GUI and Abaqus integration

- **3.4.1 First dialog box**
- **3.4.2 Second dialog box**

### 3.5 Organization of the application

- **3.5.1 Structure by function**
- **3.5.2 Directory structure and modules**

### 3.6 Description of classes

- **3.6.1 Classes interaction**
- **3.6.2 DataStr classes**
- **3.6.3 Model database classes**
- **3.6.4 ReadOdb() class**
- **3.6.5 PersistentData() class**
- **3.6.6 DbDataStr() class**
- **3.6.7 AnalyticalData classes**
- **3.6.8 XYPlotDataFromDbEntry() class**
- **3.6.9 VisualizationOdbFromDbEntry() class**
- **3.6.10 GUI classes**
- **3.6.11 CreateID function family**
- **3.6.12 Execute gui commands functions**
- **3.6.13 Main loop**

## Results

### 4.1 Introduction

### 4.2 Procedure

### 4.3 Delimitations

### 4.4 Element type comparison

### 4.5 Analysis of the influence of the cylinder dimensions

### 4.6 Mesh convergence analysis

- **4.6.1 Mesh convergence analysis with quadratic reduced integration elements**
- **4.6.2 Mesh convergence analysis with linear reduced integration elements**

### 4.7 Comparison between mesh transformations
4.7.1 Comparison between *elliptic* and *simpleScale* mesh transformations ........................................... 95
4.7.2 *advancedScale* mesh transformation ................................................................. 102
4.8 XFEM results ................................................................................................................. 103
  4.8.1 Mesh and singularity radius convergence study ..................................................... 103
  4.8.2 Comparison of the values and errors of the calculated stress intensity factors by *XFEM* .......................................................... 111
  4.8.3 Comparison between *FEM* and *XFEM* results ................................................. 114
4.9 Visualization of the stress intensity factors ................................................................. 119

5 Conclusion ......................................................................................................................... 123
  5.1 Introduction .................................................................................................................... 123
  5.2 Summary of results ........................................................................................................ 123
  5.3 Implications for practice and recommendations .......................................................... 124
  5.4 Implications for further development .......................................................................... 124
    5.4.1 Modeling automation ............................................................................................ 124
    5.4.2 Results processing and optimization ..................................................................... 124
    5.4.3 Other functionality ................................................................................................. 124
  5.5 Conclusion ..................................................................................................................... 125
List of Figures

1.1 Execution of script commands .................................. 2

2.1 Polar coordinate system at the crack tip ....................... 7
2.2 The three load type modes of a crack ......................... 8
2.3 Crack with blunted tip ...................................... 11
2.4 Crack in biaxially loaded plate .............................. 11
2.5 Stress distribution in the vicinity of the crack tip .......... 13
2.6 Visualization of the yield stress boundaries of the crack tip plastic zone ........................................ 13
2.7 Total energy of a plate as a function of the crack length a . 14
2.8 Potential energy release rate for EPFM and LEFM for different loads ........................................ 15
2.9 Elliptic crack types ....................................... 16
2.10 Elliptic crack plane ..................................... 16
2.11 Orientation of an elliptic crack ............................ 17
2.12 Special crack tip elements ................................ 19
2.13 Zones of elements around a crack tip ....................... 21
2.14 Standard, enriched and blending elements in a domain .... 23
2.15 Level set functions for a flat crack ......................... 24
2.16 Crack flanks and crack tip ................................ 25
2.17 Crack domain and crack geometry .......................... 26

3.1 Cross section of the crackNormal model ................... 29
3.2 crackNormal model ....................................... 32
3.3 WEDGE elements of the inner cylinder of the crackNormal model ........................................ 34
3.4 crack domain of the crackPartition model ................. 36
3.5 crack domain of the multiplePartitions model ............ 38
3.6 Visualization output database ................................ 39
3.7 First dialog box of the program user interface .......... 41
3.8 Second dialog box of the program user interface ......... 43
3.9 Classes interaction ...................................... 46

4.1 Comparison of values for $K_I$ obtained for crackNormal model type with different element types and elliptic transformation . 84
4.2 Comparison of values for $K_{II}$ obtained for crackNormal model type with different element types and elliptic transformation . 84
4.3 Comparison of values for $K_I$ obtained for crackNormal model type with different element types and elliptic transformation . 85
4.4 Comparison of errors for $K_I$ obtained for crackNormal model type with different element types and elliptic transformation ... 85
4.5 Comparison of errors for $K_{II}$ obtained for crackNormal model type with different element types and elliptic transformation ... 86
4.6 Comparison of errors for $K_{III}$ obtained for crackNormal model type with different element types and elliptic transformation ... 86
4.7 Convergence study for cylinder dimensions against the maximum errors for $K_I$ ...................................................... 87
4.8 Convergence study for cylinder dimensions against the maximum errors for $K_{II}$ ...................................................... 88
4.9 Convergence study for cylinder dimensions against the maximum errors for $K_{III}$ ...................................................... 88
4.10 Comparison of errors for $K_I$ along the crack front for different cylinder dimensions ............................................ 89
4.11 Comparison of errors for $K_{II}$ along the crack front for different cylinder dimensions ............................................. 89
4.12 Comparison of errors for $K_{III}$ along the crack front for different cylinder dimensions ............................................. 90
4.13 Comparison of errors for $K_I$ along the crack front for different mesh densities of quadratic reduced integration elements .... 91
4.14 Comparison of errors for $K_{II}$ along the crack front for different mesh densities of quadratic reduced integration elements .... 91
4.15 Comparison of errors for $K_{III}$ along the crack front for different mesh densities of quadratic reduced integration elements .... 92
4.16 Comparison of errors for $K_I$ along the crack front for different mesh densities of linear reduced integration elements ........ 94
4.17 Comparison of errors for $K_{II}$ along the crack front for different mesh densities of linear reduced integration elements ........ 94
4.18 Comparison of errors for $K_{III}$ along the crack front for different mesh densities of linear reduced integration elements ........ 95
4.19 Comparison of errors for $K_I$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 3 ................................................................. 96
4.20 Comparison of errors for $K_{II}$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 3 ................................................................. 96
4.21 Comparison of errors for $K_{III}$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 3 ................................................................. 97
4.22 Comparison of errors for $K_I$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 5 ................................................................. 97
4.23 Comparison of errors for $K_{II}$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 5 ................................................................. 98
4.24 Comparison of errors for $K_{III}$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 5 ................................................................. 98
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.25</td>
<td>Comparison of errors for $K_I$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 10</td>
</tr>
<tr>
<td>4.26</td>
<td>Comparison of errors for $K_{II}$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 10</td>
</tr>
<tr>
<td>4.27</td>
<td>Comparison of errors for $K_{III}$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 10</td>
</tr>
<tr>
<td>4.28</td>
<td>Comparison of the maximum errors for $K_I$ for elliptic and simpleScale mesh transformations for crack with aspect ratios of 3, 5 and 10</td>
</tr>
<tr>
<td>4.29</td>
<td>Comparison of the maximum errors for $K_{II}$ for elliptic and simpleScale mesh transformations for crack with aspect ratios of 3, 5 and 10</td>
</tr>
<tr>
<td>4.30</td>
<td>Comparison of the maximum errors for $K_{III}$ for elliptic and simpleScale mesh transformations for crack with aspect ratios of 3, 5 and 10</td>
</tr>
<tr>
<td>4.31</td>
<td>Convergence study for crackPartition XFEM model for $K_I$ stress intensity factor</td>
</tr>
<tr>
<td>4.32</td>
<td>Convergence study for crackPartition XFEM model for $K_{II}$ stress intensity factor</td>
</tr>
<tr>
<td>4.33</td>
<td>Convergence study for crackPartition XFEM model for $K_{III}$ stress intensity factor</td>
</tr>
<tr>
<td>4.34</td>
<td>Convergence study for multiplePartitions XFEM model for $K_I$ stress intensity factor</td>
</tr>
<tr>
<td>4.35</td>
<td>Convergence study for multiplePartitions XFEM model for $K_{II}$ stress intensity factor</td>
</tr>
<tr>
<td>4.36</td>
<td>Convergence study for multiplePartitions XFEM model for $K_{III}$ stress intensity factor</td>
</tr>
<tr>
<td>4.37</td>
<td>Mesh and singularity radius convergence for $K_I$</td>
</tr>
<tr>
<td>4.38</td>
<td>Mesh and singularity radius convergence for $K_{II}$</td>
</tr>
<tr>
<td>4.39</td>
<td>Mesh and singularity radius convergence for $K_{III}$</td>
</tr>
<tr>
<td>4.40</td>
<td>Comparison of the calculated values for $K_I$ along the crack front for the different XFEM model types</td>
</tr>
<tr>
<td>4.41</td>
<td>Comparison of the calculated values for $K_{II}$ along the crack front for the different XFEM model types</td>
</tr>
<tr>
<td>4.42</td>
<td>Comparison of the calculated values for $K_{III}$ along the crack front for the different XFEM model types</td>
</tr>
<tr>
<td>4.43</td>
<td>Errors of the calculated values for $K_I$ along the crack front for the different XFEM model types</td>
</tr>
<tr>
<td>4.44</td>
<td>Errors of the calculated values for $K_{II}$ along the crack front for the different XFEM model types</td>
</tr>
<tr>
<td>4.45</td>
<td>Errors of the calculated values for $K_{III}$ along the crack front for the different XFEM model types</td>
</tr>
<tr>
<td>4.46</td>
<td>Comparison between the calculated values for $K_I$ by FEM and XFEM along the crack front for crack with aspect ratio of 3</td>
</tr>
<tr>
<td>4.47</td>
<td>Comparison between the calculated values for $K_{II}$ by FEM and XFEM along the crack front for crack with aspect ratio of 3</td>
</tr>
</tbody>
</table>
4.48 Comparison between the calculated errors for $K_{III}$ by FEM and XFEM along the crack front for crack with aspect ratio of 3 . . . 116
4.49 Comparison between the calculated values for $K_I$ by FEM and XFEM along the crack front for crack with aspect ratio of 5 . . . 116
4.50 Comparison between the calculated values for $K_{III}$ by FEM and XFEM along the crack front for crack with aspect ratio of 3 . . . 117
4.51 Comparison between the calculated errors for $K_{III}$ by FEM and XFEM along the crack front for crack with aspect ratio of 5 . . . 117
4.52 Comparison between the calculated values for $K_I$ by FEM and XFEM along the crack front for crack with aspect ratio of 10 . . 118
4.53 Comparison between the calculated values for $K_{III}$ by FEM and XFEM along the crack front for crack with aspect ratio of 10 . . 118
4.54 Comparison between the calculated errors for $K_{III}$ by FEM and XFEM along the crack front for crack with aspect ratio of 10 . . 119
4.55 Visualization of the 1309848923dot31 FEM model with crack aspect ratio 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120
4.56 Visualization of the 1309869569dot74 XFEM model with crack aspect ratio 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120
4.57 Visualization of the 1310240764dot87 XFEM model with crack aspect ratio 5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
4.58 Visualization of the 1310240980dot74 XFEM model with crack aspect ratio 10 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
4.59 Visualization of the 1309907898dot52 XFEM model with crack aspect ratio 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 122
# List of Tables

3.1 GUI tree abbreviation key ........................................ 42
4.1 Models included in the element type study ................... 84
4.2 Models in figures 4.10, 4.11 and 4.12 .......................... 89
4.3 Models with quadratic reduced integration elements included in the mesh convergence study .......................... 91
4.4 Models with linear reduced integration elements included in the mesh convergence study .......................... 93
4.5 Models included in the comparison of mesh transformations ... 100
4.6 Models of type crackPartition included in the convergence study 105
4.7 Models of type multiplePartitions included in the convergence study ........................................ 106
4.8 Models of type simple included in the convergence study ..... 109
4.9 Models included in the comparison of the accuracy of the different XFEM model types .......................... 112
4.10 FEM models included in the comparison of the accuracy with XFEM models ........................................ 115
4.11 XFEM models included in the comparison of the accuracy with FEM models ........................................ 116
4.12 Visualization of models ............................................ 119
Abstract

Analysis of elliptic cracks under mixed mode loading is a challenging aspect of the design and life assessment of mechanical components. Nevertheless, perpetually increasing requirements in terms of safety and performance, demand an efficient procedure for accurate crack analysis.

A computer program is developed in the scope of the current project, aiming to address both efficiency and accuracy of analysis. The program is a plug-in to Abaqus finite element analysis program and automates significantly the modeling and analysis of stress intensity factors of elliptic cracks.

Elliptic cracks are analyzed with FEM and XFEM, and several mesh configurations are compared. A visualization technique is proposed for representation of stress intensity factors and results from the analyses are stored in custom database, providing a foundation for accumulating a large knowledge base including all analyzed crack configurations.

Results from the performed analyses prove that depending on the analysis type and mesh design, evaluated stress intensity factors may vary significantly.
Chapter 1

Introduction

1.1 Previous development and foundation of the present work

1.1.1 Background

Background of the current project is ref [7], which is focused on evaluation of stress intensity factors of elliptic cracks by FEM. For the purpose a model with the shape of a cylinder is introduced, which accommodates the analyzed crack. Cracks with axes aspect ratio between 0.01 and 100 are analyzed and the results are compared with analytical solutions. The geometry of the analyzed cracks is obtained from a circular shape by means of either elliptic or linear scale transformation. Comparison is performed between the two transformations and both linear and quadratic finite elements are used. Results prove that the elliptic transformation is superior to the linear scale one. In addition to stress intensity factors, the work also derives the strain energy factor for the analyzed cracks from which the crack propagation direction is obtained.

In addition, the methodology presented in ref [7], has been utilized in ref [13] for analysis of sub-surface cracks in hypoid gears and in ref [12] for analysis of sub-surface cracks in railway wheels.

1.1.2 Present work

The present work recreates and builds upon the developments in ref [7]. The model introduced by Guagliano et al corresponds to the crackNormal model, in terms of geometric features, also the transformations elliptic and simpleScale correspond to the transformations used in ref [7]. In addition, strain energy density is not considered in this project.

The present project, however, extends ref [7] with the following key developments:

- analysis of elliptic cracks with XFEM;
- automated history output extraction from the Abaqus output database;
- storing stress intensity factors data in a custom shelve database;
• automated report generation;
• visualization of the stress intensity factors;
• integrated into Abaqus browser with tree representation of the custom shelf database;
• functionality for analysis of surface and edge elliptic cracks.

1.2 Programming side

1.2.1 Abaqus structure and script execution

Abaqus has a modular structure, with components addressing various fields and providing complementary functionality. Abaqus/CAE provides the scripting extensions and graphical user interface for Abaqus/Standard, Abaqus/Explicit, Abaqus/CFD or Abaqus/Design. It is therefore an optional component. All of the analyses in the current project are performed with Abaqus/Standard.

The structure and workflow of execution of script commands by Abaqus is illustrated in figure 1.1. First, the script commands are executed by Abaqus/CAE, which generates the model database. When the model database is submitted for analysis Abaqus/CAE generates an input file which is passed to Abaqus/Standard, which executes the analysis and generates an output database with the results.

It follows that there are two way to approach an Abaqus analysis, one with Abaqus/CAE, and other without, by manually generating the input file. This applies to scripting as well, one way is to use the Abaqus scripting API and the other is to generate an input file by string manipulation. For the project scripts are designed to work with the Abaqus scripting API. This has the considerable advantage that, commands, in their majority, are of higher level of abstraction. The drawback is that commands are translated multiple times until the input file is generated and this makes them implicit. Interacting with the input file directly, eliminates that drawback. However, interaction is at a very low level and is mostly reduced to counting node and element labels.

![Figure 1.1: Execution of script commands](image)

1.2.2 Python and Abaqus

Python ref [20] and ref [30] is the scripting language used in Abaqus. It is used both for kernel commands and GUI. In addition to the standard features of Python, the implementation in Abaqus incorporates the API, which are the
particular to Abaqus commands. The graphical user interface of Abaqus/CAE is a custom implementation of the FOXtoolkit in Python. The graphical interface of Abaqus/CAE is can be customized in a variety of ways from building dialog boxes to modifying the default user interface or creating a custom from scratch.

1.2.3 On programming style

At a certain point during the development of the program for the project, it the issue of maintainability of the code has surfaced. According to ref [21] time spent reading code exceeds the time writing code nine fold. Therefore, maintaining a certain coding style is crucial component of programming and has been consciously reinforced. It took several trials, each iteration including a complete redesign to develop the program at its present version. The main features from a programming point of view include:

- scalability,
- object oriented,
- multiple inheritance,
- encapsulation,

Little of the functionality of the current version, would have been achieved unless a particular effort has been made during its development to adhere to the programming practices stated in ref [21].

1.3 Motivation

Abaqus native tools for fracture analysis, as of time of writing, require significant effort. Therefore, an automated framework for modeling, analysis and post-processing of the results is welcome. Nevertheless, fracture mechanics problems are becoming increasingly important and to make matters worse, a fracture mechanics analysis involves numerous parameters, which means that numerous analysis iterations must be performed.

1.3.1 Automation

The developed program in the project addresses these issues by automating the modeling a cylindric model with embedded elliptic crack. The model, can be utilized either as a submodel for a larger analysis, or it can be used in a convergence study to determine the optimal parameters and expected accuracy for a larger model including a crack definition.

1.3.2 Post-processing

Post-processing of stress intensity factors may also prove challenging, as at present it requires a significant interaction between the analyst and Abaqus. This issue is also addressed by automated extraction of the stress intensity factors from the Abaqus output database and creating plot data for convenience.
The post-processing is taken even a step further by storing crack analysis results in a custom database to create a knowledge base of all of the performed analyses.

1.3.3 Visualization

Visualization is introduced to improve the understanding of the analyzed crack. It represents stress intensity factors in a three dimensional space mapped to the crack geometry. The technique can also be utilized as a diagnostic tool for XFEM to improve understanding and thus quality of the analysis.

1.4 Significance of the work

The project and framework significance can be evaluated in the following aspects:

- modeling,
- automation,
- post-processing,
- scalability,
- knowledge base.

1.4.1 Modeling

The modeling aspect contribution is mostly in regard of time-saving and consistency. The framework is capable of generating several model databases by given parameters. In addition, FEM models, including mesh transformations cannot be created without scripting.

1.4.2 Automation

The automation aspect contribution extends beyond time-saving. It is also enabling to evaluate multiple scenarios, a challenging and error prone task if performed manually. For instance, creating a plot for the stress intensity factors may require selecting manually up to or over 300 points in succession for some XFEM analyses performed in chapter 4. The framework does this automatically, without user input.

1.4.3 Post-processing

The chief contribution in the post-processing aspect is the visualization technique, utilized for representation of the stress intensity factors and as a diagnostic tool.
1.4.4 Scalability

This scalability aspect is a measure of the framework ability to be extended, either by new model types or further post-processing techniques. At present the framework can create models for quarter and semi-elliptic cracks for all model types, though the project scope is limited only to embedded.

1.4.5 Knowledge base

The knowledge base is one of the most significant features of the framework. It provides access through a browser to the custom database with all the calculated cracks. At the moment of writing the shelf database has 484 entries, or in other words, the knowledge base contains input parameters and results for stress intensity factors for 484 different cracks, which can be easily accessed. The number can increase in the future and the framework can be extended by optimization algorithms.
Chapter 2

Theoretical overview

2.1 Introduction

The theoretical yield strength of materials is of one to three orders of magnitude higher than the one observed in practice. The discrepancy is attributed to imperfections of the structure of the material, resulting in stress concentrations, which drastically reduce the material properties. Furthermore, such defects – cracks may occur when a component is in service. Cracks may either stay dormant and not influence the function of the component, or may grow reaching a critical size and resulting in fast and often catastrophic fracture of the component. A list of some of the most prominent disasters, due to fracture are listed in ref [2].

Therefore, to prevent future catastrophic failures, it is crucial to evaluate the strength of a critical component and estimate its remaining service life and design future components to be damage tolerant.

2.2 Fracture mechanics overview

2.2.1 LEFM scope

Whether LEFM is applicable to a specific problem depends on the extent of the applied stress and the local stress field in the vicinity of the crack. For instance, if a sharp crack is considered, the linear stress at the crack tip is singular and therefore, the material will yield. The size of this plastic zone determines the applicability of LEFM i.e. conditions must be essentially linear. It is applicable mostly to high strength materials. In case stresses are high and the yielding in the vicinity of the crack zone cannot be neglected, or the material is relatively more ductile, the problem should be addressed with EPFM.

2.2.2 Stress intensity factors $K$

The stress intensity factors define the stress field in the vicinity of the crack tip. The stress intensity factors are valid only for LEFM. The stress field is defined in a polar coordinate system with coordinates $r$ and $\theta$ at the crack tip, as shown
in figure 2.1. The stress is defined by:

$$\sigma_{ij} = \frac{K}{\sqrt{2\pi r}} f_{ij}(\theta) + \ldots,$$

where $K$ is the stress intensity factor, which defines the magnitude of the stress. $K$ is defined by: $K = \sigma \sqrt{\pi a} f(a/W)$, where:

- $a$ is the half length of the crack,
- $f(a/W)$ is a dimensionless coefficient, depending on the crack geometry,
- $\sigma$ is the remotely applied stress.

The so defined $\sigma_{ij}$ becomes infinite when $r \to 0$, thus there is singularity in the stress field when plasticity is not considered. The $\sigma_{ij}$ also tends to 0, when $r \to \infty$. Therefore, the equation is valid only for $r << a$.

![Figure 2.1: Polar coordinate system at the crack tip](image)

**Modes of loading**

Stresses in the vicinity of the crack can be decomposed into a combination of modes of crack surface displacements, shown in figure 2.2:

- **Mode I** opening mode
- **Mode II** sliding mode
- **Mode III** tearing mode

A stress intensity factor corresponds to each mode of crack surface displacement, $K_I$, $K_{II}$ and $K_{III}$. This decomposition allows to estimate an arbitrary load conditions around a crack with only three parameters.

**Airy stress functions**

A function describing the elastic stress field must fulfill both the equilibrium and compatibility of strain requirements. Let such a function be $\Phi(x, y)$, an Airy stress function for a two dimensional problem.

The equilibrium equations are satisfied if:
The compatibility of strain is satisfied if:

$$\frac{\partial^4 \Phi}{\partial x^4} + 2 \frac{\partial^4 \Phi}{\partial x^2 \partial y^2} + \frac{\partial^4 \Phi}{\partial y^4} = 0$$

The compatibility of strain is satisfied if:

$$\frac{\partial^4 \Phi}{\partial x^4} + 2 \frac{\partial^4 \Phi}{\partial x^2 \partial y^2} + \frac{\partial^4 \Phi}{\partial y^4} = 0$$

**Westergaard stress equations**

A Westergaard stress function relates the geometry, stress intensity factors, stress and displacement. It is a specific type of an Airy stress function $\Phi$, defined by a complex stress function $\phi(z)$. For $\phi(z)$ is assumed that its derivative and first $\bar{\phi}(z)$ and second $\bar{\bar{\phi}}(z)$ order integrals exist. Therefore:

$$\Phi = \text{Re}(\phi(z)) + y.\text{Im}(\phi(z))$$

where $z = x + i.y$. Thus defined the function $\Phi$, leads to the Cauchy-Riemann conditions:

$$\frac{\partial \text{Re}(\Phi)}{\partial x} = \frac{\partial \text{Im}(\Phi)}{\partial y} = \frac{\text{Re}(\partial \Phi)}{\partial z} \quad (2.4)$$

$$\frac{\partial \text{Im}(\Phi)}{\partial x} = -\frac{\partial \text{Re}(\Phi)}{\partial y} = \frac{\text{Im}(\partial \Phi)}{\partial z} \quad (2.5)$$

Using equations 2.3 for the stresses, we obtain:

$$\sigma_x = \text{Re}(\phi(z)) - \text{Im}(\phi'(z)) \quad (2.6)$$

$$\sigma_y = \text{Re}(\phi(z)) + y.\text{Im}(\phi'(z)) \quad (2.7)$$

$$\tau_{xy} = -y.\text{Re}(\phi'(z)) \quad (2.8)$$
The \( \phi(z) \) function is a generalization and for each particular case it should be defined to correspond to the boundary conditions. The Westergaard complex stress function limits the problems to \( \sigma_x = \sigma_y \) and \( \tau_{xy} = 0 \).

**Biaxially loaded plate**

The case of a crack in a biaxially loaded infinite plate is shown in figure 2.4. Load \( \sigma \) is applied in the plate both directions along \( X \) axis and \( Y \) axis. The complex function for this case is:

\[
\phi(z) = \frac{\sigma}{\sqrt{1 - a^2/z^2}}
\]

Results for the stresses are obtained by substituting \( \phi(z) \) in equations 2.8 and considering the following cases:

- \( y = 0 \) and \( |x| < a \), which corresponds to the stresses on the crack flanks,

\[
\phi(z) = \phi(x) = \frac{-i\sigma}{\sqrt{a^2/x^2 - 1}}
\]

and therefore, \( \phi(z) \) is purely imaginary, resulting in \( \sigma_y = 0 \).

- \( x \to \infty \) and/or \( y \to \infty \) results in \( \phi(z) = \sigma \).

- \( x = \pm a \) and \( y = 0 \), which corresponds to the crack tips, \( \phi(z) \to \infty \).

For the derivation of the stress intensity factor \( K_I \) it is convenient to move the origin of the coordinate system to coincide with the crack tip. The \( \phi(z) \) then becomes:

\[
\phi(\eta) = \frac{\sigma}{\sqrt{1 - (a + \eta)^2/a^2}} = \frac{\sigma(a + \eta)}{\sqrt{(a + \eta)^2 - a^2}}
\]

where \( \eta = z - a \). Then \( \phi(\eta) \) can be approximated as:

\[
\phi(\eta) \approx -\frac{\sigma a}{\sqrt{2\pi\eta}} = \sigma \sqrt{\frac{a}{2\eta}} e^{-\frac{1}{2}i\theta}
\]

In polar coordinates \( \eta = re^{i\theta} \), and therefore:

\[
\phi(\eta) = \frac{\sigma \sqrt{\pi a}}{\sqrt{2\pi r}} e^{-\frac{1}{2}i\theta}
\]

Finally, the results for the stress components are:

\[
\sigma_x = \frac{\sigma \sqrt{\pi a}}{\sqrt{2\pi r}} \cos \frac{\theta}{2} \left( 1 - \sin \frac{\theta}{2} \sin \frac{3\theta}{2} \right) \quad (2.9)
\]

\[
\sigma_y = \frac{\sigma \sqrt{\pi a}}{\sqrt{2\pi r}} \cos \frac{\theta}{2} \left( 1 + \sin \frac{\theta}{2} \sin \frac{3\theta}{2} \right) \quad (2.10)
\]

\[
\tau_{xy} = \frac{\sigma \sqrt{\pi a}}{\sqrt{2\pi r}} \sin \frac{\theta}{2} \cos \frac{\theta}{2} \cos \frac{3\theta}{2} \quad (2.11)
\]

For a biaxially loaded infinite plate the factor \( f(a/W) = 1 \) and therefore, \( K_I = \sigma \sqrt{\pi a} \), depends only on the applied stress and crack length. The derivations apply for infinitely sharp tips, for cracks with blunted tips, the blunting radius \( \rho \) should be accounted for in the stress equations.
Superposition principle

The principle of superposition can be applied in linear elastic fracture mechanics to calculate stress components and stress intensity factors as follows:

\[
(\sigma_{ij})_{\text{total}} = (\sigma_{ij})_1 + (\sigma_{ij})_2 + \cdots + (\sigma_{ij})_n \tag{2.12}
\]

and from equation 2.2.2 follows:

\[
(\sigma_{ij})_{\text{total}} = (K_I)_1 \cdot f_{ij}(r,\theta) + (K_I)_2 \cdot f_{ij}(r,\theta) + \cdots + (K_I)_n \cdot f_{ij}(r,\theta)
\]

Finally, the following result is obtained:

\[
(\sigma_{ij})_{\text{total}} = (K_I)_{\text{total}} \cdot f_{ij}(r,\theta)
\]

where

\[
(K_I)_{\text{total}} = (K_I)_1 + (K_I)_2 + \cdots + (K_I)_n
\]

where \((K_I)_n\) corresponds to the load \(\sigma_n\), applied to the specimen.

Although the principle of superposition is illustrated for mode I of crack surface displacement, it is applicable to mode II and mode III as well.

Crack tip blunting

For cracks with blunted tips, shown in figure 2.3, stress field is not singular as it is the case with sharp crack tip. The near crack tip stress field is defined by:

\[
\sigma_x = \frac{K_I}{\sqrt{2\pi r}} \cos \frac{\theta}{2} \left( 1 - \sin \frac{\theta}{2} \sin \frac{3\theta}{2} \right) - \frac{K_I}{\sqrt{2\pi r}} \frac{\rho}{2r} \cos \frac{3\theta}{2} \tag{2.13}
\]

\[
\sigma_y = \frac{K_I}{\sqrt{2\pi r}} \cos \frac{\theta}{2} \left( 1 + \sin \frac{\theta}{2} \sin \frac{3\theta}{2} \right) + \frac{K_I}{\sqrt{2\pi r}} \frac{\rho}{2r} \cos \frac{3\theta}{2} \tag{2.14}
\]

\[
\tau_{xy} = \frac{K_I}{\sqrt{2\pi r}} \sin \frac{\theta}{2} \cos \frac{\theta}{2} \cos \frac{3\theta}{2} - \frac{K_I}{\sqrt{2\pi r}} \frac{\rho}{2r} \sin \frac{3\theta}{2} \tag{2.15}
\]

Stress intensity and stress concentration factors

Stress intensity factors \(K\) define the stress field near the crack tip and have dimension \([MPa\sqrt{m}]\). Stress concentration factor, on the other hand, define the ratio between the remotely applied stress and the local stress increase due to a geometric feature and it is dimensionless value. Considering a blunted crack tip the following results are obtained for a stress concentration factor:

\[
\sigma_y = \frac{2K_I}{\sqrt{\pi \rho}} = \frac{2\sigma\sqrt{\pi a}}{\sqrt{\pi \rho}}
\]

and therefore, for the stress concentration factor:

\[
\frac{\sigma_y}{\sigma} = 2\sqrt{\frac{a}{\rho}}
\]
Figure 2.3: Crack with blunted tip

Figure 2.4: Crack in biaxially loaded plate
Finite specimen size

Prior derivations of the stress intensity factors are valid strictly for an infinite plate. If the size of the plate is finite, it must be taken into account using correction coefficients $C$ and $f(a/W)$. The general form a stress intensity factor is:

$$K_I = C\sigma \sqrt{\pi a} f(a/W),$$

where $C$ and $f(a/W)$ are to be determined most often by stress analysis or analytically.

Crack tip plasticity

The linear stress field at the vicinity of the crack tip tends to infinity at the crack tip for a sharp crack. If, however, a real material is considered, there would be a zone, where the calculated linear stress field would be higher than the yield strength $\sigma_y$ of the material. Therefore, the material would plastically deform in that zone. Exact representation of the plastic zone at the crack tip has proved to be extremely challenging. Therefore, representations are either for the size of the crack tip plastic zone with assumed arbitrary shape, or approximation of the shape. For instance if the crack tip plastic zone is assumed to be of a circular shape with diameter $r_y$, then by substituting the yield strength $\sigma_y$ for the stress the following result is obtained:

$$\sigma_y = \frac{\sigma \sqrt{\pi a}}{\sqrt{2\pi r}} = \frac{K_I}{\sqrt{2\pi r}},$$

and therefore for the plastic zone diameter:

$$r_y = \frac{1}{2\pi} \left( \frac{K_I}{\sigma_y} \right)^2.$$

The equation 2.2.2 is a rough approximation, due to the selection of the shape of the plastic zone is arbitrary and the stress field is limited to the $\sigma_y$ and the higher calculated linear stress field in that region is not accounted for. The approximation is shown in figure 2.5.

More accurate representations of the crack tip plastic zone are derived by Irwin and Dugdale.

Representations of the shape of the crack tip plastic zone of first order are obtained by utilizing the yield criteria by von Mises or Tresca. In that way only the boundaries, where the material starts to yield are obtained. Furthermore, it is not accounted for the area, where the linear elastic stress exceeds the yield stress. A visualization based on the von Mises yield criteria for plane stress and plane strain is shown in figure 2.6.

2.2.3 Energy approach to Fracture Mechanics

Total energy

Consider an infinite plate with a through thickness crack with length $2a$, subjected stress $\sigma$. Therefore, for unit thickness of the plate the following quantities
Figure 2.5: Stress distribution in the vicinity of the crack tip

Figure 2.6: Visualization of the yield stress boundaries of the crack tip plastic zone
are defined: $U_0$ total energy of the plate and its loading system before introducing the crack, $U_a$ change in the elastic energy of the plate, caused by introducing a crack, $U_\gamma$ change in the plate surface energy due to the crack, $F$ work performed by the loading system during the introduction of the crack. Therefore, the total energy of the plate is:

$$U = U_0 + U_a + U_\gamma - F$$

**Potential energy**

The part of equation 2.2.3 that can perform work is defined as potential energy:

$$U_p = U_0 + U_a - F$$  \hspace{1cm} (2.16)

**Energy balance**

For the considered case, the total energy changes with the crack length $a$ as shown in figure 2.7. The total energy has a maximum at point $O$, where crack growth becomes unstable. The condition for instability is given by:

$$\frac{\partial U}{\partial a} < 0$$

substituting equation 2.16:

$$\frac{\partial(U_0 + U_a - F)}{\partial a} < 0$$

and therefore:

$$-\frac{\partial U_p}{\partial a} > \frac{\partial U_\gamma}{\partial a}$$

![Figure 2.7: Total energy of a plate as a function of the crack length $a$](image-url)
Potential energy release rate

The potential energy release rate $G$ is the energy per unit thickness available per crack extension increment:

$$G = -\frac{\partial U_p}{\partial 2a}$$

LEFM and EPFM

An approximate graph comparing the potential energy release rate calculated with EPFM and LEFM methods against load magnitude $\sigma$ and yield strength $\sigma_y$ is shown in figure 2.8. The discrepancy $A - A'$ at small stresses is small, however, when the loads increase, the crack tip plastic zone increases and the values start to diverge $B - B'$.

![Figure 2.8: Potential energy release rate for EPFM and LEFM for different loads](image)

2.2.4 Elliptic cracks

Cracks considered up to this point are through thickness cracks. However, in practice for bulk components, elliptic cracks are observed. Elliptic cracks are of three types, depending on the relative position of the crack with respect to the component:

- **embedded or full elliptic** crack is located inside the component.
- **surface or semi-elliptic** crack is an elliptic notch on the surface of the component.
- **edge or quarter elliptic** crack is an elliptic notch on two intersecting surfaces.

Cross sections of the crack plane of the elliptic crack types are shown in figure 2.9.

Analytical solutions for embedded elliptic cracks

The first analytical solution for embedded elliptic crack is derived by Irwin and is for $K_I$ crack mode:
\[
K_I = \frac{\sqrt{\pi} b}{E(k)} \left( \sin^2 \phi + \frac{b^2}{a^2} \cos^2 \phi \right)^{1/4}
\]

where \( E(k) \) is an elliptic integral of the second kind and \( \phi \) is the angle of the point on the crack front. A cross section of the elliptic crack plane is shown in figure 2.10.

**Orientation of an embedded elliptic crack**

The orientation of an embedded elliptic crack in an infinite cylinder subjected to tensile stress \( \sigma_t \) is visualized in figure 2.11. The orientation of the crack is completely defined by two angles \( \gamma \) and \( \omega \). Rotation of the crack around the axis of the cylinder is determined by the \( \omega \) angle. Whereas, \( \gamma \) is the angle between the axis of the cylinder and the normal to the crack plane.

Therefore, the stresses for the analytical solutions of the stress intensity factors are:
\[
\sigma = \sigma_t \cos^2 \gamma
\]

and

\[
\tau = \sigma_t \cos \gamma \sin \gamma
\]

Figure 2.11: Orientation of an elliptic crack

**Analytical solutions for mixed mode loading of elliptic cracks**

Analytical solutions for \( K_1, K_{II}, \) and \( K_{III} \) for embedded elliptic crack are obtained in ref [18] and are as follows:

\[
K_1(\beta) = \sigma \sqrt{\frac{\pi (b/a)}{E(k)}} \left( a^2 \sin^2 \beta + b^2 \cos^2 \beta \right)^{1/4}
\]

(2.17a)

\[
K_{II}(\beta) = -\tau \left( \frac{\pi b}{a} \right)^{1/2} \frac{bR(K, v) \cos \beta \cos \omega + aQ(k, v) \sin \beta \sin \omega}{\left( a^2 \sin^2 \beta + b^2 \cos^2 \beta \right)^{1/4}}
\]

(2.17b)

\[
K_{III}(\beta) = \tau (1 - v) \left( \frac{\pi b}{a} \right)^{1/2} \frac{aR(k, v) \sin \beta \cos \omega - bQ(k, v) \cos \beta \sin \omega}{\left( a^2 \sin^2 \beta + b^2 \cos^2 \beta \right)^{1/4}}
\]

(2.17c)

where \( \beta \) and \( \omega \) are the angles defining a point on the crack front and the angle determining the orientation of the crack in the crack plane, as illustrated in figure 2.10, \( a \) and \( b \) are the major and minor ellipse axes, \( v \) is the Poisson’s ratio, \( K(k) \) and \( E(k) \) are the complete elliptic integrals of first and second kind, \( k = \sqrt{1 - b/a} \), \( k_1^2 = 1 - k^2 \) and
\begin{align*}
R(k, v) &= \frac{k^2}{[(k^2 - v)E(k) + vk^2 K(k)]} \\
Q(k, v) &= \frac{k^2}{[(k^2 + vk^2)E(k) - vk^2 K(k)]}
\end{align*}

In ref [10] an error in equations 2.17 is discovered that has gone unnoticed, namely that instead of $\beta$, angle $\phi$ should be used figure 2.10. Observations in ref [10] are later confirmed in ref [24] and the following equations are proposed and used in the project as reference analytical solutions:

\begin{align*}
K_I(\beta) &= \sigma \sqrt{\pi b/a} \left( \frac{a^4 \sin^2 \beta + b^4 \cos^2 \beta}{(a^2 \sin^2 \beta + b^2 \cos^2 \beta)^{1/4}} \right) \\
K_{II}(\beta) &= -\tau \left( \frac{\pi b}{a} \right)^{1/2} \left( \frac{b^2 R(k, v) \cos \beta \cos \omega + a^2 Q(k, v) \sin \beta \sin \omega}{(a^2 \sin^2 \beta + b^2 \cos^2 \beta)^{1/4} (a^4 \sin^2 \beta + b^4 \cos^2 \beta)^{1/4}} \right) \\
K_{III}(\beta) &= \tau (1 - v) \left( \frac{\pi b}{a} \right)^{1/2} \left( \frac{a^2 R(k, v) \sin \beta \cos \omega - b^2 Q(k, v) \cos \beta \sin \omega}{(a^2 \sin^2 \beta + b^2 \cos^2 \beta)^{1/4} (a^4 \sin^2 \beta + b^4 \cos^2 \beta)^{1/4}} \right)
\end{align*}

\subsection*{2.3 LEFM analysis with FEM}

\subsection*{2.3.1 Special crack tip elements}

The stress field in the vicinity of the crack requires special attention, due to its singularity. Higher mesh density is necessary to approximate properly the stress field. In addition to this requirement, the stress singularity may be accounted for in two ways, either approximating it with very dense mesh or using special crack tip elements.

For instance the required quadratic displacement function of a triangular element is:

\[ u(\xi, \eta) = a_1 + \frac{a_2 \xi + a_3 \eta}{\sqrt{\eta + \xi}} + \frac{a_4 \xi \eta}{\xi + \eta} + a_5 \xi + a_6 \eta \]

which corresponds to the standard quadratic polynomial displacement function:

\[ u(\xi, \eta) = a_1 + a_2 \xi + a_3 \eta + a_4 \xi \eta + a_5 \xi^2 + a_6 \eta^2 \]

The effect is achieved by translation the mid-side nodes for quadratic elements to the crack tip, figure 2.12.

The \sqrt{r} singularity is achieved by coalescence of the nodes, as shown in figure 2.12.

\subsection*{2.3.2 Calculation of the stress intensity factors}

Methods of calculating stress intensity factors fall into two main categories substitution and energy methods.
Substitution methods

Substitution methods are post-processing procedures and use the calculated values for stress and displacement by the FEM analysis. They are of two types displacement and stress. Stress intensity factors are obtained by substituting the values for either stress or displacement in the Westergaard’s equations. For the substitution, also the \((r, \theta)\) coordinates of either Gauss points or nodes for stress and displacement respectively are required and can be easily calculated.

Energy methods

Numerous energy methods have been developed, they calculate directly only \(G\). Subsequently the stress intensity factors are obtained from \(G\). Some of the most common energy methods are ref [14]:

- energy difference technique
- virtual crack extension methods
- \(J\) - integral
- crack closure/opening work
- weight functions

From the above methods, only the energy difference technique and virtual crack extension methods are reviewed.

Energy difference technique

The energy difference technique for two dimensional analysis is composed of three steps:

- Perform analysis and calculate the potential energy \(P_1\) of the crack with given length \(a\).
- Move the position of the crack tip with \(\delta a\), where \(\delta a\) is much smaller than the size of the crack tip elements, and perform analysis to calculate the potential energy \(P_2\).
- Finally, calculate \(G\) by:

\[
G = \frac{dP}{d\delta a} = -\frac{P_2 - P_1}{\delta a}
\]
The accuracy of $G$ is dependent on the value of $\delta a$, in a way that $G$ values may be inaccurate if $\delta a$ is either too small or too large. The technique can be extended to three dimensional analyses by performing the procedure for each node of the crack. This, however, leads to multiple runs, of the analysis.

**Virtual crack extension methods**

Virtual crack extension methods are equivalent to the energy difference method, however, they eliminate the need for multiple simulations. Virtual crack extension methods are of two types discrete and continuum.

**Discrete virtual crack extension methods**

This section is a review of some of the first virtual crack extension methods.

The stiffness derivative method was introduced in ref [25] The method uses the strain energy expression:

$$U = -\frac{1}{2} \{u\}^T [K] \{u\}$$

(2.19)

where $[K]$ is the stiffness matrices over the crack tip region. The method takes the derivative of the equation 2.19 with respect to the crack length $a$. The matrix $[\delta [K]]$ is calculated as difference of the stiffness matrices for crack lengths $a$ and $a + \delta a$. Then vector products of $[\delta [K]]$ and the crack tip local displacements give the energy change.

A method developed in ref [15] calculates $[\delta [K]]$ only for elements affected by the crack extension.

**Continuum virtual crack extension methods**

In continuum virtual crack extension methods $\delta a$ is defined algebraically, rather than explicitly. In ref [8] is shown that the potential energy release rate can be derived as:

$$G = \int_V \left\{ \left( \sigma_{ij} \frac{\partial u_i}{\partial X_j} - W \delta_{jk} \right) \frac{\partial X_k}{\partial X_j} - f_i \frac{\partial u_i}{\partial X_k} \Delta X_k \right\} dV$$

(2.20)

where $W$ is the strain energy density, $\sigma_{ij}$ and $u_i$ are the stress and displacement tensors,

$$\delta_{jk} = \begin{cases} 0, & \text{if } j \neq k \\ 1, & \text{if } j = k \end{cases}$$

is the Kronecker delta, and $X_k$ are the Cartesian geometric values. The gradient $\Delta X_k$ varies linearly with $\delta a$ in the region $B$, figure 2.13, is equal to $\delta a$ in region $C$ and is zero in the region $A$.

**2.4 XFEM**

**2.4.1 Introduction**

XFEM or eXtended Finite Element Method is a technique to analyze discontinuities, independently of the of the mesh of the part. This is contrary to FEM,
which requires the mesh to conform to the crack geometry. A major feature of XFEM is that it builds upon the classic FEM and is an extension of the method, rather than a completely different methodology. For further details on XFEM can be found in ref [23], [5] and [1].

### 2.4.2 XFEM concepts

The basic concept of XFEM is to incorporate modeling of discontinuities into the element definition, and therefore, the mesh does not need to conform to the geometry of the crack. This is implemented by local enrichment of the elements surrounding the crack.

#### Partition of unity

Partition of unity is defined as a set of $m$ functions $f_k(x)$ in a domain $\Omega$, such that:

$$ \sum_{k=1}^{m} f_k(x) = 1 $$

and therefore it is true that:

$$ \sum_{k=1}^{m} f_k(x) \psi(x) = \psi(x) $$

The shape functions $N_j$ of isoparametric elements also satisfy the condition:

$$ \sum_{k=1}^{n} N_j(x) = 1 $$

where $n$ is the number of nodes in an element.
Enrichment

One way to consider enrichment is by incorporating analytical solutions for crack tip stress field for fracture analysis and thus increasing the accuracy of the solution. A starting point, when considering enrichment is the approximation function of a field variable:

\[ u = \sum_{j=1}^{n} N_j u_j \]

The same expression can be rewritten in terms of the \( m \) basis functions \( p_k \):

\[ u = \sum_{k=1}^{m} p_k a_k \tag{2.21} \]

where \( a_k \) can be determined from approximation at nodal points.

Enrichment is of two types:

**Intrinsic enrichment** is achieved by modification of the basis function

**Extrinsic enrichment** is achieved by adding new basis functions to the approximation.

**Intrinsic enrichment**

Intrinsic enrichment is achieved by modification of the basis function \( p_k \), so that it includes additional terms, which meet a requirement. For instance in equation 2.21, \( p_k = \{1, x, y\} \) for a linear two dimensional case. However, for the enriched basis for representation of the near crack tip strain field is:

\[ p^T(x) = \left[ 1, x, y, \sqrt{r} \sin \frac{\theta}{2}, \sqrt{r} \cos \frac{\theta}{2}, \sqrt{r} \sin \theta \sin \frac{\theta}{2}, \sqrt{r} \sin \theta \cos \frac{\theta}{2} \right] \]

where \( \theta \) and \( r \) are the polar coordinates of the system at the crack tip, shown in figure 2.1. And finally the strain field is:

\[ u(x) = p^T(x)a(x) \]

where \( a(x) \) is a vector of coefficients.

**Extrinsic enrichment**

Extrinsic enrichment uses external basis functions \( p_k \), so that the equation for the strain field becomes:

\[ u(x) = \sum_{j=1}^{n} N_j(x) u_j + \sum_{k=1}^{m} p_k(x) a_k \]
2.4.3 Partition of unity based methods

Partition of unity finite element method – PUFEM

The PUFEM is one of the key developments, which eventually lead to the XFEM. Its main features ref [22] are the capability to incorporate knowledge about a certain behavior in the finite element definition. The PUFEM is a generalization of the $h$, $p$ and $hp$ versions of FEM ref [4].

The displacement interpolation function for PUFEM is:

$$u(x) = \sum_{j=1}^{n} N_j(x) \left( u_j + \sum_{k=1}^{n} (p_k(x) - p_k(x_j)) a_j \right)$$

Generalized finite element method – GFEM

The GFEM utilizes different shape functions for the FEM and the enriched interpolation. Therefore, the generalized form of the displacement field is:

$$u(x) = \sum_{j=1}^{n} N_j(x) u_j + \sum_{j=1}^{n} \bar{N}_j(x) \left( \sum_{k=1}^{m} p_k(x) a_{jk} \right)$$

eXtended finite element method – XFEM

The XFEM uses local enrichment, opposed to global as is the case with PUFEM and GFEM. Local enrichment may, however, lead to incompatible solution between the local enriched region and the rest of the analyzed domain. Therefore, a transition zone, composed of blending elements is introduced on the boundary between the two regions of the domain, figure 2.14.

![Standard, enriched and blending elements in a domain](image)

Figure 2.14: Standard, enriched and blending elements in a domain

The displacement field of the transition zone is described by:

$$u(x) = (1 - R(x)) u(x) + R(x) u_{enr}(x)$$

where $R(x)$ is a ramp function so that it is 1 at the enriched boundary and 0 at the standard element boundary.
2.4.4 XFEM formulation

Approximation I

The general approximation function for displacement $u(x)$ has the form:

$$u(x) = \sum_{j=1}^{n} N_j(x) u_j(x) + \sum_{j=1}^{n} N_j(x) \psi(x) a_k$$

where $u_j$ is the vector of the nodal degrees of freedom, $a_k$ is the added set of degrees of freedom added to the standard finite element model, $\psi(x)$ is the enrichment function for the discontinuity.

Level set method for tracking boundary

Level set functions are utilized to define the location of the crack in the domain. A case for a flat crack is illustrated in figure 2.15.

![Figure 2.15: Level set functions for a flat crack](image)

The functions $\Psi$ and $\Phi$ completely define the crack location in the domain. For instance the boundary of the crack corresponds to $\Psi = 0$ and $\Phi = 0$. In addition, $\Phi > 0$ inside the crack contour and $\Phi < 0$ outside. The $\Psi$ function defines the top and bottom of the crack, and $\Psi = 0$ at the crack plane.

Heaviside function

Consider a crack and a point $x'$ on the crack flank in figure 2.16. The Heaviside function $H(x)$ is defined as:

$$H(x) = \begin{cases} 
1, & \text{if } (x - x') n \geq 0 \\
-1, & \text{otherwise}
\end{cases}$$
Figure 2.16: Crack flanks and crack tip

Approximation II

Finally the approximation for the displacement field has the form:

$$u(x) = \sum_{j=1}^{n} N_j(x) \left( u_j + H(x) a_j + \sum_{k=1}^{m} F_k(x) b_j^k \right)$$

where

$$\sum_{k=1}^{m} F_k(x) b_j^k$$ is the crack tip enrichment term,

$$H(x) a_j$$ is the Heaviside enrichment term,

$$u_j$$ are the nodal degrees of freedom and

$$m$$ is the number of the enriched nodes of the element and

$$F_k = [\sqrt{r}\sin \frac{\theta}{2}, \sqrt{r}\cos \frac{\theta}{2}, \sqrt{r}\sin \theta \sin \frac{\theta}{2}, \sqrt{r}\sin \theta \cos \frac{\theta}{2}]$$

2.4.5 **XFEM** crack definition in Abaqus

With **XFEM** both stationary crack and crack growth may be defined ref [29]. For stationary crack the initial geometry and location of the crack is defined by **faces** one, two or three dimensional – figure 2.17. In addition, the crack can also intersect the elements of the crack domain arbitrarily.

Conversely, in the case of a non-stationary crack, its location and geometry may not be specified explicitly, in which case is calculated on the basis of damage initiation and evolution law.

2.4.6 Crack analysis with **XFEM**

Analysis of stress intensity factors with **XFEM** and tetrahedral elements is presented in ref [3]. **XFEM** solutions have noise and therefore, techniques are developed to reduce the noise in the solution. A technique proposed in ref [26] is based on using selective data from the solution. A "moving average" technique, employed in ref [3].

The capability of **XFEM** to include priori knowledge about a discontinuity facilitates multiscale analysis. In this case multiscale refers to combining **FEM** to model the whole component and a modeling technique to simulate
2.5 Fracture mechanics software

General purpose FEM software like Abaqus and Ansys are not specifically designed to meet the needs of fracture mechanics. Therefore, fracture mechanics oriented software has been developed. This section is a brief review of some of the most widely used fracture mechanics software. In particular, the review is limited to Zencrack, FRANC2D and FRANC3D, NASGO, AFGROW and ADAPCRACK3D.

2.5.1 Zencrack

Zencrack is a FEM 3D fracture mechanics software. It uses automated mesh generation according to the crack geometry. Zencrack runs along side a general finite element program and requires either Abaqus, Ansys or MSC Marc. Zencrack also is available in two versions Standard and Professional. The Standard version is able to calculate fracture parameters for stationary cracks, while the Professional version includes capabilities for crack growth analysis. Zencrack works by defining a standard element block, which contains the crack geometry. The standard element block is then included in the analysis, by replacing corresponding elements of the mesh ref [19].

2.5.2 FRANC2D and FRANC3D

FRANC3D is a 3D fracture mechanics software developed by Cornell university, ref [17]. It is a FEM and BEM based. FRANC2D is a two dimensional version of FRANC3D.
2.5.3 NASGRO

*NASGRO* is a software developed by NASA for fracture analysis. It is NASA’s standard software package used by all NASA Centers ref [11]. It is utilized for fracture control and damage tolerance assessment.

2.5.4 AFGROW

*AFGROW* is a fracture mechanics software for analysis of crack initiation and crack growth.

2.5.5 ADAPCRACK3D

*ADAPCRACK3D* is a fracture mechanics software for fatigue crack growth of 3D cracks under arbitrary loading. A main purpose of the program is to determine crack path and surfaces and remaining life of a component ref [28].
Chapter 3

Methodology

3.1 Introduction

The procedures of building the models in Abaqus are implemented by Abaqus kernel commands. These commands are organized onto custom classes and functions, comprising the program. This organization enables scalability, further automation, reduces code duplication and facilitates further development and maintainability.

3.2 Description of the model types

The program is able to analyze 4 model types for each crack type. In addition to that, the FEM analysis type model, has 3 mesh transformation types. This results in total of 18 different model databases. For FEM analysis type, the model type is one – crackNormal, however, the model is capable of 3 mesh transformations, which can be considered as a model sub-types. For XFEM analysis type the model types are 3 – simple, crackPartition and multiplePartitions.

3.2.1 FEM – crackNormal model type

The crackNormal model type is the most complex of all model types. The complete analysis of a crack with this model type is performed at two stages. First, building a model with a circular crack and associative to the geometry mesh. Second, importing the input file from the first to create an orphan mesh, applying a transformation to the mesh to obtain the desired crack shape and creating boundary conditions.

Geometry of the model

Geometry of the model is a cylinder and at its center is located the crack. Initially the crack is modeled as circular. The geometric parameters are shown in fig 3.1. The model is composed of multiple parts, which define the majority of the internal edges of the model. For instance, edges of the crack zone and crack tip are the intersection between a cylinder and two washer-shaped solid parts with square cross section. In similar fashion, slanted edges connecting the
vertices of the crack zone edges and the cylinder are created by the intersection of the cylinder with 4 shell parts. Additional edges and partitions are created by subsequent partitioning of the cylinder by datum planes. The cross section of the model is shown in figure 3.1. Thus created edges, faces and cells are organized into sets and utilized in the definition of the features of the model.

Figure 3.1: Cross section of the crackNormal model

**Geometric parameters**

The geometry of the model is completely defined by the following parameters, as shown in figure 3.1:

- **crackRadius** is the radius of the crack as modeled. The radius is either equal to the elliptic crack minor axis, for elliptic and simpleScale transformations or $\sqrt{ab}$ for the advancedScale transformation, where $a$ and $b$ are the elliptic crack axes. The value of the crackRadius is calculated by the FEM-dataStr.calculateCrackRadiusBeforeTransformation() method.

- **crackZoneSide** defines the crack zone area of the model, has a square cross section and is an area of higher mesh density to accommodate the stress field around the crack front. The crackZoneSide, though independent parameter, is set to be equal the crackRadius.

- **crackTipSide** defines the zone directly surrounding the crack front. It has a square cross section and is meshed with WEDGE elements. The value of
the parameter can be set either explicitly or as a fraction of the crackZoneSide by the FEMdataStr.calculateCrackTipSide() method.

**containerHeight** parameter defines the height of the cylinder containing the crack.

**containerRadius** parameter defines the radius of the cylinder containing the crack. Both containerHeight and containerRadius should be sufficiently large to accommodate the crack. They should also define a cylinder, large enough volume to ensure that the stress field is homogeneous and thus not affect the stress intensity factors.

In total the geometry of the model is defined by four parameters containerHeight, containerRadius, crackZoneSide and crackTipSide.

**Crack parameters**

The crack parameters define the geometry of the crack $a$ and $b$ and the type of the crack. The crack geometry parameters $a$ and $b$ are the ellipse axes, corresponding to the $X$ and $Z$ coordinate axes. The crackType parameter defines the what crack would be analyzed. Possible values are embedded, surface and edge. The crackType determines the geometry of the model. For embedded crack the container is a full cylinder and for surface and edge, it is a cylinder section of 180 and 90 degrees respectively.

**Geometric sets**

Thus partitioned, the geometry of the model is organized into sets, which are subsequently utilized in the definition of the features of the model. Cell sets are defined to facilitate assignment of mesh controls. Face set is defined to facilitate contact definition between the crack flanks. Edge sets are created to facilitate seeding of the model and definition of the crack front.

The following edge sets are defined:

- **allArcEdges** set, includes all concentric internal and external circumferential edges. The edge seeds of the set define the angular density of the mesh and the number of elements along the crack front.

- **containerRefinementEdges** set, includes edges connecting the vertices of the crack zone and the cylinder. The seeds of these edges define the density of the mesh surrounding the crack zone. This area is of secondary interest, as the stress field is mostly constant.

- **crackFrontEdges** set, includes the edges of the crack front. It is utilized to define the crack.

- **crackTipRefinementEdges** set, includes the edges, having one of their ends on the crack front. These edges are utilized to constrain the number of seeds, so that the crackTipCells are meshed with WEDGE elements only.

- **crackZoneHorizontalEdges** set, includes edges of the crack zone and edges from the cylinder top and bottom surface. The edges have equal number of seeds and define the number of elements around the crack front and thus
are one of main parameters influencing the accuracy of the evaluation of the contour integral.

**crackZoneRefinementEdges** set, includes edges going radially from the crack tip zone. Seed number is related to the seeds of the **crackZoneHorizontalEdges** and also influences accuracy.

**crackZoneVerticalEdges** set includes edges of the crack zone, are partitioned by the XY plane, edges of the cylinder wall and inner cylinder. The seeds of the edges are calculated as half of the seeds assigned to the **crackZoneHorizontalEdges**.

**innerCylinderHorizontalEdges** set includes the radial edges of the inner cylinder.

Cell sets are utilized to facilitate the assignment of the mesh controls. The following cell sets are defined:

**crackTipCells** set is utilized to assign SWEEP meshing technique and WEDGE element shape to the cells surrounding the crack front.

**innerCylinderCells** set is utilized to assign SWEEP meshing technique and Medial axis algorithm to the cells. This meshing technique guarantees, that the cells contain WEDGE elements along the Z axis. These WEDGE elements may become severely distorted during mesh transformation and may fail if the crack aspect ratio becomes large. Therefore, the **FEMorphantMesh** class provides method to delete these elements and close the resulting hole, by moving and merging the nodes of the adjacent Hex elements.

**Seeds**

Seeds of the model are defined as by number, corresponding to edges of the model. The model seeds are completely defined by 4 parameters as follows:

**crackZoneMainSeeds** define the number of seeds and respectively the mesh density around the crack front. The crackZoneMainSeeds are assigned to the **crackZoneHorizontalEdges** edge set. Half of the crackZoneMainSeeds are assigned to the **crackZoneVerticalEdges** edge set.

**crackZoneRefinementSeeds** defines the mesh density in radial direction of the crack front. The seeds are assigned to the **crackZoneRefinementEdges**.

**arcSeeds** define the number of elements along the crack front. Seeds are assigned to the allArcEdges edge set.

**containerRefinementSeeds** define the mesh density of the container. The seeds are assigned to the **containerRefinementEdges** edge set.
Mesh parameters

The model can be meshed with linear and quadratic, both full and reduced integration elements. Mesh transformation is utilized to obtain a crack shape corresponding to the $a$ and $b$ parameters from the initial circular crack. Mesh transformation is applied to the orphan mesh model and is of three types:

- **simpleScale** transformation multiplies either $X$ or $Y$ coordinate of each node, depending on the ratio $a/b$, by a factor equal to the $\frac{\text{majorAxis}}{\text{minorAxis}}$ of the crack. Then the node is moved to the new coordinate.

- **advancedScale** transformation multiplies both $X$ and $Y$ coordinates of each node, by $\text{expansion}$ and $\text{contraction}$ factor, depending on the crack ellipse axes. The both factors are as follows:

  \[
  \text{expansion} = \frac{\text{majorAxis}}{\text{crackRadius}}
  \]

  and

  \[
  \text{contraction} = \frac{\text{minorAxis}}{\text{crackRadius}}
  \]

- **elliptic** transformation multiplies either the $X$ or $Y$ axis, depending on the
crack parameters by:

\[ x = x \sqrt{1 + \frac{a^2 - b^2}{x^2 + y^2}} \]

or

\[ y = y \sqrt{1 + \frac{a^2 - b^2}{x^2 + y^2}} \]

depending on the crack ratio.

**Analysis parameters**

Analysis parameters define the boundary conditions for the model. Analysis parameters for embedded crack are as follows:

- \( \sigma \) is the applied tension to the an infinite cylinder.
- \( \gamma \) angle, defines the rotation of the crack with respect to the applied tension.
- \( \omega \) angle, defines the rotation of the crack in the \( XY \) plane.

**Material**

Material is linear isotropic with Poisson ration \( v = 0.3 \) and Young’s modulus \( E = 200 \text{ GPa} \).

**Interaction properties**

Interaction properties include the contour integral definition and contact between the crack flanks.

Contact is defined as frictionless as *Tangential behavior* and *HARD as Normal behavior*. Firstly the contact is defined as *SurfaceToSurface* and in the orphan mesh is redefined as *General*, using all surfaces of the model.

Crack interaction is defined using the *crackFrontEdges* edge set a dummy direction of the *qVectors*, *Midside node parameter* as set by the *FEMdataStr settMidNodePosition()* and singularity – collapsed element side, single node. The crack interaction is subsequently redefined in the orphan mesh model, with *crackNormal* as extension direction.

**Inner cylinder operation**

The *WEDGE* elements in the inner cylinder, become severely distorted during mesh transformation and may corrupt the mesh and analysis. Therefore, in case of a crack with large or small ratio of the ellipse axes, these elements are deleted and the resulting hole closed. The difference, in the model is illustrated by figure 3.3. The *WEDGE* elements of the model on the left have been removed and the resulting hole closed by moving the nodes of the hole wall to the axis of transformation and then merging the nodes. On the right side of the figure, however, the *WEDGE* elements have been left, and show severe skewness.
3.2.2 Model types for XFEM analysis

For XFEM analysis of cracks, mesh is not required to comply with the crack geometry and this leads to a considerable simplification of the model geometry. However, finer mesh is required for more accurate results. Further, the XFEM implementation in Abaqus is available for linear elements only. For the XFEM analysis type, three model types are designed: crackPartition, multiplePartitions and simple. The XFEM model types share the analysis, crack and material parameters with the crackNormal model type. Crack definition for XFEM is defined by a crack geometry part and crack domain. The crack geometry part is a shell part of an elliptic shape with major and minor axes corresponding to the crack parameters. The crack domain represents the part, which contains the crack. Both crack geometry part and crack domain are independent from each other. Therefore, identical crack geometry part, with the corresponding minor and major axes can be utilized, regardless of the crack type and model type. Crack domain, however, changes from full cylinder for embedded crack to 180 and 90 degree sector for surface and edge cracks. The model types for the XFEM analysis differ only in their respective crack domains. This allows experimentation with different meshing techniques and elements.

3.2.3 XFEM simple model type

The simple modelType uses a cylinder as a crack domain without any partitions. Therefore, the mesh size is even in the volume.

Geometric parameters

The crack domain is completely defined by the containerHeight and containerRadius.

Geometric sets

The model has one edge set allEdges, containing the edges of the crack domain.

Mesh parameters

The crack domain can be meshed with Tetrahedral or Hexahedral elements. The seed size is assigned to the edges of the allEdges edge set.
Interaction properties

Contact between the crack flanks is defined in the XFEM definition, by specifying contact interaction properties, which are identical to those of the crackNormal.

3.2.4 XFEM crackPartition model type

The crackPartition model type is shown in figure 3.4. The model has a partition in the shape of the crack at its location in the crack domain. This permits defining a finer mesh in the vicinity of the crack, and therefore, more accurate estimation of the stress field. A limitation of this model type is that the crack domain can be meshed with linear tetrahedral elements only.

Crack partition part

The partitioning of the crack domain performed by a part with elliptic shape and minor and major axes equal to the axes of the crack. The part geometry depends also on the crackType. For embedded crack the part is a full ellipse, while for surface and edge crack type, the part is a sector of an ellipse of 180 and 90 angle.

Geometric parameters

The parameters defining the crack domain and the partition part are shown in figure 3.4 are as follows:

a the ellipse axis of the crack corresponding to the X coordinate axis.
b the ellipse axis of the crack corresponding to the Y coordinate axis.

crackRadius radius of the crack domain.
crackHeight height of the crack domain.

Geometric sets

Two geometric sets are defined in the model. The first, allEdges includes all the edges of the crack domain. The second, crackEdges includes the edges created by the partitioning, and which coincide with the edges of the crack geometry part.

Seeds

Seeds are assigned first to the edges of the allEdges set, this operation seeds all edges. However, to obtain finer mesh in the vicinity of the crack, edges from the crackEdges are seeded after the allEdges, which ensures that the seed size assigned by the first operation is overwritten.

Mesh parameters

The crack domain for this model type can be meshed only with Tetrahedral elements.
Figure 3.4: *crack domain* of the *crackPartition* model
3.2.5 XFEM multiplePartitions model type

The multiplePartitions model type is shown in figure 3.5. The crack domain is partitioned by an elliptic cylinder around the crack. This creates cells, which can be finely meshed to increase the accuracy of the solution. The model can be meshed with both Tetrahedral and Hexahedral elements. However, successful meshing mostly with tetrahedral elements may prove unpredictable. Therefore, the model is meshed with Hexahedral elements only.

Geometric parameters

The model is defined by the following parameters:

- \(a\) is the ellipse axis of the crack corresponding to the \(X\) coordinate axis.
- \(b\) is the ellipse axis of the crack corresponding to the \(Y\) coordinate axis.
- offset parameter, defines the cross section of the elliptic cylinder, which is equidistant to the crack ellipse.
- smallContainerHeight defines the height of the elliptic cylinder.
- containerRadius radius of the crack domain.
- containerHeight height of the crack domain.

Partition part and lofts

The crack domain is partitioned first by an elliptic cylinder, then loft surfaces are created between the elliptic cylinder partition edges and the crack domain circumferential edges and finally by datum planes.

The elliptic cylinder is defined by the offset parameter and the crack axes. Its geometry, also depends on the crack type. For embedded crack it is a full cylinder and for surface and edge it is a sector of an elliptic cylinder.

Loft surfaces partition additionally the crack domain, to enable and improve meshing of the model.

Finally, the crack domain is partitioned by XZ and YZ planes in case of an embedded crack, by YZ plane in case of surface crack.

Geometric sets

Two geometric sets are defined in the model: allEdges and crackContEdges. The first contains all the edges of the crack domain and the second only edges defined by the partitioning with the elliptic cylinder.

Seeds

Two seed sizes are assigned to the edges of the crack domain. First, larger seed size is assigned to the edges of the allEdges edge set. Then smaller seed size is assigned to the edges of the crackContEdges, overwriting the previous seed size.

Mesh parameters

The crack domain is meshed with linear Hexahedral elements.
Figure 3.5: crack domain of the multiplePartitions model
3.3 Visualization Odb

The visualization output database is generated to represent the stress intensity factors. A visualization output database is shown in figure 3.6. The objective of the visualization is to provide a clear representation of the stress intensity factors and how they relate with the crack geometry. The visualization is created, by generating nodal, element and field output data, based on the results of the analysis. Stress intensity factor values are written as scalar nodal quantities. The crack front is represented by truss elements connecting the nodes of the crack. On the figure 3.6, they are the thin elliptic line and represent the $K_3$ values. $K_1$ values are represented by the elements of the two elliptic cylinders located above and below the crack plane. $K_2$ values are represented by the elements in the crack plane.

![Figure 3.6: Visualization output database](image)

3.3.1 Node data

Nodes are created on the basis of the crack geometry and mesh refinement of the corresponding model database. Nodes can be divided into three groups, on the basis of the elements, which they define and the values the elements represent.

**K3 – nodes**. These nodes are used to define the truss elements and have identical coordinates with the nodes of the modeled crack in the model database.

**K1 – nodes**. The K1 – nodes are used to define the shell elements of elliptic cylinder. They are created by offsetting the $Z$ coordinate of each of the K3 – nodes. These nodes lie on four planes, offset from the crack plane. Two of them are above the crack plane and the other two are below it. Each of the two pairs of nodes are used to create the shell elements representing the $K_1$ values.
K2 – nodes. The K2 – nodes are used to define the shell elements, in the crack plane. They are created as equidistant from the K3 – nodes. The offset is measured on the normal to the ellipse at each K3 – node. The created nodes form two pair of nodes, one inside the crack, and one outside. Each pair of nodes is used to create shell elements, which represent the K2 values.

3.3.2 Element data

Elements of the visualization output database are used to interpolate the field output, which is the stress intensity factors. According to the stress intensity factor, they represent, elements can be divided into three groups:

K3 – elements are the truss elements representing the crack front.

K1 – elements are the shell elements on the elliptic cylinders located above and below the crack plane.

K2 – elements are the shell elements in the crack plane, located inside and outside the crack contour.

3.3.3 Field output data

Field output data represents the stress intensity factors for the analyzed crack. The values are averaged over specified contours from the user. The field data is NODAL and of type SCALAR. The stress intensity factors field output is the only available field output for the output database. The data is written for each node, respecting the node label, which identifies the which stress intensity factor value is written.

Regarding K1 and K2, nodes are four times the number of the crack nodes (two pairs of nodes for each), therefore, the data is repeated to comply with the number of nodes. The result for K1 is that nodes with the same x and y coordinates are assigned identical values. For K2, the nodes with identical values, lie on the normal to the crack contour. As for the K3 nodes number corresponds to the number of the crack nodes and, therefore, no further processing is required.

3.4 GUI and Abaqus integration

The program features a graphical user interface to browse the shelve databases, which contain results, input parameters and statistics about every analysis run with the program. The graphical user interface of the program is accessed through the menu Plug-ins → cracks → DB ACCESS.

3.4.1 First dialog box

The interface consists of two dialog boxes. The first dialog box, shown in figure 3.7 asks user to select the crack type to analyze. The program uses this information to determine the shelve database, read its contents and use it to prepare the second dialog box.
3.4.2 Second dialog box

The second dialog box is shown in figure 3.8 for embedded crack type, though the dialog box is identical for the surface and edge crack types. The dialog box is separated into four sections:

- Database records
- Results selection
- Contours
- Items to create

Database records

The Database records is the browser section of the dialog box, it contains all the entries of the corresponding shelve database. The database records are organized in a tree representation based on the input parameters of each entry, narrowing down the selection to the modelName of the analysis, which corresponds to the shelve key of the entry. Only the modelNames of the successful analyses are selectable and selection of multiple entries is possible.

To preserve space logical groups of parameters are abbreviated and concatenated with their respective value. For instance h200r120 denotes a cylinder with height $= 200$ and radius $= 120$. The complete list of abbreviations is given in table 3.1.

Results selection

The Results selection section contains Two subsections Analytical solutions and Analysis results. Both are composed of check boxes to specify which of the stress intensity factors should be included in the analysis.

Contours

The Contours section is composed of two subsections Contours to average and Include contour data. The Contours to average contains a list of
five contours, from which the specified stress intensity factors are averaged. The Separate contours includes the check box Include contour data. The check box is applicable for the creation of XYPlotData only. It specifies that XYPlotData to be created for the each contour of the analysis, otherwise, XYPlotData is created for the averaged data only.

**Items to create**

The Items to create section contains the following check boxes, specifying what actions to perform with the selected data:

- **XYPlotData** creates XYPlotData with the selected options.
- **Print data structure** prints the data contained in the shelf database including the averaged contours, for the selected entries in the tree.
- **Visualization Odb** creates a visualization output databases for the selected entries in the tree and the results from the averaging og the contours.

### 3.5 Organization of the application

The afore described functionality and model, output and shelf databases are created by a number of scripts, organized in modules and packages, which compose the complete program, which closes the cycle of analysis, providing the following functionality:

![Table 3.1: GUI tree abbreviation key](image)
creation of model databases for the different crack and analysis types.

- submitting the model databases.
- post processing and results extraction.
- storing the extracted results to a *shelve* database.
- creation of visualization.
- providing a graphical user interface to browse the data in the *shelve* database and selecting the desired operations and data combination.

### 3.5.1 Structure by function

The program can provisionally be divided into two independent components:

- **Running analyses**
  1. building of Abaqus model databases,
  2. executing the subsequent analysis,
  3. extraction of the required values from the Abaqus output database
  4. saving the results into a custom *shelve* database;

- **Post processing and visualization**
  1. graphical user interface to read the custom database entries
2. creation of plots from the analysis results, and analytical solutions,
3. building of visualization Abaqus output database for stress intensity factors

### 3.5.2 Directory structure and modules

Both functional components share the same directory structure for convenience and portability. Program classes and functions are organized in files, called *modules*. *Modules* are organized in directories, called *packages*. The directory structure of the program is as follows:

```
dir simpleGui
   module createEntryID.py
   module dbAccessDialogs.py
   module dbAccessDialogs_plugin.py
   module executeDbAccessCommands.py

dir db

dir scripts
   dir analyticalSolutions
      module analyticalData.py
      module baseAE.py
      module edgeAE.py
      module embeddedAE.py
      module surfaceAE.py
      module miscFunctions.py
   dir dataStr
      module baseDataStr.py
      module femDataStr.py
      module xfemBaseDataStr.py
      module xfemDataStrMP.py
      module xfemSimpleDataStr.py
      module xfemTETdataStr.py
   dir modelDb
      module baseCrack.py
   dir fem
      module femCrack.py
      module orphanMesh.py
   dir xfem
      module xfemBaseCrack.py
      module xfemCrackMP.py
      module xfemSimpleCrack.py
      module xfemTETcrack.py
```

```
dir persistence
```
The program directory should be located in a directory named `abaqus_plugins`, which should be located in either:

- user home directory,
- abaqus directory,
- current directory,
- or `plugin_dir`, specified in the abaqus environment file.

Installing the program in one of these directories, enables Abaqus to identify the program and make it available in the Plug-ins menu, as a `DB ACCESS` entry item.

### 3.6 Description of classes

#### 3.6.1 Classes interaction

Each class has a strictly defined role and interaction among classes is provided by interfaces (`accessor` methods) implementing the important concept of encapsulation. **Encapsulation** is to isolate the internal operations of an object from the other interacting objects and code one operation only once.

In addition, by encapsulation internal logic and operations of classes are decoupled from each other, which makes them separate units. Moreover, any modifications to the class methods and algorithms of the class do not affect the rest of the program.

A simplified representation of the program structure is shown in figure 3.9. Blocks are given generic names, referring to families of classes or functions, serving equivalent purposes. Therefore, the particular name of the class and implementation depends on the model and analysis type.

**Data structure class** is used to store input parameters, process the input parameters into a format more suitable for the other interacting classes and store output data from the interacting classes and the Abaqus output database.

**Model database class** defines the necessary Abaqus kernel commands to build the model database and submit the analysis.

**ReadOdb class** is utilized to open the Abaqus output database, extract the stress intensity factor values and coordinates of the corresponding nodes, process the coordinates and corresponding values, and write the obtained results to the **Data structure classes**.
Persistence class is utilized to determine the shelf database to which to save, filter the data to be saved, save the data, identify duplicate in the database and read from the database.

Analytical solutions class is utilized to calculate the analytical solution, corresponding to the designated analysis.

Visualization output database class is utilized to create an Abaqus output database for visualization of the stress intensity factors of a designated analysis.

XYPlotData class is utilized to create XYPlotData for a designated analysis.

Db data structure class is utilized to temporarily store data read from the shelf database and process requests from the user.

DataStr classes family is the backbone of the program. It collected input parameters, stores them in a tree structure and provides methods for manipulation of the stored parameters. Furthermore, each of the rest of the classes interacts with a DataStr() class by the provided interfaces.
BaseDataStr() class

The BaseDataStr() is the superclass of the DataStr classes family and the other members are subclasses of the BaseDataStr() class. The BaseDataStr() class defines the methods common with the rest of the dataStr classes. Superclasses are designated in the subclass definition and the superclasses’ methods become available to the subclass. When a class method is called, first the method name is searched in the name space of the subclass, and if not found the search propagates to the superclasses until a method with the corresponding name is found. In case a method is not found, an exception is raised. The search algorithm enables the option for overriding or customization of superclass methods within the subclass by defining a new method with the same name. This programming technique is called inheritance. It is a fundamental technique in object-oriented programming (OOP) and of great utility for reduction of code duplication and increasing code reuse.

The purpose of the BaseDataStr() class is to provide storage and access interface that is inherited by the subclasses. This guarantees, that the interface and the storage structure remain consistent, throughout the program and only the required customizations are implemented in the corresponding subclass.

The BaseDataStr() class, provides methods for:

- initializing and storing input and analysis parameters,
- generating a unique model name for each simulation,
- defining the revolutionAngle parameter, depending on the crack type,
- determination of the analysis success,
- equal sign operator overloading.

A unique model name is created for each analysis by the createModelName() method. The algorithm is based on the number of seconds since the Epoch, which is obtained by the python command time(). The raw format of the name is 1307559473.99, which roughly corresponds to June 6, 2011, 21:57:37. In that raw format, however, the model name is not a valid Abaqus model name, therefore, it is modified to become 1307559473dot99, which is.

Most of the parts of the Abaqus model are defined as revolved parts and their revolutionAngle is different for each crack type (embedded, edge, or surface). The calculateRevolutionAngle() method specifies the revolutionAngle parameter.

The mesh of the FEM model type is fairly complex and is defined by a multitude of parameters, it is also being deformed during transformation. Therefore, it may happen for some parameter configuration, that the mesh may not pass the analysis checks at job submission and the analysis would fail. Therefore, mesh quality checks are performed twice, once right after the meshing of the model, before writing the input for the orphan mesh and second after the mesh transformation of the orphan mesh model. Results of both reports are stored to the data structure, (under the results:mesh:priorTransformation and results:mesh:postTransformation keys) by the methods setPriorMeshTransformationReport() and setPostMeshTransformationReport(). The success of the analysis is set to True should the failedElements is zero for the second (post-transformation) report.
Important option to have is the ability to check whether an identical model with the same configuration of input parameters has been subject to past analysis. To facilitate the implementation, the BaseDataStr() class has an implementation of the equal sign overloading. The method is \texttt{\_\_eq\_\_()} and is executed automatically when an instance of the BaseDataStr() or any of its subclasses is compared for equality. The method verifies, whether the data structure of the class under the \texttt{input} key is equal to the compared object.

**FEMdataStr() class**

FEMdataStr() class is a subclass of BaseDataStr() class and inherits all of the BaseDataStr() methods and also defines new ones. The FEMdataStr() class is utilized for the FEM analysis type.

The essential methods of the class and their application are described below:

- \texttt{\_\_init\_()} method is augmented BaseDataStr.\_\_init\_() method, which sets the parameter \texttt{input:analysisType} to FEM for identification purposes in subsequent analyses of the data.

  - \texttt{setMidNodePosition()} method sets the relative position of the middle node of the singularity at the crack tip. It is designed to assist the mesh of the model to pass the analysis checks. The midNode position is not relevant for stress intensity factors and linear elements.

  - \texttt{createDatumsData()} method creates datum parameters, for datum planes, required for partitioning the model.

  - \texttt{calculateCrackRadiusBeforeTransformation()} method identifies the radius of the crack prior mesh transformation. Initially elliptic crack is modelled as a penny-shaped crack, which after the mesh transformation becomes of elliptic shape. Considering the different transformation types and crack geometry parameters, this method ensures consistent input to the mesh transformation operation.

  - \texttt{calculateCrackZoneSide()} method sets the crackZoneSide parameter, shown in figure 3.1. It is similar to the \texttt{calculateCrackRadiusBeforeTransformation()} method, however, it determines the size of the partition for refined mesh around the crack tip.

  - \texttt{calculateCrackTipSide()} method sets the size of the cracktipSide parameter, shown in figure 3.1. Its value is relative to the crackZoneSide. The method is designed to increase the flexibility of meshing in case meshes do not pass the analysis checks.

  - \texttt{calculateMdbGeometricParameters()} method processes the geometric parameters from the \texttt{input} branch into a more suitable format for subsequent operations. Thus calculated parameters are written to the \texttt{mdb} branch of the data structure.

  - \texttt{createPartsData()} method creates the required parameters for generation of all the constituent parts of the model. Parameters are located under the \texttt{mdb:parts} branch. Under the \texttt{mdb:parts:sketchPoints} are the sketch parameters for shell and solid parts. These parts are composed of one
revolve feature. Parameters for each part correspond to a dictionary key, which serves as a name of the part. In addition, the method creates two more entries for the names of the merged part and orphan mesh part.

createPartitionData() method creates the required parameters for partitioning the model, depending on the crack type. Under the branch mdb:partitionData:innerCylinderPartitions are located parameters for partitioning the inner cylinder cells by datum plane with a designated identification number. Under the mdb:partitionData:piePartitionsDatum are located a number, designating which datums should be used to partition the model according to the crack type.

createPolarCoordinatesOfAPointOnEntityForASet() method created preliminary point coordinates on the XZ plane, for selection of edges, faces and cells, which would be organized in sets. Coordinates of the points are subsequently processed, with respect to the entity and crack type to ensure, that they are on the edges and cells or inside of the cells. Coordinates are created under the mdb:sets:pointsProjectedOnZeroDegreePlane. Depending on the feature, they are representing, are located under the faces for faces and cells for cells. Coordinates for edges are divided into crossSectionEdges for edges in XZ or YZ planes, and into arcEdges for edges on XY or parallel plane.

calculateXYZCoordinates() method calculates and returns the final coordinates of a point on an edge or face or inside a cell. The point is utilized for selection of a geometric feature and organizing it in a set.
createSeedsData() method creates the required parameters to completely define seeding of the model. A complete set of seed parameters is generated for each edge set. Data is stored under the `mdb:seeds` branch of the data structure.

createElementTypeData() method creates element type data, for the model, according to the `elementType` input. The possible values for the input are: linearRI, linearFI, quadraticRI and quadraticFI. Corresponding element codes are as follows, for linearRI – C3D8R, C3D6, C3D4; linearFI – C3D8, C3D6, C3D4; quadraticRI – C3D20R, C3D15, C3D10M; quadraticFI – C3D20, C3D15, C3D10M. Data is stored under the `mdb:meshParameters:elementCodes` branch of the data structure.

setInnerCylinderHoleRadius() method is utilized to store the radius of the modes on the wall of the innerCylinder hole of the orphan mesh model, created by deleting of the wedge elements. Parameter is stored under the `mdb:sets:nodeSetsData:radius` branch of the data structure. The parameter is accessed by the `getInnerCylinderHoleRadius()` method.

processTransformationInputData() method determines the axis of transformation of the mesh in accordance with the crack geometry. The axis of transformation coincides with the crack elliptic shape’s major axis. The method also copies the `transformationType` parameter from `input:meshParameters` to `mdb:meshParameters`. The axis of transformation parameter is stored under the branch `mdb:meshParameters:transformationAxis`.

**XFEMbaseDataStr() class**

`XFEMbaseDataStr()` class is a subclass of `BaseDataStr()`. It serves as a superclass for `XFEMsimpleDataStr()`, `XFEMtetDataStr()` and `XFEM-dataStrMP()`, which are utilized for the corresponding model types of the XFEM analysis types. It is, however, independent of the `FEMdataStr()` class. The purpose of `XFEMbaseDataStr()` as a superclass is to provide methods that are common to its subclasses by customization and augmentation of the `BaseDataStr()` class. Its utility is aligned with the purpose of the `DataStr` classes family.

The essential methods of the class and their application are described below:

`__init__()` method is an augmentation of the `BaseDataStr.__init__()` method, therefore, has the same functionality with the addition of setting the parameter `input:analysisType` to `XFEM` to identify the analysis type in subsequent data analyses.

`createDatumsData()` method, creates parameters defining datum planes. Data is stored to the `mdb:datumsData` branch of the data structure.

`calculateMdbGeometricParameters()` method processes the input geometric parameters from the `input` branch of the data structure into a more suitable format to facilitate subsequent operations. Data is stored to the `mdb:geometricParameters` branch.
createPartsData() method creates the required parameters for generation of the crackDomain and crackGeometry parts. It also writes the name of the mergedPart as a dictionary key. Data is stored under the mdb:parts branch.

createCrackPartitionPartData() method creates the required parameters to create a shell part, that is used to partition the crackDomain part at the crack location to create internal edges in the crackDomain part, to which a smaller seed size is assigned. The part geometry is dependent on the crack type. Geometrically, its shape is represented by the intersection between the crack geometry and the crackDomain. For embedded crack its shape is identical to the crack geometry. For edge and surface crack types it is sector of an ellipse. Data is stored under the mdb:parts:crackPartitionPart branch.

createSeedsData() method creates the complete data for seeding the model. Data is stored under the mdb:seeds branch.

createElementTypeData() method creates element type data for the model, according to the elementType input parameter. Possible values and the corresponding element codes are:

- LinearTET — C3D4
- LinearHexRI — C3D8R
- LinearHexFI — C3D8

Data is stored to the mdb:meshParameters:elementCodes branch.

createCompleteSetsData() method creates the required parameters for edges of the crackDomain part and faces of the crackGeometry part. Data is stored to the mdb:sets:setData branch of the data structure.

setSingularityCalcRadius() method is utilized to store the singularityCalcRadius to input:interactionProperties:crack:singularityCalcRadius.

For complete specification of the accessor methods of the XFEMbaseDataStr class refer to Appendix

XFEMsimpleDataStr() class

XFEMsimpleDataStr() class is a subclass of the XFEMbaseDataStr(), which is a subclass of the BaseDataStr() class. Therefore, the XFEMsimpleDataStr() class inherits all the methods of both of its superclasses, with the customizations of the XFEMbaseDataStr() class. The class is utilized for the simple model type of the XFEM analysis type.

It has one only one method:

_init_() method is an augmentation of the XFEMbaseDataStr._init_() method. In addition to the functionality of the base method it sets the parameter input:modelType to simple to serve as an identification of model type in subsequent analyses.
XFEMtetDataStr() class

XFEMtetDataStr() class is subclass of the XFEMbaseDataStr() class. It is similar to the XFEMsimpleDataStr() class with the difference that XFEMtetDataStr() class is utilized for crackPartition model type of the XFEM analysis type. It is also a single method class with the only method:

_init__() method is an augmentation of the XFEMbaseDataStr.__init__() method. In addition to the functionality of the base method it also sets the parameter input:modelType to crackPartition to serve as an identification in subsequent analyses.

XFEMdataStrMP() class

XFEMdataStrMP() class is a subclass of the XFEMbaseDataStr() class and is utilized for the multiplePartitions model type of the XFEM analysis type.

The essential methods of the class are:

_init__() method is an augmentation of the XFEMbaseDataStr.__init__(). In addition to the functionality of the base method it sets the input:modelType to multiplePartitions for identification in subsequent analyses.

calculateMdbGeometricParameters() method processes the input geometric parameters from the input branch into a more suitable form for subsequent operations. Thus calculated parameters are stored to the mdb:geometricParameters branch of the data structure.

createPartsData() method is an augmentation of the XFEMbaseDataStr.createPartsData() method. In addition to the functionality of the base method it creates data for the smallContainer part, in accordance to the crack type.

calculateCoordsOfCrackContainerEllipseEdges() method calculates and returns the coordinates of a point on the circumferential edge of the smallContainer part. It takes two arguments: an angle of a line through the origin and the point, and the X axis, and hieghtUnitVector, which determines the sign of the Z coordinate.

calculateMdbDatumsData() method creates the required parameters for creation of datum planes. Data is stored to the mdb:datumsData branch.

createAdditionalSetsData() method creates parameters for a set of the edges of the smallContainer. Data is stored under the mdb:setsData:selectByBoundingBoxEdges:ellipseContainerEdges branch.
createSeedsData() method creates the required seed parameters for the model. It redefines the XFEMbaseDataStr.createSeedsData(), which creates has identical purpose, but defines different parameters. Data is stored to the mdb:seeds branch.

3.6.3 Model database classes

Model database classes is a family of classes, which provide methods with Abaqus kernel commands, utilized to create an Abaqus model. To each model type for FEM and XFEM analysis type corresponds a member of the model database classes. All classes take one argument a DataStr class, corresponding to the model type. The required input parameters and supporting analysis parameters are read and stored to the DataStr class. In addition, the model database classes use an internal data structure, named self.mdb, which is of dictionary type and is used to store and point to components of the Abaqus model. The organization of the model database classes family is as follows:

BaseCrackMdb() is a superclass of all other classes in the family;
FEMcrackMdb() and FEMcrackOrphanMesh() used to create the analysis of FEM type;
BaseXFEMcrackMdb() superclass for classes used to create XFEM analysis type, regardless of the model type;
XFEMcrackMdbMP() used to create XFEM analysis type and multiplePartitions model type;
SimpleXFEMcrackMdb() used to create XFEM analysis type and simple model type;
XFEMtetCrackMdb() used to create XFEM analysis type and crackPartition model type.

BaseCrackMdb() class

BaseCrackMdb() class serves as a superclass of all model database classes and provides methods used by most of the subclasses.

The BaseCrackMdb() class methods are as follows:

_init_() method initializes the class, creates the [self.mdb] variable and takes a DataStr class.
initializeAbaqusModel() method creates a new Abaqus model, gives it a name, generated by the DataStr class and deletes the generic Model-1 form the Abaqus database.
initializeViewport() method creates a viewport named after the modelName to display the created model and sets XY plane as compass privileged plane. It also sets self.mdb[viewport] as a pointer to the viewport.
setViewportViewingPoint() method changes the parameters of the isometric view of the viewport, so that the Z axis points upwards and sets a view in the viewport self.mdb[viewport].
setDisplayedObject() method takes one argument, and sets it as a displayedObject of the viewport.

createSolidParts() method calls the createASolidRevolvedPart() for each solid revolved part, specified in the DataStr class.

createShellParts() method calls the createAShellRevolvedPart() for each shell revolved part, specified in the DataStr class.

createASolidRevolvedPart() method creates a sketch and revolves it to create a 3D DEFORMABLE_BODY. The revolution angle is the angle determined by the DataStr class in accordance with the crack type. The created part is pointed to by the self.mdb[parts][name of the part]. After the part is created the method calls createPartOrientation() method.

cREATEPARTORIENTATION method takes a part as an input argument, creates a datum coordinate system of the part and calls the createMaterialOrientation() method, to which it passes the part.

cREATEMATERIALORIENTATION() method sets the material orientation of a part that takes as an input argument.

createAShellRevolvedPart() method creates a sketch and revolves it to create a 3D DEFORMABLE_BODY. The revolution angle is the angle determined by the DataStr class in accordance with the crack type. The created part is pointed to by the self.mdb[parts][name of the part]. After the part is created the method calls createPartOrientation() method.

createMergedPart() method creates a part by merging all the instances in the model assembly and gives it the name specified in the DataStr() class, corresponding to the mergedPart.

cREATEDATUMS() method creates datum planes in the rootAssembly of the model. Datums are created according to the defined parameters in the DataStr class.

deleteAllInstances() method deletes all part instances from the model assembly. The method is used to clear the model of the instances, after the mergedPart is created.

createInstancesFromAllParts() method creates instances of all parts in the model. The method finds all the part names in the model and calls the createInstance() with the part name as an argument. It is used to create instances of all parts, from which the mergedPart is created.

createMergerPartInstance() method creates instance of the mergedPart by calling the createInstance() method and passing the mergedPart name. The method also sets the variable self.mdb[mergedInstance] to point to the created instance.

createInstance() method, takes a part name as an argument, creates instance of the part with the same name and returns the instance.

createMaterial method creates a material with elastic properties and name as specified in the DataStr() class.
createSection method creates a homogeneous solid section with the material name, specified in the DataStr class.

assignSectionToAllParts() method finds the names of all parts in the model and for each one calls the assignSectionToPart() with the part name as an argument.

assignSectionToMergedPart() method calls the assignSectionToPart() with the mergedPart name as an argument.

assignSectionToPart() method takes a part name as an input argument and assigns the section with name designated in the DataStr class.

regenerateAssembly() method calls the regenerate() method of the Abaqus model rootAssembly. The method is utilized to force Abaqus to rebuild and recalculate the model following some operations. This guarantees that subsequent operations are applied to a model that is up to date and all prior operations have been completed. In case of applying commands to a not up to date model, unexpected results may be obtained.

createStep() method creates a static step with a name specified in the DataStr class.

deleteHistoryOutputs() method identifies and deletes all history output requests. The method is utilized to delete the default history output request and to clear the orphan mesh model, after importing the input file.

createHistoryOutput() method creates a history output request for the crack, designating $K_{FACTORS}$ as an output quantity and parameters defined in the DataStr class.

createContactInteractionProperty() method creates a contact property with name specified in the DataStr class and normal behavior as HARD with allowed separation of contact surfaces and FRICTIONLESS tangential behavior.

createGeneralContact() method creates a standard contact with contact property name designated in the DataStr class. This type of contact is utilized for the orphan mesh model to avoid designating contact surfaces, which is required by other contact types.

assignElementType() method assigns the defined in the DataStr elements to the model assembly.

generateMesh() method generates mesh on the mergedInstance.

seedEdges() method assigns a specified number of seeds to every edge of all specified edge sets. Seed number data is obtained from the DataStr class and edge sets are read from the self.mdb[sets][edges].

verifyMesh() method performs a mesh quality verification with the ANALYSISCHECKS option, which is the mesh quality verification, that is performed automatically by Abaqus on job submission. The meshQualityReport command returns a dictionary with the following keys: warningEle-
ments, failedElements, naElements with selection mask of the corresponding elements as dictionary value, and numElements with a number designating the number of the elements. The mesh report is then processed so that only the number of elements corresponding to each key is returned. The importance of the report is that the analysis success can be predicted by the number of the failedElements. If the number of failedElements is not 0, the analysis will fail.

createBC() method calls the appropriate method to create the boundary conditions of the model depending on the crack type. For embedded crack type, the method calls createInfiniteCylinderBC() method. For edge surface crack types methods are not available at the time of writing.

createInfiniteCylinderBC() method creates a displacement constraint for each node of the external faces of the mergedInstace, depending on the coordinates of the node and angles γ and ω, which define the crack orientation in space and σ, which is tension magnitude. Boundary conditions are called displacementBC-n, where n is the consequent number of the BC. Nodal displacements are obtained by the method calculateInfiniteCylinderDisplacementForNode() and passing a node for an argument.

makeRegionForBCFromNode() method returns a region created by a node. The method is utilized to get a region, required by the DisplacementBC() command.

calculateInfiniteCylinderDisplacementForNode() method is utilized to calculate the displacement assigned as a boundary condition of the node. The method employs the following procedure. First, node coordinates (x, y, z) are multiplied by a rotation-transformation matrix (3.1) to obtain coordinates (x', y', z') corresponding to crack orientation defined by γ and ω.

\[
\begin{bmatrix}
\cos \gamma \cos \omega & \cos \gamma \sin \omega & -\sin \gamma \\
-\sin \omega & \cos \omega & 0 \\
\sin \gamma \cos \omega & \sin \gamma \sin \omega & \cos \gamma
\end{bmatrix}
\] (3.1)

Next displacements are calculated by the following formulas:

\[
\Delta z = \frac{\sigma}{E} z'
\]

\[
\Delta y = -v \frac{\sigma}{E} y'
\]

\[
\Delta x = -v \frac{\sigma}{E} x'
\]

Finally (Δx, Δy, Δz) are multiplied by the inverse of the rotation-transformation matrix 3.2 to obtain the displacements at the designated nodes.

\[
\begin{bmatrix}
\cos \gamma \cos \omega & -\sin \omega & \sin \gamma \cos \omega \\
\cos \gamma \sin \omega & \cos \omega & \sin \gamma \sin \omega \\
-\sin \gamma & 0 & \cos \gamma
\end{bmatrix}
\] (3.2)

createJob() method creates an analysis job for the model and setting job parameters to use multiple core CPU and memory limit for optimal performance.
submitJob() method submits the model analysis job and waits for the job completion.

closeMdb() method closes the Abaqus model database.

saveMdb() method saves the Abaqus model database.

FEMcrackMdb() class

FEMcrackMdb() class is a subclass of the BaseCrackMdb() class and therefore, inherits all of its methods. The objective of the FEMcrackMdb() class is to create an Abaqus model database for FEM analysis type of a an elliptic crack, which can be of embedded, edge or surface type. Methods defined within the FEMcrackMdb() along with those inherited from the superclass provide functionality to initialize the model, create the necessary geometry, seed and mesh the model, and write an input file. The FEMcrackMdb() is designed to work with FEMcrackOrphanMesh() class, which is utilized to import and further process the generated input file.

Methods of the FEMcrackMdb() class are as follows:

createAllParts() methods calls the createSolidParts() and createShellParts, which create all of the necessary parts of the model.

partitionMergedPartInstance() method is utilized for further partitioning the mergedPart. The method calls partitionInnerCylinderCells() and partitionContainerAsPie().

partitionInnerCylinderCells() method is utilized to partition the inner-Cylinder cells with datum planes parallel to the emphXY plane. Datum plane numbers are stored in the DataStr and correspond to the order of their creation in the model. The partitioning is required to obtain cells, which can be meshed with sweep technique.

partitionContainerAsPie() method is utilized to partition the mergedInstance into sectors by YZ and XZ datum planes to enable structured and sweep meshing. Datum plane numbers depend on the crack type and are read from the DataStr class.

createSets() method is utilized to create geometry sets for edges, faces and cells. The method calls the corresponding methods createSetsForEdges(), createSetsForFaces() and createSetsForCells().

createSetsForEdges() method is utilized to create all edge sets. Set names and coordinates of a point on each edge are read from the DataStr() class. The method iterates through the names of the sets and for each set iterates through the corresponding edge points, selecting the edges and assigning them to the corresponding set.

createSetsForCells() method is utilized to create all cell sets. Its operation is analogous to the createSetsForEdge() method, but instead of selecting edges, the method selects cells.

createSetsForFaces() method is utilized to create all face sets. Its operation is analogous to createSetsForEdges() and createSetsForCells().
createCrackInteraction() method is utilized to define the crack in the model. First, the method defines a crack seam with crackFlanks faces set. Second, the method defines the contour integral with dummy qVectors option. Crack parameters are read from the DataStr() class. Thus defined crack would probably give an error during analysis, due to the qVectors direction, which is chosen arbitrarily. However, the contourIntegral is re-defined with the proper parameters by the FEMcrackOrphanMesh() class. The purpose of this contourIntegral definition is to force Abaqus to retain some of the crack properties and geometric features during the export and import of the orphan mesh, which otherwise would be challenging to define.

createContactInteraction() method defines a contact interaction between the crack faces. The method defines a standard surface to surface contact interaction.

meshInstance() method is utilized to automate meshing of the model. It calls seedEdges(), assignMeshControls(), assignElementType() and generateMesh() methods.

assignMeshControls() method is utilized to assign different meshing techniques and element shapes to the cells of the model. Cells immediately surrounding the crack tip are assigned SWEEP meshing technique, with the default algorithm and WEDGE elements. Cells comprising the innerCylinder are assigned SWEEP meshing technique with MEDIAL_AXIS algorithm and HEX_DOMINATED elements. For the rest of the model, the default meshing technique is STRUCTURED is set by default by Abaqus and is not explicitly defined.

writeInputFile() method is utilized to write an input file from the constructed model, which is imported and processed by the FEMcrackOrphanMesh(). The method should be called after the model has been fully defined.

FEMcrackOrphanMesh() class

FEMcrackOrphanMesh() class is utilized to import a model from an input file and perform additional operations to complete and submit the model for analysis. It is a subclass of the BaseCrackMdb() class.

The FEMcrackOrphanMesh() class methods are as follows:

__init__() method is an augmentation of the BaseCrackMdb.__init__() method. It takes the DataStr() class as an input and in addition to the method from the superclass defines the self.mdb[sets] variable.

initializeAbaqusModel() method is utilized to create an Abaqus model, import an input file with named according to the modelName in the DataStr() class, assign it to the self.mdb["model"] variable and delete the generic Model-1. The method is a redefinition of the method with the same name from the superclass.
clearImportedModel() method is utilized to remove features that were employed in the FEMcrackMdb() class, but are no longer deemed necessary, rename necessary features, whose names were changed during the import and assign pointer variables to model features. The method calls deleteSets(), renameInteractions(), renamePart(), assignVariableToPart(), renameInstance() and assignVariableToInstance() methods.

deleteSets() method is utilized to delete the sets of the model. It reads the names of the sets from the DataStr(), changes the name strings to uppercase to match the modifications during the import and deletes them. By reading the set names from the DataStr() and not from the model sets repository, ensures that sets created by Abaqus for internal use are not deleted.

renameInteractions() method is utilized to correct the names of the crack, contact property, update history output requests and delete contact interaction. The method calls renameCrack(), renameContactProperty(), renameHistoryOutput(), updateHistoryoutput() and deleteContactInteraction() methods. The purpose of the renaming operations is to be able to refer to the features with the names that have been defined in the DataStr() class and to eliminate any confusion with feature names.

renameCrack() method is utilized to rectify the name of the crack, modified during the import, to correspond to the name defined in the DataStr() class. The method first identifies the crack feature of the model, reads the correct name from the DataStr() and sets the name of the feature to correspond with the one in the DataStr() class.

renameContactProperty() method is utilized to rectify the name of the contact interaction property. It functions in a similar fashion to the renameCrack() method.

renameHistoryOutput() method is utilized to rectify the name of the history output request. The method functions in a similar fashion to the renameCrack() and renameContactProperty() methods.

updateHistoryOutput() method is utilized to update the name of the contour integral in the history output request definition.

deleteContactInteraction() method is utilized to delete the standard contact interaction from the model.

renamePart() method is utilized to rectify the the name of the part, which has been modified during the import. The method functions in a similar fashion to the renameCrack() and renameContactProperty() and renameHistoryOutput() methods.

assignVariableToPart() method is utilized to assign the variable self.mdb[-orphanMeshPart] to the part of the model, which can be employed to point to the part at later stages. The method identifies the part of the model and assigns it to the specified variable.
renameInstance() method is utilized to rectify the name of the orphan mesh instance in the assembly of the model. The method functions in an analogous fashion as the renameCrack(), renameContactProperty(), renamePart() and renameHistoryOutput() methods.

assignVariableToInstance() method is utilized to assign self.mdb[orphanMeshInstance] to the instance of the orphan mesh part for future reference. The method is analogous to the assignVariableToPart() method.

createElementSets() method is utilized to create element sets, defined in the DataStr() class. The method reads the element set data from the DataStr() and calls the createInnerCylinderElementSet() with arguments a set name and the corresponding parameters to filter the required elements.

createInnerCylinderElementSet() method is utilized to select elements from the model by a bounding cylinder and organize them in a set. Set name and the parameters defining the bounding cylinder are passed to the method as input arguments.

deleteCentralWedgeElements() method is utilized to delete the WEDGE shape elements in the innerCylinder cells. During mesh transformation, when the ratio of the ellipse of the crack is either large or small, the WEDGE shape elements along the Z axis may become severely distorted and corrupt the mesh quality, which may lead the analysis to fail. Therefore, it is necessary these elements to be removed and the obtained hole closed. The method calls the selectCentralWedgeElements() to get the labels of the elements. Next the method calls the deleteElementsWithLabels() method, to which passes the element labels as an input argument.

selectCentralWedgeElements() method is utilized to select the WEDGE elements in the innerCylinder cells and return their labels. The method iterates through the element sets and and for each element calls the isWedgeElement method, passing the element and if isWedgeElement() returns True, stores the element label.

isWedgeElement() method is utilized to verify whether an element has a WEDGE shape and returns either True or False. The method compares the element.type attribute of the element to the element codes C3D15 and C3D6, which correspond to quadratic and linear WEDGE shape elements.

deleteElementsWithLabels() method is utilized to delete elements corresponding to the passed labels argument. The method first creates a sequence from the labels argument and then deletes the elements from the orphan mesh part.

createSetForExternalNodes() method is utilized to create a node set from nodes on the top, bottom and cylinder surface of the model, to which are assigned boundary conditions. The method iterates through all nodes of the model and assigns them to the node set if they meet the criteria. If (x, y, z) are nodal coordinates then the node to be assigned to the set,
either $|z| = 1/2h$, where $h$ is the containerHeight or $x^2 + y^2 = r^2$, where $r$ is the containerRadius.

**createSetFromInnerCylinderNodes()** method is utilized to create a two node sets containing the nodes on the innerCylinderHole wall. Each node set contains nodes on the same side of the crack plane, so that crack would not be affected by merging the nodes of each set. First the method calls the method **findInnerCylinderElementSetWedgeElementsRadius()**, which identifies the radius of the innerCylinderHole. Then the method calls the **createInnerCylinderNodeSetFromElementSet()** for each element set, which creates the two node sets.

**findInnerCylinderElementSetWedgeElementsRadius()** method is utilized to find and store to the DataStr() class the radius of the innerCylinder-Hole. If $(x, y, z)$ are coordinates of a node, the method iterates through the nodes of the element sets and finds the smallest value of $\sqrt{x^2 + y^2}$, adds to it the selectionTolerance value and stores it to the DataStr() as the innerCylinderHole radius.

**createInnerCylinderNodeSetFromElementSet()** method is utilized to assign nodes from and element set, that are no further from the Z axis than the innerCylinderHole radius to a node set. The method iterates through each node of the element set and compares the $\sqrt{x^2 + y^2}$ to the inner-CylinderRadius, where $(x, y, z)$ are the node coordinates. If $\sqrt{x^2 + y^2}$ is smaller than the innerCylinderHole radius, the node label is stored. Eventually a node set is created from the stored node labels.

**applyMeshTransformation()** method is utilized to transform the orphan mesh according to the specified transformation type and parameters. The method iterates through all nodes of the model and according to the transformation type calls **calculateEllipticCoordinates()**, **calculateSimpleScaleCoordinates()** or **calculateAdvancedScaleCoordinates()**, passing a node as an argument. Then the calculated coordinates are passed to moveNode() method, which moves the node accordingly.

**calculateEllipticCoordinates()** method is utilized to perform the elliptic transformation of the node coordinates from $(x, y, z)$ to $(x_e, y_e, z_e)$. The method reads the crack parameters $(a$ and $b$) and the transformationAxis from the DataStr(). Next it calculates a new set of coordinates by:

$$
x_e = x \sqrt{1 + \frac{a^2 - b^2}{x^2 + y^2}}
$$

$$
y_e = y
$$

$$
z_e = z
$$

if the transformationAxis is set to X. If the transformationAxis is set to
Y, the new set of coordinates are:

\[
\begin{align*}
  x_e &= x \\
  y_e &= y \sqrt{1 + \frac{a^2 - b^2}{x^2 + y^2}} \\
  z_e &= z
\end{align*}
\]

Finally the method returns the new set of coordinates.

calculateSimpleScaleCoordinates() method is utilized to perform the simpleScale transformation of the node coordinates from \((x, y, z)\) to \((x_s, y_s, z_s)\). The method reads the crack parameters \((a \text{ and } b)\) and the transformationAxis from the DataStr(). Next it calculates a new set of coordinates by \(x_s = (a/b) x\), \(y_s = (a/b) y\) and \(z_s = z\), if the transformationAxis is set to X. If the transformationAxis is set to Y by \(x_s = x\), \(y_s = (a/b) y\) and \(z_s = z\). Finally the method returns the new set of coordinates.

calculateAdvancedScaleCoordinates() method is utilized to perform the simpleScale transformation of the node coordinates from \((x, y, z)\) to \((x_a, y_a, z_a)\). The method reads the crack parameters \((a \text{ and } b), \text{ transformationAxis} \text{ and crackInitialRadius}\) from the DataStr(). Next it calculates two scale factors \(\text{expansionFactor} = a/\text{crackInitialRadius}\) and \(\text{contractionFactor} = b/\text{crackInitialRadius}\). According to the transformationAxis the new set of coordinates is \(x_a = x \cdot \text{expansionFactor}\), \(y_a = y \cdot \text{contractionFactor}\) and \(z_a = z\) if transformationAxis is X, or \(y_a = y \cdot \text{expansionFactor}\), \(x_a = x \cdot \text{contractionFactor}\) and \(z_a = z\) if transformationAxis is Y.

moveNode() method is utilized to edit the current coordinates of a node to a new set of coordinates, which effectively changes the nodes position in space. The method takes two input arguments a node and the new coordinates.

closeInnerCylinderHole() method is utilized to close the innerCylinderHole by translating and merging the nodes of the innerCylinderWall. The method merges the nodes from the two node sets one at a time, to ensure that the nodes from one crack flank are not merged with the nodes from the other. The procedure is implemented in a loop that calls moveInnerCylinderHoleNodesToPlane() method with the nodes from the set as argument and then calls mergeNodes(), passing the nodes of the set again. Finally the method calls regenerateAssembly() to update the model.

moveInnerCylinderHoleNodesToPlane() method is utilized to translate nodes, passed as input argument. The method reads the transformationAxis and depending on whether the it is X or Y translates the nodes to the XZ or YZ plane.

mergeNodes() method is utilized to merge nodes, passed as input argument, within a distance from each other, defined as mergingTolerance in the DataStr() class.
redefineCrackInteraction() method is utilized to delete the old crack definition and create a new one. The crack extension direction is defined by the *CRACK.NORMAL* option.

getExternalNodes() method returns nodes, which would be assigned boundary conditions.

getCrackContainerInstance() method returns the instance of the orphan mesh model.

**BaseXFEMcrackMdb() class**

*BaseXFEMcrackMdb()* class is a subclass of the *BaseCrackMdb()* and is a superclass for *XFEMcrackMdbMP()* , *SimpleXFEMcrackMdb()* and *XFEMtetCrackMdb()* classes.

The *BaseXFEMcrackMdb()* methods are as follows:

createCrackPart() method is utilized to create the crack geometry part. The part has one feature *BaseShell* part, opposed to *BaseShellRevolve*.

createCrackAndCrackDomainInstances() method is utilized to create instances from the crack geometry and crack domain parts. The method calls createInstance() method to which passes the name of the crack geometry part. Next, the method calls the createMergedPartInstance() method.

createCrackPartitionPart() method is utilized to create a shell part, used to partition the *crackDomain* part at the crack location to create internal edges in the *crackDomain* part. The part geometry is dependent on the crack type. Geometrically, its shape is represented by the intersection between the crack geometry and the crackDomain. For embedded crack its shape is identical to the crack geometry. For edge and surface crack types it is an ellipse sector.

createSets() method is utilized to create a faces and edges sets and a datum coordinate system, used as a reference for the boundary conditions. The method calls createSetsForEdges() and createSetsForFaces() methods.

createSetsForEdges() method is utilized to create edge sets, which will be assigned seeds. Set names and parameters of the *BoundingCylinder*, which is used to select the edges are read from the *DataStr()* class.

createSetsForFaces() method is utilized to create face sets. The method is analogous to the createSetsForEdges() method.

createXFEMcrack() method is utilized to define *XFEM* crack.

createFieldOutputRequest() method is utilized to define a request for the *PHILSM* field output variable. The *PHILSM* variable is used for visualization of the crack during the post processing of the Abaqus output database.
**deleteFieldOutputs()** method is utilized to delete all field output requests in the model.

**assignMeshControls()** method is utilized to assign meshing technique to the model according to the element type. If the assigned elements are tetrahedral, the meshing technique is set to FREE and if the assigned elements are hexahedral, the default meshing technique is used.

**seedEdges()** method is used to assign seeds to the edges. Edges in all edge sets are seeded by size. Seed parameters are read from the **DataStr()** class.

**createSetForExternalNodes()** method is utilized to create a node set including nodes, to which are assigned boundary conditions. The method is analogous to the **FEMcrackOrphanMesh.createSetForExternalNodes()** method.

**getExternalNodes()** method returns the nodes in the set defined by **createSetForExternalNodes()**.

**getCrackContainerInstance()** method returns the instance of the crackContainer.

**XFEMcrackMdbMP()** class

**XFEMcrackMdbMP()** class is a subclass of **BaseXFEMcrackMdb()**, which is a subclass of **BaseCrackMdb()** and, therefore, inherits the methods of the both classes. The inherited methods including those defined in the **XFEMcrackMdbMP()** class provide functionality to create an Abaqus model database for XFEM analysis type of an elliptic crack of either embedded, edge or surface type. The model type created by the class is **multiplePartitions**

**XFEMcrackMdbMP()** defines the following methods:

**createAllParts()** method is utilized to create all the necessary parts for the model. The method calls **createCrackPart()**, **createSolidParts()**, which creates the crackContainer part in this case and **createSmallContainer()** methods.

**createSmallContainer()** method is used to create the smallContainer part. The smallContainer part is a solid part, used for partitioning of the crackContainer. It defines a full or a sector of an elliptic cylinder, around the crack, so that the volume of interest can be meshed with smaller elements, than the rest of the model. The part is dependent both on the crack geometric parameters and type. The axes of cross section of the smallCylinder are offset from the crack ellipse axes. Both offset and height of the cylinder are defined in the **DataStr()** class.

**createInstancesForPartitioning()** method is utilized to create instances, which are merged to create mergedPart. The method calls **createInstance()** twice, passing the crackContainer part name the first time and smallContainer part name the second time.
createLoftsOnCrackDomain() method is utilized to create loft features in the *mergedPart*, which partitions it into cells to enable meshing of the geometry. Lofts are created between the cross section edges of the *smallContainer* and the circumferential edges of the *mergedPart*.

createPiePartitions() method has analogous functionality to the FEMcrackMdb.partitionContainerAsPie().

createSets() method is utilized to create face and edge sets. Edge sets are used for seeding the containing edges and face set is used in the XFEM crack definition. The method creates a datum coordinate system, which is used as a reference for the boundary conditions. The method calls createSetsForExteriorEdges(), createSetsByBoundingBox() and createSetsForFaces().

createSetsForExteriorEdges() method calls the BaseXFEMcrackMdb.-createSetsForEdges() method.

createSetsByBoundingBox() method is utilized to create edge set for the edges, created by partitioning with the *smallContainer*. Parameters for the selection of the edges are read from the DataStr() class.

assignMeshControls() method redefines the BaseXFEMcrackMdb.assignMeshControls() and has analogous purpose. If the assigned elements are tetrahedral the meshing technique is set to FREE and allowMapped setting is set to False and if the assigned elements are hexahedral, the meshing technique is set to STRUCTURED.

**XFEMtetCrackMdb() class**

The **XFEMtetCrackMdb()** class is a subclass of BaseXFEMcrackMdb(), which is a subclass of BaseCrackMdb(). The **XFEMtetCrackMdb()** class is utilized to create an Abaqus model database for XFEM analysis type of an elliptic crack of either embedded, edge or surface type. The model created by the class is of crackPartition type.

**XFEMtetCrackMdb() class**

- **createAllParts()** method is utilized to create the parts of the model. It calls the methods createSolidParts() and createCrackPart. If the crack type is not embedded, the method also calls createCrackPartition().

- **createInstancesForPartitioning()** method is utilized to create instances from a selection of parts, which will comprise the mergedPart. If the crack type is embedded the method calls createInstancesFromAllParts(), which instances all available parts in the model. If the crack type is not embedded, however, the method creates instances of the crackPartition and crackDomain parts, by calling the createInstance() method twice and passing the crackPartition and crackDomain part names as arguments.

**XFEMsimpleCrackMdb() class**

The **XFEMsimpleCrackMdb()** class is a subclass of the BaseXFEMcrackMdb() class. The **XFEMsimpleCrackMdb()** class is utilized to create an
Abaqus model database for *XFEM* analysis type of an elliptic crack of either embedded, edge or surface type. The model created by the class is of *simple* type.

**XFEMsimpleCrackMdb()** class defines the following methods:

- **createAllParts()** method is utilized to create the parts of the model. The method calls **createSolidParts()** and **createCrackPart()**.

- **assignSectionToCrackDomain()** method is utilized to create assign section to the crackDomain part.

### 3.6.4 ReadOdb() class

**ReadOdb()** class is utilized to read and extract the history output data from an Abaqus output database of any combination of the model, analysis and crack types. The class takes one argument a **DataStr()** class, which defines the corresponding output database and to which the extracted and processed results are stored for further analysis.

The **ReadOdb()** class defines the following methods:

- **__init__()** method initializes the class, assigns the **DataStr** input argument to a variable and initializes two internal variables: temporary storage and a pointer variable to the output database.

- **openOdb()** method is utilized to open an output database, defined in the **modelName** variable of the **DataStr()** class. The opened output database is assigned a variable for future reference.

- **createOdbViewport()** method is utilized to create a viewport to display the output database.

- **readHistoryOutputs()** method is utilized to assign the repository with the history output data to a temporary storage variable.

- **extractValuesFromHistoryOutput()** method is utilized to read and store value for point of each contour of the stress intensity factors and coordinates of the data point to the temporary storage. First, the method reads the number of contours that have been requested in the model definition, from the **DataStr()** class and initializes the temporary storage to match the number of contours and the stress intensity factors. Then for each dataKey in the history output repository calls the methods **assignSIFvalue()** and **assignCoordinates()** methods. A dataKey corresponds to a value in the repository and contains information identifying the value. To identify what type is the value the dataKey is checked against certain string patterns and the value is assigned to the appropriate place in the temporary storage.

- **assignCoordinates()** method is utilized to assign (store) the value corresponding to the passed dataKey, if the dataKey corresponds to a coordinate value.

- **assignSIFvalues()** method is utilized to assign (store) the value corresponding to the passed dataKey, if the dataKey corresponds to a stress intensity factor and contour number.
The `calculateBetas()` method is used to calculate the angle $\beta$ of a point on the crack and the $X$ axis. The method iterates through the stored coordinates and calls `calculateCrackPointAngle` for each set of coordinates and then stores the angle to the temporary storage.

The `calculateCrackPointAngle()` method is utilized to calculate the $\beta$ angle and return it.

The `sortData()` method is utilized to remove duplicate data and sort the extracted stress intensity factors and data point coordinates according to the $\beta$ angle of the point.

The `add360DegreeDataPoint()` method is utilized to create one additional point at the end of each of the sorted sequences for stress intensity factors, angles, and coordinates for `embedded` crack type. The value of the point is equal to the value of the first point for all sequences except for the $\beta$ angles, where it is equal to the first $\beta$ angle in the sequence + 360 degrees. The method internally checks the crack type and if it is not `embedded` skips the procedure.

The `averageSortedSIFs()` method is utilized to create a new sequence for each stress intensity factor, where every value is an average of the same data point over a specified range of the contours. The range of averaging is read from the `DataStr()` class.

The `writeResultsToDataStr()` method stores the sorted and processed values of the quantities to the `DataStr()` class.

The `getOdb()` method returns the pointer to the open output database.

The `closeOdb()` method closes the output database.

### 3.6.5 PersistentData() class

The `PersistentData()` class provides access to the `shelf` database for read, write and processing of the input data. The class takes two arguments, `scriptPath` – a string specifying the parent directory of the `db` directory, in which the `shelf` databases are stored, and `DataStr()` class. The class creates three `shelf` databases for each crack type and the active `shelf` depends on the crack type defined in the `DataStr()` class.

The `PersistentData()` defines the following methods:

- `__init__()` method is utilized to initialize internal variables for the class. It also calls the `setShelvePath()`

- `setShelvePath()` method is utilized to set path to the `shelf` databases. It takes input argument of the `scriptPath` format.

- `determineActiveShelve()` method is utilized to designate the full path to the active `shelf`, depending on the crack type.

- `setActiveShelve()` method is utilized explicitly set active `shelf`. The method takes one argument, which is the crack type.
readAll() method opens the active shelve reads the data from the shelve, closes the shelve and returns the data.

writeToDb() method is utilized to store data to the active shelve. First, the method calls prepareDataForShelving() to extract only the necessary data from the DataStr(). Next it reads the model name from the DataStr() and sets it as a key under which the data will be stored to the shelve. Finally it writes the data to the active shelve and closes the shelve.

prepareDataForShelving() method is utilized to organize the data, which will be stored. The method creates a dictionary with keys:

input – for input parameters from the DataStr()
reports – for reports data from the DataStr()
odb – for results data from the DataStr()

checkForDuplicates() method is utilized to verify, whether the active shelve has an entry with the same input parameters as the DataStr() argument designated at the class creation. The method opens the active shelve and iterates through its entries. If a duplicate is found, closes the shelve and returns True. Otherwise the method closes the shelve and returns False.

getDuplicates() method is utilized to return the keys of any identical to the DataStr() class. The method opens the active shelve, iterates through its entries and stores a sequence of keys of the duplicate entries. Then the method closes the shelve and returns the sequence.

readKey() method is utilized to read from the active shelve the data, associated with a key, which is passed as an input argument.

### 3.6.6 DbDataStr() class

The DbDataStr() class is designed as DataStr() class counterpart, which serves to store and operate on the data, when postprocessing the results. An instance of the DbDataStr() class is created as soon as data is read from the shelve database. The class is an abstraction layer of the BaseDataStr() and internally creates an instance of the BaseDataStr() class and most of its methods return a call to the identically called methods of the BaseDataStr(). Thus created the class hides unnecessary functionality, changes to BaseDataStr() will not break the DbDataStr() and it is straightforward to define new methods. The class takes one input argument, which is the data directly read from the shelve.

The following methods of the DbDataStr() class call the identically named methods of the BaseDataStr() and if the method takes an argument, it is passed with the method call: getAnalyticalResults(), getVisualizationResults(), getAnalysisType(), getSortedBetaAngles(), getSortedContoursIIFs(), writeErrorResults(), getErrorResults(), writeAnalyticalResults(), writeVisualizationResults(), getCrackType(), getMaterialProperties(), getCrackParameters(), getAnalysisParameters(), getModelName(), getDataStr().
In addition to the afore mentioned, the `DbDataStr()` class defines the following methods:

- `__init__` method creates an instance of the `BaseDataStr()` and calls the `setDataStr()` and passes the input argument
- `writeResultRequests()` method is utilized to write the requests from the graphical user interface to the data structure.
- `getResultRequests()` method returns the requests from the GUI.
- `setModelName()` method is utilized to overwrite the `modelName` value of the data structure.
- `calculateAveragedSIFs()` method is used to create a new sequence for each stress intensity factor and calculate the value at each data point as an average from a specified from the gui contour range.
- `getSortedAveragedSIFs()` method returns the sequences of the averaged stress intensity factors.
- `getSortedCrackCoordinates()` method returns a sequence of the sorted crack coordinates.

### 3.6.7 AnalyticalData classes

`AnalyticalData` classes are a family of classes, utilized to calculate and write to the `DbDataStr()` class analytical solutions for the stress intensity factors of the crack analysis in the `DbDataStr()` class. The `AnalyticalData` class family is composed of the following classes and functions:

- `AnalyticalData()` class
- `BaseAnalyticalExpressions()` class
- `EdgeCrackAnalyticalSolutions()` class
- `EmbeddedCrackAnalyticalSolutions()` class
- `SurfaceCrackAnalyticalSolutions()` class
- `miscFunctions` module, containing the following functions:
  - `Rfun()` — calculates $R$ value
  - `Qfun()` — calculates $Q$ value
  - `compEllipIntK()` — calculates complete Elliptic integral for $a$ and $b$ of a crack
  - `compEllipIntE()` — calculates complete Elliptic integral for $a$ and $b$ of a crack
AnalyticalData() class

AnalyticalData() class is utilized to read the crack parameters and user requests from the DbDataStr, call appropriate methods and functions to calculate the data and write the results back to the DbDataStr(). The class takes one input argument the DbDataStr() class.

The AnalyticalData() class defines the following methods:

__init__() method is utilized to initialize the required parameters and classes, which are called by the other methods of the class to calculate the analytical solutions. Depending on the crack type, the method initializes the EmbeddedCrackAnalyticalSolutions() class for embedded, SurfaceCrackAnalyticalSolutions() class for surface or EdgeCrackAnalyticalSolutions() class for edge crack type and assigns it to the self.expressions variable. Finally the method calls the initializeParameters() method of the initialized class.

calculateAnalyticalResults() method is utilized to calculate and store an analytical value for the requested stress intensity factors. The method iterates through the $\beta$ angles of the crack and for each $\beta$ calls the self.expressions.calculateExpressionForAngle() with $\beta$ as an input argument, which returns the calculated values.

writeAnalyticalResultsToDataStr() method is utilized to store the results obtained by calculateAnalyticalResults() to the DbDataStr.

calculateVisualizationResults() method is utilized to calculate analytical solutions with higher resolution (larger number data points) than the calculateAnalyticalResults(), which calculates analytical solutions for the beta angles of the analysis. The method calls the self.expressions.getSolutionsForVisualization(), which returns the results.

writeVisualizationResultsToDataStr() method is utilized to write the results obtained by calculateVisualizationResults() to the DbDataStr.

calculateErrors() method is utilized to compare the analysis results with the analytical results and calculate a quantitative estimate of the accuracy of the solution. The method calculates three types of estimates:

errors – analytical and analysis results are compared for each point of the crack and for each requested stress intensity factor. Each value is calculated as the absolute value of the difference between the analysis and the analytical solution divided by the maximum absolute value of all analytical and analysis values.

maxErrors – the maximum value of the errors for each stress intensity factor.

maxAbsoluteError – the maximum value of the maxErrors.

writeErrorsToDataStr() method is utilized to write the results obtained by calculateErrors() to the DbDataStr.
BaseAnalyticalExpressions() class

The BaseAnalyticalExpressions() class is a superclass for the EmbeddedCrackAnalyticalSolutions(), SurfaceCrackAnalyticalSolutions() and EdgeCrackAnalyticalSolutions() classes. The purpose of the class is to facilitate the subclasses, which provide methods only to evaluate the analytical solution at a point. The class takes two input arguments, DbDataStr() class and SIFkeys, which contain the names of the requested analytical values.

The BaseAnalyticalExpressions() defines the following methods:

__init__() method is utilized to initialize the required variables for the class.

calculateSolutionForAngle() method is utilized to calculate and return a dictionary containing the values for each of the requested stress intensity factors for a beta angle, which is passed as an input argument.

getSolutionsForVisualization() method is utilized to return the values for the requested stress intensity factors, calculated at high resolution. The method calls calculateSolutionsForVisualization() method and returns the calculated values.

calculateSolutionsForVisualization() method is utilized to calculate analytical solutions for the requested stress intensity factors at resolution of 1 degree and store the calculated values.

EmbeddedCrackAnalyticalSolutions() class

The EmbeddedCrackAnalyticalSolutions() class is a subclass of the BaseAnalyticalExpressions() class. It is designed to calculate analytical values of the stress intensity factors for embedded crack type.

The EmbeddedCrackAnalyticalSolutions() class defines the following methods:

initializeParameters() method is utilized to read crack and analysis parameters from the DbDataStr() class. The method also calls the Rfun(), Qfun(), compEllipIntE() and compEllipIntK() functions and stores the values.

k1() method is utilized to calculate the value of K1 for a beta angle, passed as input argument.

k2() method is utilized to calculate the value of K2 for a beta angle, passed as input argument.

k3() method is utilized to calculate the value of K3 for a beta angle, passed as input argument.

SurfaceCrackAnalyticalSolutions() class

The SurfaceCrackAnalyticalSolutions() class is a subclass of the BaseAnalyticalExpressions() class. It is designed to calculate analytical values of the stress intensity factors for surface crack type. At the time of writing an empty class.

71
**EdgeCrackAnalyticalSolutions() class**

The **SurfaceCrackAnalyticalSolutions()** class is a subclass of the **BaseAnalyticalExpressions()** class. It is designed to calculate analytical values of the stress intensity factors for *edge* crack type. At the time of writing an empty class.

### 3.6.8 XYPlotDataFromDbEntry() class

**XYPlotDataFromDbEntry()** class provides methods to create **XYPlotData** in Abaqus for specified parameters. The method takes an instance of the **DbDataStr()** class as input argument and reads the required data for the **XYPlotData** from it.

The **XYPlotDataFromDbEntry()** defines the following methods:

- **__init__()** method initialized the internal variables for the class.
- **createAnalyticalXYPlotData()** method is utilized to create **XYPlotData** for the analytical solutions. The method creates **XYPlotData** only for the specified stress intensity factors.
- **createAveragedXYPlotData()** method is utilized to create **XYPlotData** for the averaged analysis output data. The method creates **XYPlotData** only for the specified stress intensity factors.
- **createContourXYPlotData()** method is utilized to create **XYPlotData** for the analysis output data. The method creates **XYPlotData** only for the specified stress intensity factors and contours.
- **createErrorXYPlotData()** method is utilized to create **XYPlotData** for the errors between the analytical and averaged over a specified range analysis data.
- **createVisualizationXYPlotData()** method is utilized to create **XYPlotData** for the analytical solutions for visualization, which may appear smoother. The method creates **XYPlotData** only for the specified stress intensity factors.

### 3.6.9 VisualizationOdbFromDbEntry() class

**VisualizationOdbFromDbEntry()** class is utilized to create an Abaqus output database for visual representation of the stress intensity factors. Crack parameters and the analysis results are read from the **DbDataStr()** class, which is an input argument for the class. The geometric features of the visualization are rebuilt from the analysis output to create an accurate representation of the results.

The **VisualizationOdbFromDbEntry()** defines the following methods:

- **__init__()** method initialized the internal variables of the class, `self.data`, `self.odb` and `self.dataStr`. `self.odb` is utilized as a pointer for the newly created output database. `self.data` is utilized to store *vector* values, which define offsets of the rings of *shell* elements from the crack contour and node set names.
initializeAbaqusOdb() method is utilized to create a name for the visualization, create the output database and assign pointers to important components of the database.

initializeViewport() method is utilized to create a viewport for the output database and privileged plane for the compass.

setViewportViewingPoint() method is utilized to orient the view so that the view is isometric and overwrite the Iso view.

setDisplayedObject() method is utilized to set the viewport to display the output database for a specific step and frame.

createMaterial() method is utilized to create an Elastic material with parameters read from the DbDataStr().

createSections() method is utilized to create a TrussSection for the truss elements and HomogeneousShellSection for the shell elements.

createpart() method is utilized to create a part named crackVisualization.

createNodes() method is utilized to create the required nodes of the part. The method initializes a nodeData variable to store a tuple containing the node label and coordinates, for each node. The variable is passed to and returned by each method, called by createNodes(). This ensures that methods have a starting point, and thus node numbering is consistent. Furthermore, the variable contains the necessary information, and is utilized to create all nodes. The method calls the following methods:

createCrackFrontNodes() to create nodes for the truss elements, representing the crack front geometry and K3 values;

createK1Nodes() to create nodes for the shell elements, representing K1 values;

createK2Nodes() to create nodes for the shell elements, representing K2 values.

Finally the method creates the nodes from the information stored in nodeData.

createK1Nodes() method is utilized to create node data for vertically offset from the crack front nodes. The created nodal data corresponds to nodes of equal number to the crack nodes with constant offset. The method takes two input arguments, nodeSetName and nodeData. The nodeData is updated with a tuple containing node number and coordinates for each node. Finally, each node number, created by the method, is associated with the nodeSetName, which is passed as an input argument, in the self.odb variable.

createK2Nodes() method is analogous to createK1Nodes(), however, it is utilized to create node data for nodes in the crack plane, which lie on the normal to the crack ellipse. The method calls calculateCoordinatesK2EllipseNormal() to calculate the coordinates of a node.
calculateCoordinatesK2EllipseNormal() method is utilized to calculate coordinates for a node used to create shell elements. Node coordinates are calculated as to lie on the intersection between the normal to the crack ellipse at reference crack node and an ellipse with axes equal to the corresponding axes of the crack ellipse with added offset. The offset is proportional to the minor axis of the ellipse of the crack. The method takes two input arguments: coordinates and vector. The coordinates argument specifies the coordinates of the reference crack node. The vector is a multiple in the calculation of the offset value. Larger |vector| results in a larger offset. The sign of the vector specifies, whether the node coordinate is inside, if vector is negative, or outside, if vector is positive, of the crack ellipse.

calculateBeta() method is utilized to calculate and return the \( \beta \) angle of a line through a specified point and the \( X \) axis. The method takes the \((x, y)\) coordinates of the point.

createCrackFrontNodes() method is utilized to create node data for nodes, used to create the truss elements, representing the crack edge and K3 value. The method is analogous to createK2Nodes() and createK3-Nodes(), however, the node coordinates of the nodeData in this case, are identical to the crack node coordinates.

createElements() method is utilized to create the elements of the model. The method uses the elementData variable, which passes to methods, which calls to obtain the required data to create elements and elementCounter to keep consistent element numbers. Elements are created in three steps. Firstly the method calls createCrackFrontElements, to which passes elementCounter and gets elementData and elementCounter. Then the method uses the elementData to create the T3D2 truss elements and assigns them to an element set K3-Elements. Second, the method sets the elementData variable to an empty tuple and calls the methods createOuterK2() and createInnerK2(), to which passes the variables elementData and elementCounter. With the data from elementData, the method creates CPS4R shell elements and assigns them to an element set K2-Elements. Finally, the method sets the elementData variable to an empty tuple and proceeds as in the second step, however, the methods it calls are createUpperK1() and createLowerK1() and the set, to which the elements are assigned is K1-Elements.

createOuterK2() method is utilized to create the elements comprising the ring of shell elements in the crack plane and located on the outer side of the crack ellipse contour. The method reads the node labels for the required nodes from the self.odb and iterates through the number of elements, each time creating a new element data and counting the number of elements.

createInnerK2() method is analogous to createOuterK2() method, however, it creates element data for elements located inside the crack contour.

createUpperK1() method is analogous to the methods createOuterK2() and createInnerK2, however, it creates elements located on the cylinder with cross section the crack front and on the positive side of the \( Z \) axis.
createLowerK2() method is analogous to the createUpperK1() method, however, it creates elements on the negative side of the Z axis.

createCrackFrontElements() method is utilized to create element data for the truss elements of the crack front. The method iterates through the node numbers stored in the self.odb each time updating the elementData variable with the new element data and elementCounter with the number of the current element in the elementData. Finally the method returns the variables elementCounter and elementData.

createStepFrame() method is utilized to create a step and frame in the Abaqus output database, to which field data is associated.

createInstance() method creates an instance of the part in the output database.

createFieldOutput() method is utilized to prepare and associate the analysis results from the DbDataStr() for stress intensity factors of the crack to the corresponding nodes of the output database as field output. The field output created as SCALAR field and named SIF. It is created in the step and frame created by the createStepFrame() method.

setDefaultField() method is utilized to set the SIF field output as default.

saveOdb() method is utilized to save and close the output database. The database is saved in the Abaqus active directory.

reopenOdb() method is utilized to open the created output database for Abaqus to display it.

3.6.10 GUI classes

GUI classes is a name of a family of classes, designed to draw dialog boxes in Abaqus, read entries from the shelf database and process user input. GUI class family is comprised by: SelectDb(), AccessDb() and dbAccessDialogs_plugin() classes. The classes are registered to Abaqus by the command getAFXApp().getAFXMainWindow().getPluginToolset().registerGuiMenuButton() at the bottom of the dbAccessDialogs_plugin.py.

dbAccessDialogs_plugin() class

The dbAccessDialogs_plugin() class is a subclass of the AFXForm() abstract superclass. The AFXForm() class is provided by the Abaqus FOX-toolkit extension and provides the infrastructure to process form modes. The AFXForm() class defines generic methods, used by its subclasses, however, it might be necessary some of these methods to be customized in the subclass definition.

The dbAccessDialogs_plugin() class defines the following methods:

__init__ method is utilized to initialized the class, it takes one argument owner, which is of typeAFXGuiObjectManager and is passed automatically to the class. The method also defines the name of the selected by default shelf database, AFXGuiCommand, FXMAPFUNCs and calls the createKeywords() method. The name of the shelf database
is updated, depending on which radio button in the first dialog box is selected. The `AFXGuiCommand` invokes the `readFromDb()` function from the `executeDbAccessCommands.py` module with a *dictionary of keywords*, which convey the user input commands. A `FXMAPFUNC` is defined to invoke a corresponding method, when it catches a message, identified by its `ID` and generated by interaction with the dialog boxes.

The call to the `createKeywords()` method is to create the *keywords*, which convey requests from the user. Four `FXMAPFUNCS` are defined, three for the radio buttons of the first dialog box for selecting the crack type and one for catching the messages from the tree in the second dialog box.

`onTreeSelect()` method is called whenever a the corresponding `FXMAPFUNC()` catches a message, generated on user interaction with the tree of the second dialog box. The method sets the value of the `self.dataKeywords[dbKeys]` to a concatenated *string* of the selected items in the tree of the second dialog box. For the purpose the method iterates members of the variable `self.selectableTreeItemIDs` and checks if the item is selected. If the item is selected the corresponding database key is concatenated to the *string* value of the `AFXStringKeyword`.

`getModelName()` method is utilized to split the value of the `dbKeys` *keyword* and return the last member of the resulting tuple.

`onEmbeddedButton()` method is called whenever the corresponding `FXMAPFUNC()` catches a message, generated on user selecting the *Embedded* crack radio button. The method modifies the values of the `db` and `activeDb` to reflect the changes in the user interface. To create the value of the `db` value the method calls `getDbPath()`.

`onSurfaceButton()` method is analogous to the `onEmbeddedButton`, however, it is called when the user selects *Surface crack* radio button.

`onEdgeButton()` method is analogous to the `onEmbeddedButton` and `onSurfaceButton`, however, it is called when the user selects *Edge crack* radio button.

`createKeywords()` method is utilized to create a *dictionary of keywords* – `self.dataKeywords`, which is structured in the following way:

- `dbKeys` entry contains an `AFXStringKeyword` with value equal to a concatenated *string* of the selected `shelf` database keys from the tree of the second dialog box.
- `activeDb` entry contains an `AFXStringKeyword` with value equal to `shelf` database name, which is named identically to the crack type.
- `db` entry contains an `AFXStringKeyword` with value equal to the full path including the `shelf` name.
- `thisDir` entry contains an `AFXStringKeyword` with value equal to the value of the `thisDir` variable, a *string* with the directory containing the module of the class.
analytical entry contains a dictionary with keys K1, K2 and K3 associated with an AFXBoolKeyword. The value of the AFXBoolKeyword is altered in accordance to the state of the corresponding check box is marked in the Analytical data section of the second dialog box.

analytical entry is similar to the analytical, however, the analysis corresponds to the check boxes in the Analysis data section.

contoursToAverage entry contains an AFXStringKeyword with value equal to a concatenated string of the selected entries in the Contours to average section.

includeContours entry contains an AFXBoolKeyword associated with the Include contour data check box.

XYPlotData entry contains an AFXBoolKeyword associated with the XYPlotData check box.

VisualizationOdb entry contains an AFXBoolKeyword associated with the VisualizationOdb check box.

printData entry contains an AFXBoolKeyword associated with the printData check box.

getDbPath() method is utilized to create and return the directory path including the name of the active shelf database.

getFirstDialog() method is used internally to generate the first dialog box, by creating an instance of the class for the dialog box. The method returns the first dialog box instance.

getNextDialog() method is used internally to generate the following dialog box. The method takes one argument, an instance of the previous dialog box. For the purposes of the program, the method generates the second dialog box, creating an instance of the class for the second dialog box.

doCustomChecks() method is utilized to verify the validity of the selected options in the second dialog box. The method verifies that at least one shelf database key and entry of the Select contours to average is selected. Otherwise an error message is displayed.

okToCancel() method is used internally to close the dialog box.

SelectDb() class

The SelectDb() class is a subclass of the AFXDataDialog() class, which is a superclass for all dialog boxes, which collect data form the user. The SelectDb() defines the first dialog box, in which the user selects the crack type and respectively database, which will be the object of analysis for the second dialog box.

The class has only one method __init__(), which takes one argument an instance of the dbAccessDialogs_plugin() class.
**AccessDb() class**

The *AccessDb()* class is a subclass of the *AFXDataDialog()* class. The *AccessDb()* defines the second dialog box, in which the user selects the one or more database entries from the tree to analyze and designates the options for post processing. The tree in the dialog box is generated when an instance of the class is created and, therefore, reflects the current *shelf* database entries. The class takes two input arguments *dbAccessDialogs_plugin()* and the path to the active *shelf* database.

The *AccessDb()* class defines the following methods:

- **__init__()** method is utilized to initialize the dialog box. The method calls the *createWidgets()* method, which creates the widgets of the dialog box.

- **createWidgets()** method is utilized to create the layout of the dialog box and call methods to create the rest of the widgets. The method calls *createTreeWidget(), createDataOptionsWidgets(), createContoursOptionsWidgets()*, and *createOutputOptionsWidgets()*.

- **createOutputOptionsWidgets()** method is utilized to create the check box widgets:
  - *XYPlotData*
  - *Print Data structure*
  - *Visualization Odb*

- **createContoursOptionsWidgets()** method is utilized to create the *Separate contours* group box widget and the *Include contour data* check box widget.

- **createDataOptionsWidgets()** method is utilized to create check box widgets associated with the dictionary keys and corresponding *AFXBoolKeyword* of the *dataKeywords[analytical]* and *dataKeywords[analysis]* variables from the *dbAccessDialogs_plugin()* class. The number of the check box widgets and their names is controlled by the contents of the two variables.

- **createTreeWidget()** method is utilized to build the tree widget and creates its structure from the *shelf* database entries. The tree branches represent model parameter, progressively narrowing down the selection of possible models, down to the model name. The tree root entries are the analysis types *FEM* and *XFEM*, which are propagated with the available entries in the corresponding *shelf* database. All of the tree items are disabled and thus not selectable, with the exception of model names of the simulations that have been completed successfully. A record of the selectable tree entries is kept and utilized in the *dbAccessDialogs_plugin()* class to update the respective *dataKeywords[dbEntries]*. The method creates the *FEM* and *XFEM* tree entries, opens and iterates through the active *shelf* database calling the *createEntryID.createID()* function to create a unique *ID* for the entry and passes it to *createTreeItems()* method. Finally the method closes the *shelf* database.
createTreeItems() method is utilized to create the structure of the tree widget. The tree entries are organized in hierarchical pairs of parameter name entry, which groups its corresponding value entries available in the shelve database. Value entries are further propagated with hierarchical pairs until the model name is reached. To ensure, the consistency of the tree structure, each tree entry is assigned a unique key, which determines the parent/child relationship of the entries. The method iterates the sorted keys of the unique entry ID and calls createTreeEntry() once to create the parent parameter name entry and second time to create the value entry.

createTreeEntry() method is utilized to identify the parent of a new entry and create the entry under the parent. The entry is made selectable only if it is a modelName and its analysisSuccess variable is True. The method also keeps the record of selectable entries, which is utilized by the dbAccessDialogs_plugin() to keep track of the selected tree entries.

3.6.11 CreateID function family

The CreateID is a family of three functions in the createEntryID.py module, utilized to create a unique ID for a shelve database entry. The family is composed of three functions: createID(), createIDfem() and createIDxfem().

createID() function

The createID() function is called by the AccessDb.createTreeWidget() to create a unique ID for an entry from a shelve database. The function calls either createIDfem() or createIDxfem(), according to the analysis type of the model. Finally the function returns the unique ID.

createIDfem() function

The createIDfem() function generates a unique ID for a shelve database entry, based on the input parameters of the model. The function is designed to operate on input data for FEM analysis type. The ID is a dictionary with keys of format 01_parameter name for consistent sorting. The corresponding value is a combination of the parameters and their values. The dictionary key is utilized in the generation of the parent entries of the hierarchy and the values of the corresponding child entry.

createIDxfem() function

The createIDxfem() function is similar to the createIDfem(), however, it operates on input data for XFEM analysis type.

3.6.12 Execute gui commands functions

The execute gui commands is a family of functions, designed to call the required classes and methods, according to user requests. The functions are defined in the executeDbAccessCommands.py. The following functions are defined in the module: readFromDb(), prepareDbDataStr(), appendPath(), createXYPlots(), printData() and createVisualizationOdb().
readFromDb() function

The readFromDb() function is utilized to process the dataKeywords, which are passed as an input argument by the dbAccessDialogs.plugin(), when the Create button of the second dialog box is pressed. First, the method calls the function appendPath() to make the required modules available in the system path. Second, the method extracts the shelve keys from the dataKeywords/dbKeys/ string in a tuple. Next, the method iterates through the keys, reads the corresponding entry from the shelve database and calls the prepareDbDataStr() functions, which returns an instance of the DbDataStr() class. Then the instance is passed to createXYPlots(), printData() and createVisualizationOdb() if the corresponding dataKeywords are True.

prepareDbDataStr() function

The prepareDbDataStr() function is utilized to create an instance of the DbDataStr() class. The function takes three arguments as input data, which is the data read from the shelve database, requests, dictionary with the requests from the user and dbKey, which is the model name. Firstly the method creates an instance of the DbDataStr() class and then, calls the appropriate DbDataStr() class methods to write the requests and the dbKey value as a model name to the class.

appendPath() function

The appendPath() function is utilized to append the directory containing the required modules to the sys.path, which is a record of directories, where Abaqus searches during import.

createXYPlots() function

The createXYPlots() function is utilized to call the XYPlotDataFromDbEntry() and its methods to create XYPlotData for the passed instance of the DbDataStr() class.

printData() function

The printData() function is utilized to print the contents of the passed instance of the DbDataStr() class.

createVisualizationOdb() function

The createVisualizationOdb() function is utilized to create an Abaqus output database for visualization of the stress intensity factors from the passed DbDataStr() class instance. The method creates an instance of the VisualizationOdbFromDbEntry() class and calls the required methods to create the output database.

3.6.13 Main loop

The main loop name corresponds to modules defining functions to create and analyze a model database, extract the results from the output database and
write to the corresponding *shelve* database. Parameters for the model database are defined as range of values, which define a combination of input parameters. A function iterates through each configuration of input parameters and proceeds with the creation, analysis and results processing of the model. To avoid duplication of analyses and to be able to resume in case of an interruption, the *shelve* database entries are compared with the current input parameters before the creation of the model database. If a duplicate is detected the loop jumps to the next parameter configuration, until the configuration is unique for the *shelve* database. The **main loop** is comprised of four modules corresponding to each model type:

- **femCrackLoop.py** is utilized to create a *FEM* analysis type.
- **xfemCrackLoop.py** is utilized to create a *XFEM* analysis type and *crack-Partition* model type.
- **xfemCrackMPLoop.py** is utilized to create a *XFEM* analysis type and *multiplePartitions* model type.
- **xfemSimpleCrackLoop.py** is utilized to create a *XFEM* analysis type and *simple* model type.
Chapter 4

Results

4.1 Introduction

The program developed in the current project is capable of handling automated analysis of elliptic cracks. The only limitation is imposed by the available computational resources. The chapter covers the following points:

- element type comparison for FEM analysis type
- investigation of the optimal size of the cylinder
- mesh convergence analysis for FEM analysis type
- comparison of the mesh transformations for the FEM analysis type
- mesh and singularity radius convergence study for XFEM analysis type
- comparison of the accuracy of the models of XFEM analysis type
- comparison between FEM and XFEM results
- visualization of stress intensity factors

4.2 Procedure

The analysis of the results is performed in several steps. In each step of the analysis the optimum value of the analyzed parameter is selected and fixed for the consecutive step of the analysis, thus reducing the amount of parameters and narrowing the possible configurations to the optimal. As a starting point for the analysis the model of type crackNormal with elliptic mesh transformation is selected, for which is known to give the most accurate results.

1. Evaluation of the accuracy for each element type for FEM analysis type. Results of the stress intensity factors obtained for linear full integration, linear reduced integration, quadratic reduced integration and quadratic full integration elements are analyzed. Further, simulations are performed with the element type, which gives the smallest error.
2. Investigation of the optimal size of the cylinder containing the crack. Dimensions of the cylinder significantly influence the results for the stress intensity factors. In addition, the analytical solutions are for elliptic cracks in infinite medium. Therefore, it is crucial to estimate and limit the influence of the size of the cylinder on the final results.

3. Mesh convergence analysis for the FEM analysis, performed only for the selected element types.

4. Comparison of the mesh transformation types for otherwise identical meshes of the FEM analysis type.

5. Mesh and singularity radius convergence study for the different models of the XFEM analysis type. Only the optimal model dimensions, obtained are considered.

6. Comparison of the accuracy of the XFEM model types.

7. Comparison is made between FEM and XFEM results.

8. Review of the stress intensity factors visualization technique developed in the project.

9. Discussion of the results

4.3 Delimitations

Results in this chapter are generated for $\gamma = 30^\circ$ and $\omega = 60^\circ$, applied remote stress $\sigma = 100$. All the calculations for the FEM and XFEM analyses are performed for 5 contours. Values obtained by averaging over contours 2, 3 and 4 are presented in this chapter. This simplification streamlines the generation of the results, however, it may degrade the accuracy, although, this effect should be limited and manifest itself mostly as additional noise to the solution.

Apart from the averaging, no additional processing operations have been performed on the results, which have been read from the Abaqus output database.

4.4 Element type comparison

As a starting point for the analysis of the element accuracy are the four simulations in table 4.1. Direct comparisons of the stress intensity factors are shown in figure 4.1 for $K_I$, figure 4.2 for $K_{II}$ and figure 4.3 for $K_{III}$. Errors between the simulation and analytical values for the stress intensity factors are shown in figure 4.4 for $K_I$, figure 4.5 for $K_{II}$ and figure 4.6 for $K_{III}$.

Comparing the results from table 4.1, the maximum error is less than 14% and quadratic reduced integration element appears to give the most accurate results. Graphs, however, reveal that the quadratic elements introduce more noise in the solution. Nevertheless, the quadratic reduced integration elements are selected for the next step in the analysis.
Figure 4.1: Comparison of values for $K_I$ obtained for crackNormal model type with different element types and elliptic transformation

Figure 4.2: Comparison of values for $K_{II}$ obtained for crackNormal model type with different element types and elliptic transformation

Table 4.1: Models included in the element type study

<table>
<thead>
<tr>
<th>ID</th>
<th>mesh transform</th>
<th>element type</th>
<th>crack ratio</th>
<th>$K_I$</th>
<th>$K_{II}$</th>
<th>$K_{III}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1307331098dot91</td>
<td>elliptic</td>
<td>quadraticFI</td>
<td>3</td>
<td>6.79</td>
<td>6.55</td>
<td>12.62</td>
</tr>
<tr>
<td>1307363961dot35</td>
<td>elliptic</td>
<td>quadraticRI</td>
<td>3</td>
<td>6.36</td>
<td>5.92</td>
<td>12.45</td>
</tr>
<tr>
<td>1307329103dot72</td>
<td>elliptic</td>
<td>linearRI</td>
<td>3</td>
<td>5.39</td>
<td>13.41</td>
<td>11.76</td>
</tr>
<tr>
<td>1307328639dot16</td>
<td>elliptic</td>
<td>linearFI</td>
<td>3</td>
<td>8.77</td>
<td>8.89</td>
<td>13.65</td>
</tr>
</tbody>
</table>
Figure 4.3: Comparison of values for $K_I$ obtained for crackNormal model type with different element types and elliptic transformation

Figure 4.4: Comparison of errors for $K_I$ obtained for crackNormal model type with different element types and elliptic transformation
Figure 4.5: Comparison of errors for $K_{II}$ obtained for *crackNormal* model type with different element types and *elliptic* transformation

Figure 4.6: Comparison of errors for $K_{III}$ obtained for *crackNormal* model type with different element types and *elliptic* transformation
4.5 Analysis of the influence of the cylinder dimensions

The analysis of the influence of the cylinder dimensions is performed by comparison of the maximum error for each of the stress intensity factor for different cylinder dimensions. The total number of simulations includes all the configurations with cylinder height and radius of: 40, 80, 120, 200, 300, including the configuration of a cylinder with height and radius of 100. The results are shown in figure 4.7 for $K_I$, figure 4.8 for $K_{II}$ and figure 4.9 for $K_{III}$.

In addition, figure 4.10 illustrates the errors along the crack for the $K_I$ stress intensity factor, figure 4.11 for $K_{II}$ and figure 4.12 for $K_{III}$. Details about these simulations are presented in table 4.2.

Results prove the expected tendency of increased accuracy with increased dimensions of the cylinder. However, the dependence is not linear and is most pronounced for the height. Regarding the stress intensity factors, the trend is most pronounced for $K_I$, however, errors for $K_{II}$ and $K_{III}$ slightly increase after a certain point.

Finally, the optimal dimensions for a cylinder with a crack with $a = 30$ and $b = 10$ are:

\[
\begin{align*}
\text{height} &= 200 \\
\text{and} \\
\text{radius} &= 120
\end{align*}
\]
Figure 4.8: Convergence study for cylinder dimensions against the maximum errors for $K_{II}$

Figure 4.9: Convergence study for cylinder dimensions against the maximum errors for $K_{III}$
Figure 4.10: Comparison of errors for $K_I$ along the crack front for different cylinder dimensions

Figure 4.11: Comparison of errors for $K_{II}$ along the crack front for different cylinder dimensions

<table>
<thead>
<tr>
<th>ID</th>
<th>cylinder</th>
<th>max error [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>height</td>
<td>radius</td>
</tr>
<tr>
<td>1309805073dot26</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>1309792093dot37</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>1307363961dot35</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1309770498dot61</td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 4.2: Models in figures 4.10, 4.11 and 4.12
Figure 4.12: Comparison of errors for $K_{III}$ along the crack front for different cylinder dimensions

4.6 Mesh convergence analysis

In this section the influence of the mesh refinement is analyzed by comparing different mesh densities for linear reduced integration and quadratic reduced integration elements. Analysis is performed for model with dimensions defined in section 4.5 and elliptic mesh transformation.

4.6.1 Mesh convergence analysis with quadratic reduced integration elements

Models included in the study are presented in table 4.3. According to table 4.3 the maximum errors of the stress intensity factors are almost independent of the mesh density.

For more complete representation, graphs of the errors for the stress intensity factors for some extreme cases, along the crack front are shown in figure 4.13 for $K_I$, figure 4.14 for $K_{II}$ and figure 4.15 for $K_{III}$.

Although, maximum errors of the evaluated stress intensity factors is relatively constant, higher mesh density corresponds to a solution with less noise.
Table 4.3: Models with *quadratic reduced integration* elements included in the mesh convergence study

<table>
<thead>
<tr>
<th>ID</th>
<th>seeds</th>
<th>max error [%]</th>
<th>$K_I$</th>
<th>$K_{II}$</th>
<th>$K_{III}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1309848413dot98</td>
<td>5 5 3 5</td>
<td>failed analysis checks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309840073dot36</td>
<td>3 5 12 5</td>
<td>4.93 5.91 11.91</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309850324dot89</td>
<td>5 5 12 5</td>
<td>4.33 5.74 12.04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309842434dot45</td>
<td>5 3 3 5</td>
<td>failed analysis checks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309837868dot7</td>
<td>3 3 12 5</td>
<td>4.82 5.91 11.93</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309837555dot03</td>
<td>3 3 5 5</td>
<td>5.05 5.94 11.85</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309843793dot24</td>
<td>5 3 12 5</td>
<td>4.38 5.8 12.07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309837418dot02</td>
<td>3 3 3 5</td>
<td>failed analysis checks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309842855dot88</td>
<td>5 3 5 5</td>
<td>4.54 6.09 11.98</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309789759dot25</td>
<td>5 5 9 5</td>
<td>4.41 5.72 12.01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309839557dot28</td>
<td>3 5 5 5</td>
<td>5.17 6.01 11.84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309848923dot31</td>
<td>5 5 5 5</td>
<td>4.54 6.0 11.98</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309839345dot21</td>
<td>3 5 3 5</td>
<td>failed analysis checks</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.13: Comparison of errors for $K_I$ along the crack front for different mesh densities of *quadratic reduced integration* elements
Figure 4.14: Comparison of errors for $K_{II}$ along the crack front for different mesh densities of quadratic reduced integration elements

Figure 4.15: Comparison of errors for $K_{III}$ along the crack front for different mesh densities of quadratic reduced integration elements
4.6.2 Mesh convergence analysis with linear reduced integration elements

Models included in the study are presented in table 4.4. Contrary to the results in section 4.6.1, maximum errors of the stress intensity factors exhibit strong mesh density dependence, especially for the $K_{II}$ factor. For more complete representation, graphs of the errors for the stress intensity factors for some extreme cases, along the crack front are shown in figure 4.16 for $K_I$, figure 4.17 for $K_{II}$ and figure 4.18 for $K_{III}$.

In conclusion, the simulation with ID 1309835867dot25, table 4.4 provides the optimal balance between accuracy and mesh size, from the performed simulations.

Table 4.4: Models with linear reduced integration elements included in the mesh convergence study

<table>
<thead>
<tr>
<th>ID</th>
<th>seeds</th>
<th>Max error [%]</th>
<th>$K_I$</th>
<th>$K_{II}$</th>
<th>$K_{III}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1309835867dot25</td>
<td>5 5  5 5</td>
<td>3.85</td>
<td>12.75</td>
<td>11.6</td>
<td></td>
</tr>
<tr>
<td>1309835491dot63</td>
<td>5 3  15 5</td>
<td>3.48</td>
<td>13.68</td>
<td>11.26</td>
<td></td>
</tr>
<tr>
<td>1309835163dot74</td>
<td>5 3  3 5</td>
<td>5.56</td>
<td>14.23</td>
<td>12.9</td>
<td></td>
</tr>
<tr>
<td>1309834706dot86</td>
<td>3 3  12 5</td>
<td>9.16</td>
<td>32.28</td>
<td>16.3</td>
<td></td>
</tr>
<tr>
<td>1309836322dot15</td>
<td>5 5  15 5</td>
<td>3.31</td>
<td>13.89</td>
<td>11.12</td>
<td></td>
</tr>
<tr>
<td>1309835814dot17</td>
<td>5 5  3 5</td>
<td>5.53</td>
<td>14.17</td>
<td>12.74</td>
<td></td>
</tr>
<tr>
<td>1309835202dot98</td>
<td>5 3  5 5</td>
<td>4.02</td>
<td>12.74</td>
<td>11.75</td>
<td></td>
</tr>
<tr>
<td>1309835270dot46</td>
<td>5 3  12 5</td>
<td>3.52</td>
<td>13.49</td>
<td>11.28</td>
<td></td>
</tr>
<tr>
<td>1309834845dot13</td>
<td>3 5  3 5</td>
<td>4.47</td>
<td>25.13</td>
<td>12.22</td>
<td></td>
</tr>
<tr>
<td>1309834682dot92</td>
<td>3 3  5 5</td>
<td>7.05</td>
<td>26.93</td>
<td>13.14</td>
<td></td>
</tr>
<tr>
<td>1309834763dot64</td>
<td>3 3  15 5</td>
<td>9.26</td>
<td>31.98</td>
<td>16.27</td>
<td></td>
</tr>
<tr>
<td>1309835011dot44</td>
<td>3 5  15 5</td>
<td>11.01</td>
<td>34.24</td>
<td>17.4</td>
<td></td>
</tr>
<tr>
<td>1309834661dot92</td>
<td>3 3  3 5</td>
<td>3.34</td>
<td>24.22</td>
<td>12.43</td>
<td></td>
</tr>
<tr>
<td>1309834907dot25</td>
<td>3 5  12 5</td>
<td>10.88</td>
<td>34.14</td>
<td>17.43</td>
<td></td>
</tr>
<tr>
<td>1309834869dot43</td>
<td>3 5  5 5</td>
<td>8.49</td>
<td>29.03</td>
<td>14.0</td>
<td></td>
</tr>
<tr>
<td>1309835969dot28</td>
<td>5 5  12 5</td>
<td>3.35</td>
<td>13.72</td>
<td>11.15</td>
<td></td>
</tr>
</tbody>
</table>
Figure 4.16: Comparison of errors for $K_1$ along the crack front for different mesh densities of linear reduced integration elements.

Figure 4.17: Comparison of errors for $K_{II}$ along the crack front for different mesh densities of linear reduced integration elements.
4.7 Comparison between mesh transformations

4.7.1 Comparison between elliptic and simpleScale mesh transformations

The accuracy of evaluation of stress intensity factors obtained from models with elliptic and simpleScale mesh transformations is compared. The comparison is made with both linear reduced integration and quadratic reduced integration elements. For the comparison, cylinder dimensions and mesh configurations obtained in sections 4.5 and 4.6 are used. Models included in the comparison are presented in table 4.5. Graphs for the maximum errors are illustrated in figure 4.28 for $K_I$, figure 4.29 for $K_{II}$ and figure 4.30 for $K_{III}$.

Results from the comparison of a crack with aspect ratio 3 are shown in figure 4.19 for $K_I$, figure 4.20 for $K_{II}$ and figure 4.21 for $K_{III}$. For cracks with ratios 5 and 10, results from the comparison are shown in figures 4.22, 4.23, 4.24, 4.25, 4.26, 4.27.
Figure 4.19: Comparison of errors for $K_I$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 3

Figure 4.20: Comparison of errors for $K_{II}$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 3
Figure 4.21: Comparison of errors for $K_{III}$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 3

Figure 4.22: Comparison of errors for $K_I$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 5
Figure 4.23: Comparison of errors for $K_{II}$ for *elliptic* and *simpleScale* mesh transformation along the crack front for crack with aspect ratio of 5

Figure 4.24: Comparison of errors for $K_{III}$ for *elliptic* and *simpleScale* mesh transformation along the crack front for crack with aspect ratio of 5
Figure 4.25: Comparison of errors for $K_I$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 10

Figure 4.26: Comparison of errors for $K_{II}$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 10
Figure 4.27: Comparison of errors for $K_{III}$ for elliptic and simpleScale mesh transformation along the crack front for crack with aspect ratio of 10

<table>
<thead>
<tr>
<th>ID</th>
<th>mesh transform</th>
<th>element type</th>
<th>crack ratio</th>
<th>max error [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$K_I$</td>
<td>$K_{II}$</td>
</tr>
<tr>
<td>1309929194dot81</td>
<td>simpleScale</td>
<td>quadRI</td>
<td>3</td>
<td>4.99</td>
</tr>
<tr>
<td>1309932913dot67</td>
<td>simpleScale</td>
<td>linearRI</td>
<td>3</td>
<td>5.71</td>
</tr>
<tr>
<td>1309835867dot25</td>
<td>elliptic</td>
<td>linearRI</td>
<td>3</td>
<td>3.85</td>
</tr>
<tr>
<td>1309848923dot31</td>
<td>elliptic</td>
<td>quadRI</td>
<td>3</td>
<td>4.5</td>
</tr>
<tr>
<td>1309930698dot59</td>
<td>elliptic</td>
<td>linearRI</td>
<td>5</td>
<td>1.88</td>
</tr>
<tr>
<td>1309930186dot92</td>
<td>elliptic</td>
<td>quadRI</td>
<td>5</td>
<td>3.9</td>
</tr>
<tr>
<td>1309930764dot13</td>
<td>simpleScale</td>
<td>linearRI</td>
<td>5</td>
<td>11.76</td>
</tr>
<tr>
<td>1309930410dot0</td>
<td>simpleScale</td>
<td>quadRI</td>
<td>5</td>
<td>5.97</td>
</tr>
<tr>
<td>1309931867dot58</td>
<td>elliptic</td>
<td>linearRI</td>
<td>10</td>
<td>3.65</td>
</tr>
<tr>
<td>1309930836dot52</td>
<td>elliptic</td>
<td>quadRI</td>
<td>10</td>
<td>4.51</td>
</tr>
<tr>
<td>1309931955dot25</td>
<td>simpleScale</td>
<td>linearRI</td>
<td>10</td>
<td>28.07</td>
</tr>
<tr>
<td>1309931338dot52</td>
<td>simpleScale</td>
<td>quadRI</td>
<td>10</td>
<td>10.75</td>
</tr>
</tbody>
</table>

Table 4.5: Models included in the comparison of mesh transformations
Figure 4.28: Comparison of the maximum errors for $K_I$ for elliptic and simpleScale mesh transformations for crack with aspect ratios of 3, 5 and 10

Figure 4.29: Comparison of the maximum errors for $K_{II}$ for elliptic and simpleScale mesh transformations for crack with aspect ratios of 3, 5 and 10
The advancedScale mesh transformation in its present implementation proved to give unreliable results. The problem can be tracked to the merging on nodes of the innerCylinder. The procedure works reliably for elliptic and simpleScale mesh transformations, however, is not reliable for advancedScale. Therefore, results for this particular transformation are not considered and is not advisable to be used until the algorithm is fixed.
4.8 **XFEM results**

4.8.1 **Mesh and singularity radius convergence study**

The accuracy of the *XFEM* solution, in addition to mesh density, depends on the *singularity calculation radius*. It determines which elements, located radially in the vicinity of the crack front, would be included in the calculation of the singularity. Judging from the results, it may have a significant impact on the accuracy of the results.

The convergence study is carried out on models with dimensions determined in section 4.5 and crack aspect ratio of 3.

Results for the $K_{II}$ stress intensity factor have consistently been calculated with the opposite to the analytical solution sign and errors are in the vicinity of 200%.

**Convergence study for the crackPartition model**

The *crackPartition* model type is meshed with *linear tetrahedral elements*, which are generally perceived as inferior to hexahedral elements and require dense meshes to converge.

Models in the convergence study are presented in table 4.6. Results for $K_I$ stress intensity factors are shown in figure 4.31, $K_{II}$ in figure 4.32, $K_{III}$ in figure 4.33.

![Figure 4.31: Convergence study for crackPartition XFEM model for $K_I$ stress intensity factor](image)
Figure 4.32: Convergence study for crackPartition XFEM model for $K_{II}$ stress intensity factor

Figure 4.33: Convergence study for crackPartition XFEM model for $K_{III}$ stress intensity factor
<table>
<thead>
<tr>
<th>ID</th>
<th>singularity radius</th>
<th>seed size</th>
<th>max error [%]</th>
<th>$K_I$</th>
<th>$K_{II}$</th>
<th>$K_{III}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1309881638dot61</td>
<td>0.5</td>
<td>5</td>
<td>0.5</td>
<td>7.9</td>
<td>204.57</td>
<td>22.75</td>
</tr>
<tr>
<td>1309867375dot96</td>
<td>0.5</td>
<td>5</td>
<td>1</td>
<td>3.48</td>
<td>200.22</td>
<td>17.89</td>
</tr>
<tr>
<td>1309861189dot43</td>
<td>0.5</td>
<td>5</td>
<td>2</td>
<td>5.89</td>
<td>198.75</td>
<td>37.41</td>
</tr>
<tr>
<td>1309888005dot75</td>
<td>0.5</td>
<td>10</td>
<td>0.5</td>
<td>5.68</td>
<td>203.0</td>
<td>31.3</td>
</tr>
<tr>
<td>1309869569dot74</td>
<td>0.5</td>
<td>10</td>
<td>1</td>
<td>4.53</td>
<td>200.51</td>
<td>17.77</td>
</tr>
<tr>
<td>1309862320dot31</td>
<td>0.5</td>
<td>10</td>
<td>2</td>
<td>5.31</td>
<td>201.45</td>
<td>26.11</td>
</tr>
<tr>
<td>13098690615dot21</td>
<td>0.5</td>
<td>15</td>
<td>0.5</td>
<td>9.12</td>
<td>205.0</td>
<td>20.7</td>
</tr>
<tr>
<td>1309870282dot26</td>
<td>0.5</td>
<td>15</td>
<td>1</td>
<td>3.53</td>
<td>199.86</td>
<td>18.56</td>
</tr>
<tr>
<td>1309862604dot44</td>
<td>0.5</td>
<td>15</td>
<td>2</td>
<td>7.32</td>
<td>205.19</td>
<td>31.94</td>
</tr>
<tr>
<td>1309877851dot66</td>
<td>1</td>
<td>5</td>
<td>0.5</td>
<td>7.05</td>
<td>214.25</td>
<td>21.36</td>
</tr>
<tr>
<td>1309866047dot71</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>6.94</td>
<td>199.0</td>
<td>18.84</td>
</tr>
<tr>
<td>1309869569dot44</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>2.25</td>
<td>198.5</td>
<td>25.59</td>
</tr>
<tr>
<td>1309887212dot44</td>
<td>1</td>
<td>10</td>
<td>0.5</td>
<td>5.68</td>
<td>203.0</td>
<td>31.3</td>
</tr>
<tr>
<td>1309869302dot88</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>7.3</td>
<td>199.86</td>
<td>20.91</td>
</tr>
<tr>
<td>1309862816dot35</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td>5.33</td>
<td>202.21</td>
<td>24.83</td>
</tr>
<tr>
<td>1309890029dot56</td>
<td>1</td>
<td>15</td>
<td>0.5</td>
<td>8.33</td>
<td>212.85</td>
<td>22.79</td>
</tr>
<tr>
<td>1309870142dot52</td>
<td>1</td>
<td>15</td>
<td>1</td>
<td>7.8</td>
<td>199.23</td>
<td>20.36</td>
</tr>
<tr>
<td>1309862530dot48</td>
<td>1</td>
<td>15</td>
<td>2</td>
<td>4.49</td>
<td>206.65</td>
<td>30.46</td>
</tr>
<tr>
<td>1309874034dot54</td>
<td>2</td>
<td>5</td>
<td>0.5</td>
<td>18.55</td>
<td>210.53</td>
<td>94.47</td>
</tr>
<tr>
<td>1309864682dot98</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>7.45</td>
<td>203.82</td>
<td>19.76</td>
</tr>
<tr>
<td>1309859802dot52</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>5.39</td>
<td>197.16</td>
<td>26.31</td>
</tr>
<tr>
<td>1309886494dot28</td>
<td>2</td>
<td>10</td>
<td>0.5</td>
<td>5.68</td>
<td>203.0</td>
<td>31.3</td>
</tr>
<tr>
<td>1309869048dot44</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>8.8</td>
<td>209.58</td>
<td>22.39</td>
</tr>
<tr>
<td>1309862054dot23</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>5.84</td>
<td>199.68</td>
<td>25.65</td>
</tr>
<tr>
<td>1309889696dot37</td>
<td>2</td>
<td>15</td>
<td>1</td>
<td>20.78</td>
<td>208.35</td>
<td>94.26</td>
</tr>
<tr>
<td>1309869997dot09</td>
<td>2</td>
<td>15</td>
<td>1</td>
<td>8.96</td>
<td>207.7</td>
<td>21.95</td>
</tr>
<tr>
<td>1309862457dot59</td>
<td>2</td>
<td>15</td>
<td>2</td>
<td>7.62</td>
<td>104.18</td>
<td>30.81</td>
</tr>
<tr>
<td>1309870420dot11</td>
<td>5</td>
<td>5</td>
<td>0.5</td>
<td>18.55</td>
<td>210.53</td>
<td>94.47</td>
</tr>
<tr>
<td>1309862678dot81</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>7.45</td>
<td>203.82</td>
<td>19.76</td>
</tr>
<tr>
<td>1309859167dot16</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>5.39</td>
<td>197.16</td>
<td>26.31</td>
</tr>
<tr>
<td>1309885742dot85</td>
<td>5</td>
<td>10</td>
<td>0.5</td>
<td>5.68</td>
<td>203.0</td>
<td>31.3</td>
</tr>
<tr>
<td>1309868759dot33</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>8.8</td>
<td>209.58</td>
<td>22.39</td>
</tr>
<tr>
<td>1309861904dot84</td>
<td>5</td>
<td>10</td>
<td>2</td>
<td>5.84</td>
<td>199.69</td>
<td>25.65</td>
</tr>
<tr>
<td>1309888886dot84</td>
<td>5</td>
<td>15</td>
<td>0.5</td>
<td>20.78</td>
<td>208.35</td>
<td>94.27</td>
</tr>
<tr>
<td>1309869835dot18</td>
<td>5</td>
<td>15</td>
<td>1</td>
<td>8.98</td>
<td>207.7</td>
<td>21.95</td>
</tr>
<tr>
<td>1309858154dot86</td>
<td>5</td>
<td>15</td>
<td>2</td>
<td>7.62</td>
<td>204.18</td>
<td>30.81</td>
</tr>
</tbody>
</table>

Table 4.6: Models of type crackPartition included in the convergence study
Table 4.7: Models of type multiplePartitions included in the convergence study

<table>
<thead>
<tr>
<th>ID</th>
<th>singularity radius</th>
<th>seed size</th>
<th>max error [%]</th>
<th>$K_I$</th>
<th>$K_{II}$</th>
<th>$K_{III}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>general</td>
<td>container</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309986788dot06</td>
<td>0.5</td>
<td>10</td>
<td>1</td>
<td>37.37</td>
<td>221.32</td>
<td>23.25</td>
</tr>
<tr>
<td>1309984483dot97</td>
<td>0.5</td>
<td>10</td>
<td>2</td>
<td>37.37</td>
<td>221.32</td>
<td>23.25</td>
</tr>
<tr>
<td>1309987175dot79</td>
<td>0.5</td>
<td>20</td>
<td>1</td>
<td>56.44</td>
<td>327.06</td>
<td>33.83</td>
</tr>
<tr>
<td>1309985293dot77</td>
<td>0.5</td>
<td>20</td>
<td>2</td>
<td>56.44</td>
<td>327.27</td>
<td>33.83</td>
</tr>
<tr>
<td>1309986877dot41</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>31.39</td>
<td>208.08</td>
<td>28.56</td>
</tr>
<tr>
<td>1309984680dot36</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td>31.39</td>
<td>208.08</td>
<td>28.56</td>
</tr>
<tr>
<td>1309987100dot02</td>
<td>1</td>
<td>15</td>
<td>1</td>
<td>55.84</td>
<td>221.56</td>
<td>31.71</td>
</tr>
<tr>
<td>1309985152dot52</td>
<td>1</td>
<td>15</td>
<td>2</td>
<td>55.84</td>
<td>221.56</td>
<td>31.71</td>
</tr>
<tr>
<td>1309987202dot14</td>
<td>1</td>
<td>20</td>
<td>1</td>
<td>39.2</td>
<td>264.25</td>
<td>25.7</td>
</tr>
<tr>
<td>1309985340dot47</td>
<td>1</td>
<td>20</td>
<td>2</td>
<td>39.2</td>
<td>264.25</td>
<td>25.7</td>
</tr>
<tr>
<td>1309986968dot52</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>16.35</td>
<td>199.61</td>
<td>28.96</td>
</tr>
<tr>
<td>1309984881dot75</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>16.35</td>
<td>199.61</td>
<td>28.96</td>
</tr>
<tr>
<td>1309987134dot36</td>
<td>2</td>
<td>15</td>
<td>1</td>
<td>27.22</td>
<td>216.11</td>
<td>30.81</td>
</tr>
<tr>
<td>1309985220dot46</td>
<td>2</td>
<td>15</td>
<td>2</td>
<td>27.22</td>
<td>216.11</td>
<td>30.81</td>
</tr>
<tr>
<td>1309987226dot5</td>
<td>2</td>
<td>20</td>
<td>1</td>
<td>14.42</td>
<td>195.64</td>
<td>19.08</td>
</tr>
<tr>
<td>1309985392dot83</td>
<td>2</td>
<td>20</td>
<td>2</td>
<td>14.42</td>
<td>195.64</td>
<td>19.09</td>
</tr>
</tbody>
</table>

Convergence study for the multiplePartitions model

The multiplePartitions model is meshed with linear hexahedral reduced integration elements. The models in the convergence study are presented in table 4.7.
Figure 4.34: Convergence study for *multipleParts* XFEM model for $K_I$ stress intensity factor

Figure 4.35: Convergence study for *multipleParts* XFEM model for $K_{II}$ stress intensity factor
Convergence study for the *simple* model

The *simple* model is meshed with *linear hexahedral reduced integration* elements. The models in the convergence study are presented in table 4.8.
<table>
<thead>
<tr>
<th>ID</th>
<th>singularity radius</th>
<th>seed size</th>
<th>max error [%]</th>
<th>$K_I$</th>
<th>$K_{II}$</th>
<th>$K_{III}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1310135264dot81</td>
<td>1</td>
<td>4</td>
<td>28.13</td>
<td>215.56</td>
<td>28.67</td>
<td></td>
</tr>
<tr>
<td>1309907173dot28</td>
<td>1</td>
<td>5</td>
<td>12.27</td>
<td>210.25</td>
<td>37.22</td>
<td></td>
</tr>
<tr>
<td>1309907898dot52</td>
<td>1</td>
<td>10</td>
<td>39.02</td>
<td>167.13</td>
<td>33.73</td>
<td></td>
</tr>
<tr>
<td>1310133138dot24</td>
<td>2.5</td>
<td>4</td>
<td>21.58</td>
<td>210.3</td>
<td>33.56</td>
<td></td>
</tr>
<tr>
<td>1309906144dot41</td>
<td>2.5</td>
<td>5</td>
<td>16.94</td>
<td>204.39</td>
<td>42.01</td>
<td></td>
</tr>
<tr>
<td>130990836dot46</td>
<td>2.5</td>
<td>10</td>
<td>42.12</td>
<td>169.7</td>
<td>33.5</td>
<td></td>
</tr>
<tr>
<td>1310131219dot07</td>
<td>5</td>
<td>4</td>
<td>21.85</td>
<td>210.3</td>
<td>33.56</td>
<td></td>
</tr>
<tr>
<td>1309906107dot27</td>
<td>5</td>
<td>5</td>
<td>16.06</td>
<td>204.07</td>
<td>42.61</td>
<td></td>
</tr>
<tr>
<td>1309907767dot41</td>
<td>5</td>
<td>10</td>
<td>42.12</td>
<td>169.7</td>
<td>33.5</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.8: Models of type simple included in the convergence study

Figure 4.37: Mesh and singularity radius convergence for $K_I$
Figure 4.38: Mesh and singularity radius convergence for $K_{II}$

Figure 4.39: Mesh and singularity radius convergence for $K_{III}$
4.8.2 Comparison of the values and errors of the calculated stress intensity factors by XFEM

Accuracy of the crackPartition, multiplePartitions and simple model types is compared. Models included in the study are presented in table 4.9. The models are selected on the basis of the best accuracy for a given model type. The selection, however, does not guarantee, that the number of evaluations of the stress intensity factors is equal. Furthermore, the contrary is true and the number of points for evaluation of the stress intensity factors varies significantly with the different model types. Nevertheless, the comparison would be useful to give a general notion of what accuracy can be expected and what strategy may be give optimal results in future studies.

Comparison of the stress intensity factors along the crack front are illustrated in figure 4.40 for \( K_I \), figure 4.41 for \( K_{II} \) and figure 4.42 for \( K_{III} \). Errors of the stress intensity factors are illustrated in figure 4.43 for the \( K_I \), figure 4.44 for \( K_{II} \) and figure 4.45 for \( K_{III} \).

Stress intensity factors, calculated with XFEM show considerable noise and the maximum error value is not a precise measure for the overall accuracy. This is especially true for model 1309888886dot84, which has a very high peak, which leads to high value of the maximum error.

Figure 4.40: Comparison of the calculated values for \( K_I \) along the crack front for the different XFEM model types
Figure 4.41: Comparison of the calculated values for $K_{II}$ along the crack front for the different XFEM model types

Figure 4.42: Comparison of the calculated values for $K_{III}$ along the crack front for the different XFEM model types

Table 4.9: Models included in the comparison of the accuracy of the different XFEM model types

<table>
<thead>
<tr>
<th>ID</th>
<th>sing. radius</th>
<th>seed size</th>
<th>max error [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>general</td>
<td>crack</td>
<td>cont.</td>
</tr>
<tr>
<td>1309985392dot83</td>
<td>2</td>
<td>20</td>
<td>—</td>
</tr>
<tr>
<td>13099867375dot36</td>
<td>0.5</td>
<td>5</td>
<td>—</td>
</tr>
<tr>
<td>13098888886dot84</td>
<td>5</td>
<td>15</td>
<td>0.5</td>
</tr>
<tr>
<td>1309907173dot28</td>
<td>1</td>
<td>5</td>
<td>—</td>
</tr>
</tbody>
</table>
Figure 4.43: Errors of the calculated values for $K_I$ along the crack front for the different XFEM model types

Figure 4.44: Errors of the calculated values for $K_{II}$ along the crack front for the different XFEM model types
4.8.3 Comparison between FEM and XFEM results

The obtained results for the stress intensity factors from FEM and XFEM analyses are compared in this section. Comparison is performed for cracks with aspect ratios of 3, 5 and 10. The included models are of types crackNormal with elliptic mesh transformation and quadratic reduced integration elements and crackPartition with linear tetrahedral elements. The comparison is performed for models with cylinder dimensions determined in section 4.5, regardless of the crack aspect ratio. Therefore, it could be expected that the comparison performed for crack aspect ratio of 3 to be the most representative.

The comparison is limited to $K_I$ and $K_{III}$ stress intensity factors, as the results for $K_{II}$ from the XFEM analysis are calculated with opposite signs.

Comparison for crack aspect ratio of 3 $K_I$ is illustrated in figure 4.46, for $K_{III}$ in figure 4.47 and errors along the crack front in figure 4.48. For crack with aspect ratio of 5, $K_I$ is illustrated in figure 4.49, $K_{III}$ in figure 4.50 and errors in figure 4.51. For crack with aspect ratio of 10, $K_I$ is illustrated in figure 4.52, $K_{III}$ in figure 4.53 and errors in figure 4.54. Finally, models included in the comparison are listed in tables 4.10 and 4.11.

Results of the comparison prove that the XFEM and FEM models have comparable accuracy for $K_I$ and $K_{III}$ stress intensity factors. The decrease in accuracy of the $K_I$ and $K_{III}$ for the XFEM analysis for crack aspect ratios of 5 and 10 could be due to a requirement for a cylinder with larger dimensions to better represent the infinite medium. The mesh transform, however, increases the dimensions of the cylinder, which may have influence, which is not accounted for. Regarding the $K_{II}$ values calculated with XFEM have the opposite sign.
SIF $-2.155 \times 10^{2}$ $-1.606 \times 10^{2}$ $-1.058 \times 10^{2}$ $-5.103 \times 10^{1}$ $+3.779 \times 10^{0}$ $+5.859 \times 10^{1}$ $+1.134 \times 10^{2}$ $+1.682 \times 10^{2}$ $+2.230 \times 10^{2}$ $+2.778 \times 10^{2}$ $+3.326 \times 10^{2}$ $+3.874 \times 10^{2}$ $+4.423 \times 10^{2}$

Step: step−1
Primary Var: SIF
Deformed Var: not set   Deformation Scale Factor: not set
SIF visualization

Figure 4.46: Comparison between the calculated values for $K_I$ by FEM and XFEM along the crack front for crack with aspect ratio of 3

Figure 4.47: Comparison between the calculated values for $K_{III}$ by FEM and XFEM along the crack front for crack with aspect ratio of 3

<table>
<thead>
<tr>
<th>ID</th>
<th>crack ratio</th>
<th>seeds</th>
<th>max error [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>czm</td>
<td>czr</td>
</tr>
<tr>
<td>1309948923dot31</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1309930186dot92</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1309930836dot52</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 4.10: FEM models included in the comparison of the accuracy with XFEM models
Figure 4.48: Comparison between the calculated errors for $K_{III}$ by FEM and XFEM along the crack front for crack with aspect ratio of 3

Figure 4.49: Comparison between the calculated values for $K_I$ by FEM and XFEM along the crack front for crack with aspect ratio of 5

<table>
<thead>
<tr>
<th>ID</th>
<th>crack ratio</th>
<th>seeds general</th>
<th>max error [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>crack</td>
<td>$K_I$</td>
</tr>
<tr>
<td>1309870282dot26</td>
<td>3</td>
<td>15</td>
<td>3.53</td>
</tr>
<tr>
<td>1310240764dot87</td>
<td>5</td>
<td>15</td>
<td>7.0</td>
</tr>
<tr>
<td>1310240980dot74</td>
<td>10</td>
<td>15</td>
<td>7.86</td>
</tr>
</tbody>
</table>

Table 4.11: XFEM models included in the comparison of the accuracy with FEM models
Figure 4.50: Comparison between the calculated values for $K_{III}$ by FEM and XFEM along the crack front for crack with aspect ratio of 3

Figure 4.51: Comparison between the calculated errors for $K_{III}$ by FEM and XFEM along the crack front for crack with aspect ratio of 5
Figure 4.52: Comparison between the calculated values for $K_I$ by FEM and XFEM along the crack front for crack with aspect ratio of 10.

Figure 4.53: Comparison between the calculated values for $K_{III}$ by FEM and XFEM along the crack front for crack with aspect ratio of 10.
Figure 4.54: Comparison between the calculated errors for $K_{111}$ by FEM and XFEM along the crack front for crack with aspect ratio of 10

<table>
<thead>
<tr>
<th>ID</th>
<th>crack ratio</th>
<th>analysis type</th>
<th>model type</th>
<th>figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1309848923.31</td>
<td>3</td>
<td>FEM</td>
<td>crackNormal</td>
<td>4.55</td>
</tr>
<tr>
<td>1309869569.74</td>
<td>3</td>
<td>XFEM</td>
<td>crackPartition</td>
<td>4.56</td>
</tr>
<tr>
<td>1310240764.87</td>
<td>5</td>
<td>XFEM</td>
<td>crackPartition</td>
<td>4.57</td>
</tr>
<tr>
<td>1310240980.74</td>
<td>10</td>
<td>XFEM</td>
<td>crackPartition</td>
<td>4.58</td>
</tr>
<tr>
<td>1309907898.52</td>
<td>3</td>
<td>XFEM</td>
<td>simple</td>
<td>4.59</td>
</tr>
</tbody>
</table>

Table 4.12: Visualization of models

4.9 Visualization of the stress intensity factors

In this section are presented visualizations of the stress intensity factors for cracks with aspect ratios of 3, 5 and 10 and FEM and XFEM analyses. Models are listed in table 4.12 and illustrated in figures 4.56, 4.58, 4.57, 4.55 and 4.59. The visualization can also be utilized as a verification tool. For instance, figure 4.59 is a visualization of a XFEM simple model type with seed size of 10. The visualization, clearly illustrates the accuracy of the approximation.
Figure 4.55: Visualization of the 1309848923dot31 FEM model with crack aspect ratio 3

Figure 4.56: Visualization of the 1309869569dot74 XFEM model with crack aspect ratio 3
Figure 4.57: Visualization of the 1310240764dot87 XFEM model with crack aspect ratio 5

Figure 4.58: Visualization of the 1310240980dot74 XFEM model with crack aspect ratio 10
Figure 4.59: Visualization of the $1309907898dot52$ XFEM model with crack aspect ratio 3
Chapter 5

Conclusion

5.1 Introduction

The present project is an attempt to facilitate the analysis of elliptic cracks by providing a framework, which is capable of generating, analyzing, extracting the calculated values from the output database, writing the results to a custom shelve database and visualizing the results.

The program automates the described process and the created shelve database, which is a valuable knowledge base containing calculations for cracks with different parameters.

5.2 Summary of results

From the results in chapter 4 can be concluded that the crackNormal model type with quadratic reduced integration elements and elliptic mesh transformation would give the most consistent results. Comparable accuracy can also be achieved also with linear reduced integration elements, however, they should be used with meshes with higher density. Noise in the FEM solution can be reduced by increasing the mesh density, a viable option is also using linear reduced integration elements with a fine mesh.

XFEM can also be used successfully to analyze elliptic cracks, however, at the moment of writing, Abaqus has only an XFEM implementation with linear elements. Results prove that the accuracy of the XFEM solution is less consistent than the accuracy of the conventional FEM performed with quadratic reduced integration elements and has considerably more noise. Therefore, a dense mesh is generally required for accurate evaluation of stress intensity factors. In addition, with the increase of the crack aspect ratio the number of elements increase in the mesh, resulting in a more computationally expensive solution.

The mesh independent formulation of XFEM is a considerable advantage, however, to improve accuracy, use of dense mesh in the vicinity of the singularity is strongly advisable.

The visualization technique proposed can also be especially convenient as a diagnostic tool for verification of the XFEM approximation of the crack geometry as shown in figure 4.59.
5.3 Implications for practice and recommendations

The developed program in the project significantly facilitates modeling and analysis of embedded cracks. In addition, the developed visualization technique considerably improves representation of stress intensity factors by mapping the values to coordinates in three dimensional space.

Both FEM and XFEM model databases can be employed in a larger analysis by means of submodeling.

In addition, the program can be of utility for convergence studies for optimal mesh and analysis technique for a particular case. The convergence study can be executed on the model databases created by the program and then the results applied as a guidance in analysis of larger models.

Furthermore, the program can be utilized to facilitate the selection of optimal combination of seed size and singularity radius for an XFEM analysis, by performing a convergence study, similar to the one discussed in section 4.8.1 and designing a mesh for the analysis with the obtained parameters.

Use of results of the present project and the developed program are a good starting point for further analyses and fashion it is used would mostly depend on the specific analysis. However, my personal recommendation is to automate the analysis as feasible, the amount of parameters is vast and extensive analysis without automation may prove particularly challenging and unfeasible if performed manually. For instance, even creating a graph from the history output of the stress intensity factors for an XFEM analysis, without automation may prove quite challenging.

5.4 Implications for further development

The program for the project in its current state should provide a sound foundation for further development. Furthermore, it has been designed with scalability in mind and can be extended in a variety of ways.

5.4.1 Modeling automation

A major feature, would be an automated procedure to analyze cracks by submodeling, which would facilitate significantly parametric studies and optimization.

5.4.2 Results processing and optimization

One possibility for further functionality is the development of algorithms to further analyze results and find parameter configurations, which meet a certain criteria.

5.4.3 Other functionality

A more trivial development is to include results for J-integral and other fracture mechanics parameters.
5.5 Conclusion

In conclusion, the project attempted to create a self-contained and automated framework, built upon Abaqus software for modeling, analysis, results storage and visualization of elliptic cracks. As a result, the framework allows to analyze stress intensity factors, calculated by FEM and XFEM and compare their values with analytical solutions. The obtained results, comparisons between analysis types and convergence studies are presented in chapter 4. Furthermore, results prove, that XFEM is a promising development for stress intensity factors, although, the issue with opposite sign for $K_{II}$ has not been resolved.

The introduced visualization technique, would come as a useful representation and diagnostic tool in analyzing XFEM crack geometry approximation, as well as to improve representation of stress intensity factors or other fracture mechanics parameter.
Bibliography


