4 Implementation

The core of this research is the delivery of an implementation that is capable of simulating the propagation of hydraulic fractures and their intersection with existing fractures. Different codes in different languages were implemented and put together in a suite named XFEMHF. In this chapter all the implementation steps are presented explained. A more detailed description is made of the subroutines that are plugged in a finite element solver, which is the software Abaqus, by Simulia (2014).

First, the Abaqus software is presented, with focus to the user subroutines tool. Then, after an overview of every step of the XFEMHF code, each of the subroutines are explained in detail. An explanation on the type of elements and their integration procedures is also given and, finally, the limitations of the implementation are discussed.

4.1. Abaqus Software

4.1.1. General Description

Abaqus is a widely used commercial software, both in industry and academia, mainly known for its adaptability to very different kinds of numerical problems. In its essence, it is a solver for linear system of equations which allows the incorporation of non-linear problems with different physics in various types of finite elements.

Within a wide variety of modules and add-ons, the Abaqus finite element suite includes a graphical interface for input, monitoring of simulations and output interpretation (Abaqus/CAE) and three modules: Abaqus/Standard, a general-purpose finite element program, Abaqus/Explicit, an explicit dynamics finite element program and Abaqus/CFD, a general-purpose computational fluid dynamics program (Simulia, 2014). For the sake of simplicity, from this point the word Abaqus is meant to represent the module used in this work, which is Abaqus/Standard v6.14.

Abaqus offers an extensive diversity of element types and material models, applicable to different physical analyses: mechanical, pore hydraulic, thermal, electrical, electromagnetic and acoustic. Some of the available elements also allow coupled simulations between two or more of the previously mentioned physics.

Although it offers a wide variety of functionalities, also allows users to integrate their own subroutines in the calculation, which expand even more the software capabilities. More than 50 available user subroutines have many different functions, such as definition of complex constitutive models (p. ex. CREEP, HARDINI, UDMGINI, UMAT), definition of complex boundary conditions (p. ex. DFLOW, DLOAD, DISP), definition of constraints (p. ex. MPC, UMESHMOTION), definition of elements (p. ex. UEL and UELMAT) and management with external applications (UEXTERNALDB).

Abaqus solves non-linear problems by breaking the simulation into a number of time increments and finds the approximate equilibrium configuration at the end of each time increment. Using the Newton method, it often takes Abaqus several iterations to determine an acceptable solution to each time increment (Simulia, 2014). The calculations may be subdivided in:

- Steps: define an analysis procedure or loading. Different loads, boundary conditions, analysis procedures, and output requests can be used in each step,
- Increments: are part of a step. In nonlinear analyses each step is broken into increments so that the nonlinear solution path can be followed. The size of each increment may be fixed by the user or automatically chosen by Abaqus,
- Iterations: are an attempt at finding an equilibrium solution in an increment. If the model is not in equilibrium at the end of the iteration, Abaqus tries another iteration.

4.1.2. XFEM in Abaqus

Abaqus presents different techniques or elements to simulate discontinuities in a FEM model. Within the XFEM, which is called in the Abaqus Documentation as an “enriched feature”, the software differentiates between stationary and propagating cracks. Considering the focus of this research, in this chapter only the coupled hydro-mechanical element with XFEM for propagating cracks is detailed.

Abaqus implements the XFEM using the Phantom Node technique (Song, Areias and Belytschko, 2006), which was based on a previous work by Hansbo and Hansbo (2004). This technique considers the duplication of the mesh elements, being the duplicated nodes called “Phantom Nodes”, represented in Figure 4.1 by hollow circles. Moreover, additional nodes, known as “Edge-Phantom Nodes” (red triangles in Figure 4.1), allow the representation of the fluid pressure inside the fracture. Prior to damage initiation only one copy of the element is active. Upon damage initiation the displacement and pore pressure degrees of freedom associated with the corner phantom nodes are activated and both copies of the element are allowed to deform independently, pore pressures are allowed to diffuse independently, and the created interface behaviour is enforced with a traction-separation cohesive law. The pore fluid pressure at the top and bottom faces of the fracture are interpolated from the pore pressure degrees of freedom at the corner real nodes and phantom nodes. The difference with the fracturing fluid pressure (interpolated at the edge-phantom node) is the driving force that controls the leakage of fracturing fluid into the porous medium (Zielonka *et al.*, 2014).

The fracture geometry is mathematically described by the Level-Set Method, which assumes that two signed distance functions per node are generally required to describe a crack geometry (Simulia, 2014).

The propagation criterion may be based on stress or strain state, which interpolated to the crack tip or computed in the element ahead of the crack tip. Its direction is set to have perpendicular direction to the minimum principal stress in the tip region.

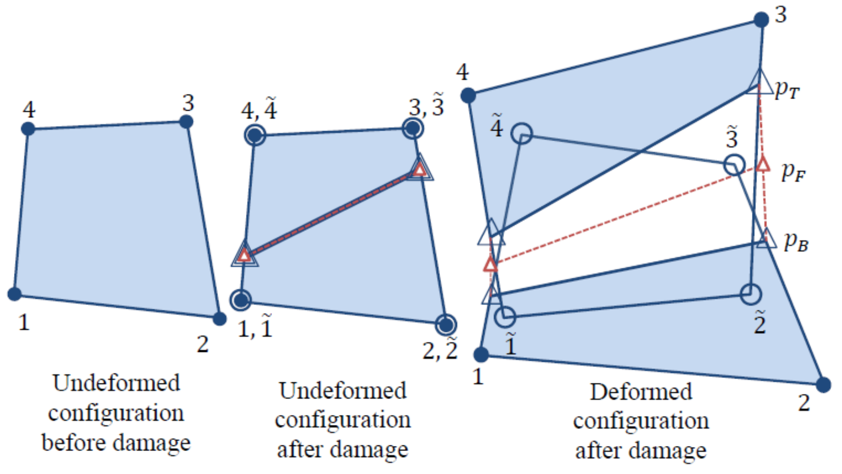


Figure 4.1 – Implementation of the XFEM with "corner" and "edge" phantom nodes (Zielonka *et al.*, 2014)

Areias and Belytschko (2006) showed that the kinematic decomposition in the Phantom Node method is equivalent to the one in the extended finite element method (XFEM) by Moes and Dolbow (1999) and Belytschko and Black (1999), having the advantage of being of easier implementation in a standard FEM code. On the other hand, it turns the implementation of partially cracked elements cumbersome.

The Abaqus XFEM elements have had a very positive effect of facilitating the use of this technique both in industry and academia (see Chapter 2). However, these cannot be used in the scope of this research, once Abaqus refers that XFEM elements cannot be intersected by more than one fracture (Simulia, 2014). Therefore, the built-in Abaqus XFEM elements are not part of this research, being substituted by the use of user elements coded in subroutines.

4.1.3. Abaqus User Subroutines

To achieve this thesis's proposed goals, two Abaqus user subroutines were used. The two user subroutines that manage and organize the workflow composed by most of the code written for this research are presented next. Figure 4.2 shows the flow of an Abaqus calculation and when each of the subroutines are called.

UEXTERNALDB

Though very simple, this is an extremely helpful subroutine when the user needs external procedures to be run during the simulation. This user subroutine is called once at the beginning of the analysis, at the beginning of each increment, at the end of each increment, and at the end of the analysis (in addition, the user subroutine is also called once at the beginning of a restart analysis). In addition to the number of the step, number of the increment and the time interval information, the variable LOP given to the user defines in which phase of the calculation (step, increment, etc.) the subroutine is being called. This way, it is possible to manage the calls to:

- Read the input files at the beginning of the calculation,
- Other processes or subroutines at the beginning/end of each increment/step
- Prepare output files at the end of each step or the calculation.

UEL

The User ELement subroutine gives the user the freedom to define any type of element topology and which governing equations are considered in that element. The user defines in the input the number of nodes of the element and their position. Furthermore, the input must also define the degrees of freedom, up to 30, that are attributed to each node.

Each time element calculations are required, i.e. for every element in every iteration, the UEL subroutine is called. Then, the code must perform all the calculations that are appropriate for the topology and the physics of the element. The subroutine must deliver the jacobian matrix of the Newton-Raphson method (AMATRX), the right-hand-side vector of the overall system of equations (RHS) and an array containing the values of the eventual solution-dependent state variables associated with the element (SVARS).

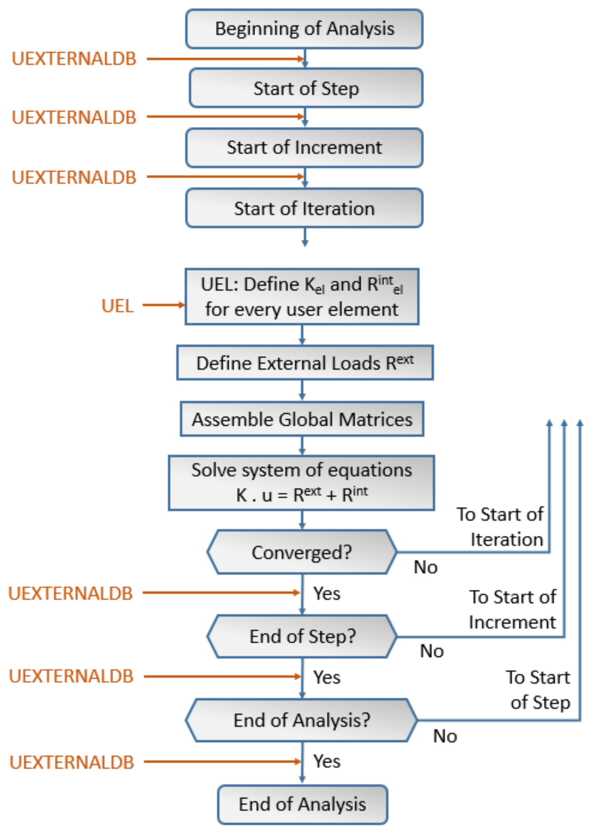


Figure 4.2 – Calls of user subroutines within the flow in Abaqus

4.2. XFEMHF code

4.2.1. Overview

As usual in research focused in code implementation, different software or codes were used to achieve the proposed goals.

A known limitation of the Abaqus graphical interface (Abaqus/CAE) is that it does not support the definition and visualization of user elements. Although not being the main focus of the research, the input and output tools are also essential. In a first step, during implementation, as they make the code validation easier. In a second step, because they guarantee that other users run simulations without need of advanced knowledge about the background process, allowing further contributions to the research topic.

The whole process of preparing, running and analysing a simulation may be described in three main steps, as seen in Figure 4.3.

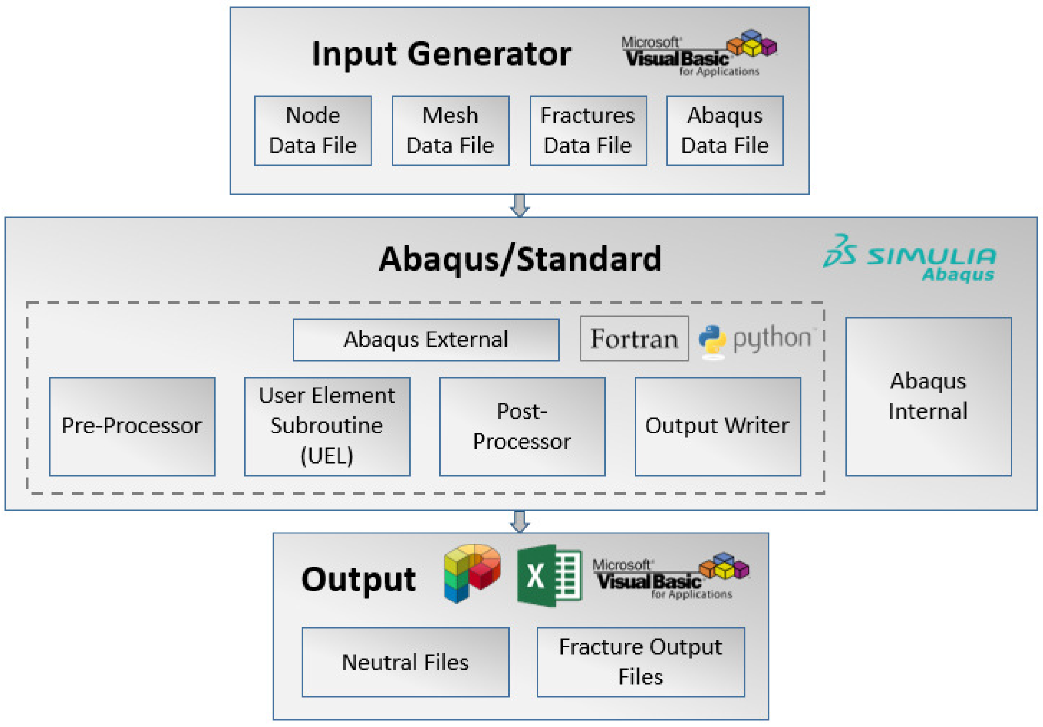


Figure 4.3 – Main steps of a simulation

Input generator

Four input files must be defined for every simulation: node data, mesh (topological) data, fracture data and Abaqus regular input data. These files are automatically generated by a code developed in VBA – Visual Basic for Applications.

Abaqus internal and external subroutines

The core code of this research is written in Fortran and compiled by Abaqus/Standard; therefore, this Chapter focus specifically in this step. The XFEM formulation is implemented in the external Abaqus User Subroutines, while several processes that are common with the standard FEM, such as the global stiffness matrix assembling or the convergence checker, are left to be done by the Abaqus internal subroutines. In specific cases, which will be referred in Chapter 4.2.2.2, a Python Script is used to interpret and adapt results during the simulation.

Output

The results may be visualized in the software Pos3D (Carvalho, Martha and Filho, 1997) which reads a Neutral File type generated during the simulation. For some specific variables along the fractures, a code developed in VBA interprets the fracture results and properly shows them in MS Excel graphs.

4.2.2. Abaqus algorithm

4.2.2.1. General algorithm

As mentioned before, the core of the implementation of this research is the Abaqus + User subroutines algorithm, i.e. the intermediate step of Figure 4.3. A more detailed flow of this algorithm is shown in Figure 4.4, where the boxes with dashed outlines represent the parts of the simulation where the code implemented in this research is called.

Starting by doing some Abaqus internal checks, the simulation then runs a user subroutine that reads all the input files generated in the first step of Figure 4.3 and allocates the auxiliary matrices and vectors in memory. Next, a geometry pre-processor is run. Based on the mesh and initial fracture information provided in the input files, this processor attributes all the values related with enrichment functions to the nodes and integration points of the model. Further information about the geometry pre-processor is present in Chapter 4.2.3. Then, the loops over steps, increments and iterations start. At the beginning of a new increment, the simulator checks if propagation occurred at the end of the previous increment – evidently, this does not happen at the first increment of the first step of the simulation. If propagation occurred, then new enriched degrees of freedom are active and the pre-processor is run, in order to update all the enrichment related data in the newly-propagating segments.

The loop over the iterations starts with a loop over all mesh elements. For each element the UEL subroutine computes the Jacobian matrix, the right-hand-side and the state variables are computed. This procedure is explained in more detail in Chapter 4.2.4. After assembling all the external forces defined in the input, the internal functions of Abaqus use the element matrices to assemble the global matrices and solve the linear system of equations. If convergence is not attained, another iteration is run.

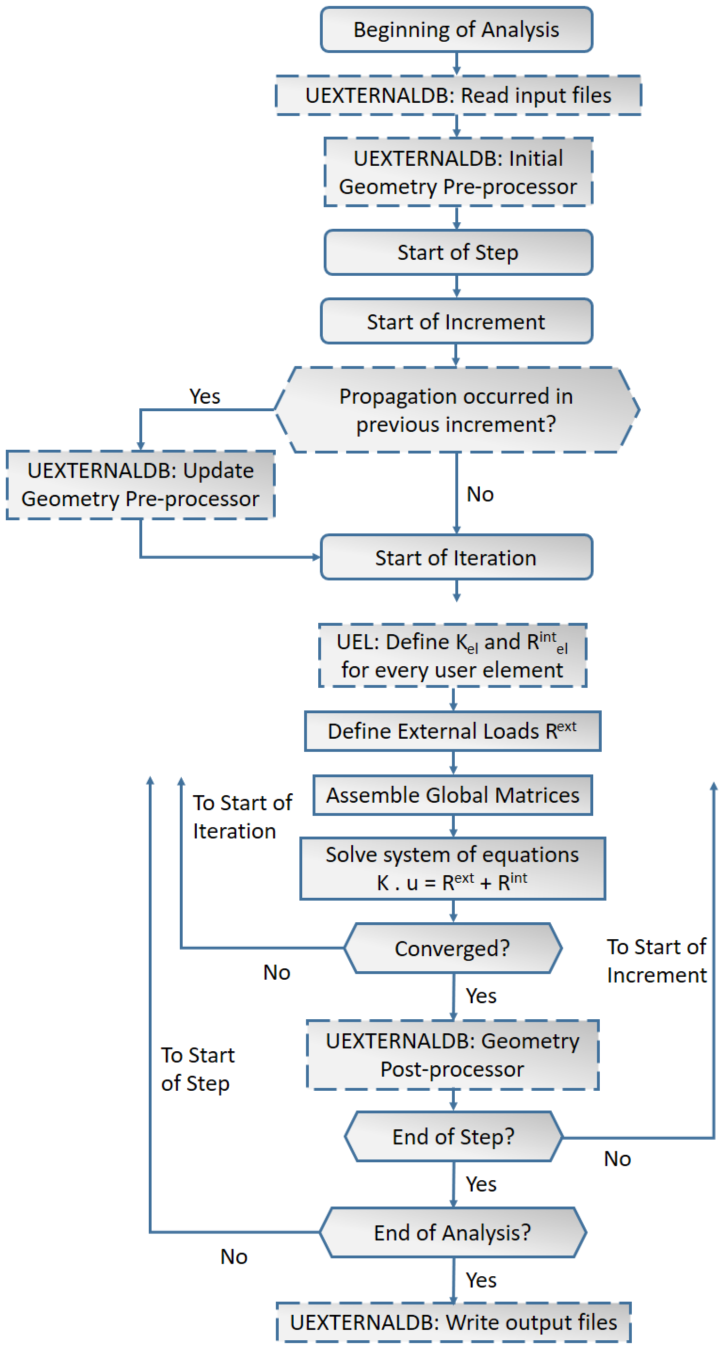
When the solution converges, the increment finishes and a fracture geometry post-processor is called. This procedure checks if propagation criteria are meet anywhere in the model. In case propagation occurs, the direction and length of the new fracture segments are also computed. A more detailed explanation of this procedure is delivered in Chapter 4.2.5. When the loop over the increments restarts, the coordinates of the new fracture segments are used by the pre-processor to place them in the mesh and compute their enrichment data, which will be used in the next increment.

Finally, when the last increment of the last step converges, the simulations finishes by writing all Abaqus output files and also the user output files that are interpreted in the third step of Figure 4.3.

4.2.2.2. The specific case of in-situ stress state

In most geotechnical events involving hydraulic and natural fractures the overburden cannot be neglected. Furthermore, it is widely known that the in-situ stress state extremely affects the way fractures behave and propagate. Consequently, this effect must also be considered in the developed simulator.

The in-situ stress state occurs due to various phenomena that happened during the geological history of the layer, such as overburden, tectonic movements and metamorphism. The modelling of all these effects to attain a correct in-situ stress state is extremely difficult. For that reason, that modelling is usually disregarded when only short periods of time (in a geological time scale) are to be simulated, being substituted by the input definition of an initial stress state. Therefore, the consideration of in-situ stresses in Abaqus is not straightforward, given that an initial stress field must be simulated as an equilibrium state which is the result of the gravitational fields and the model's boundary conditions. Due to the use of a User Element Subroutine (UEL), Abaqus cannot interpret the in-situ stresses from the input file and consequently a specific algorithm has to be defined.



Beginning of Analysis
UEXTERNALDB: Read input files
UEXTERNALDB: Initial Geometry Pre-processor
Start of Step
Start of Increment
Propagation occurred in previous increment?
Yes
UEXTERNALDB: Update Geometry Pre-processor
No
Start of Iteration
UEL: Define K_{el} and R^{int}_{el} for every user element
Define External Loads R^{ext}
Assemble Global Matrices
To Start of Iteration
Solve system of equations $K . u = R^{ext} + R^{int}$
Converged?
No
Yes
To Start of Increment
UEXTERNALDB: Geometry Post-processor
To Start of Step
End of Step?
No
Yes
End of Analysis?
No
Yes
UEXTERNALDB: Write output files

Figure 4.4 – Flow of a XFEMHF simulation – dashed outlines represent coded subroutines and continuous represent Abaqus internal routines

The implemented code contemplates the following steps, in order to consider in-situ stresses in the model (see Figure 4.5):

- On a first Abaqus analysis, all degrees of freedom of the mesh are considered to be fixed (i.e. boundary conditions set to zero). The in-situ stresses are applied to the whole model as internal stresses and one calculation step is run. The obtained reactions in every degree of freedom are output to a specific file and the simulation finishes;

- A Python script compiled by Abaqus is then run. This script translates the reactions in the previous mentioned output file to a new format, which is readable by Abaqus;
- A second Abaqus analysis is run. In the first step of this simulation, all the reactions obtained in the first simulation are applied to all degrees of freedom. Together with the internal stresses applied previously, this will guarantee that the simulation starts in equilibrium with zero displacement and the defined in-situ stress. Then, all the steps are defined as in a regular simulation, with all the loads and boundary conditions that the user requests.

Due to Abaqus limitations, this simulation must be run in two separated analyses. If this process was run in one single analysis, the Abaqus preparation subroutines would search for the files with the reactions generated by the Python script at the beginning of the analysis. As these files are generated after the geostatic step of the simulation, the preparation subroutines would deliver an error and Abaqus would abort before starting computations.

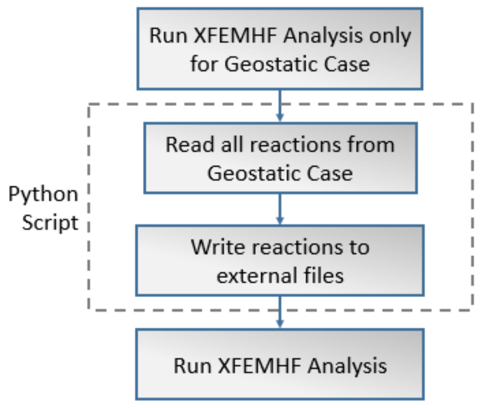

Run XFEMHF Analysis only for Geostatic Case
Python Script
Read all reactions from Geostatic Case
Write reactions to external files
Run XFEMHF Analysis

Figure 4.5 – General flow of a XFEMHF simulation with initial stress state

4.2.3. Fracture geometry pre-processor

The fracture geometry pre-processor runs at the beginning of the simulation and then updates data at the beginning of each increment that follows a propagation event, as seen in Figure 4.4. It should be noted that fractures do not change geometry during the iterations of an increment, so there is no need to run this procedure inside the increment loop.

The goal of this procedure is to define the enrichment data for the mesh, so the element matrices are built accordingly. The following information is delivered by the pre-processor:

- Which nodes and elements in the mesh are enriched, and which fracture is associated with that enrichment
- Enrichment function values at every enriched node
- Local coordinates and weight of integration points in enriched elements
- Enrichment function values at every integration point of the enriched elements
- Position of all fracture segments
- Local coordinates and weight of all fracture integration points
- Value of the jump function at every fracture integration point
- Fracture segments direction and length

At the first run, which can be called the "general definition stage", the pre-processor has the flow presented in Figure 4.6. After the reading of all the input data, each set of enrichment functions (H_i, H_{ii}, H_{iii}, etc.) is attributed to the initial fractures. Then, all the nodes and elements that are affected by fractures have their enrichment degrees of freedom activated. It must be noted that, because specific tip enrichments are not being used, the representation of the tip is achieved by deactivating the enrichment in the nodes that belong to the element border that is touched by the tip.

In sequence, all the elements that are cut by fractures are divided into sub-domains, allowing definition of the position of the integration points, as explained in Chapter 4.2.7.2. Next, the enrichment functions in every enriched node and integration points of enriched elements are computed. Although different types of enrichment functions are allowed in this implementation, only the signed level set function (H) is used in the simulations, as stated in Chapter 3.3.2.

Following, the fracture pressure degrees of freedom are attributed to the nodes, using the methodology described in Chapter 4.2.6. Then, the fracture integration points are placed and their weights attributed. Finally, the geometry of the fractures segments is stored, after computing their lengths and directions.

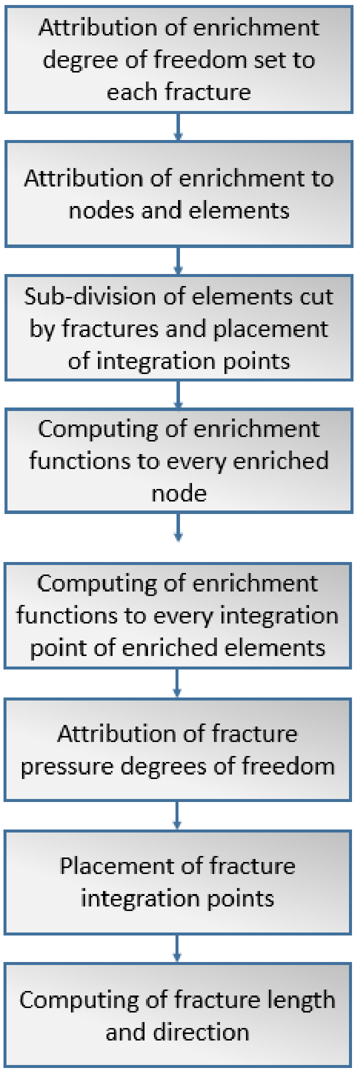

Attribution of enrichment degree of freedom set to each fracture
Attribution of enrichment to nodes and elements
Sub-division of elements cut by fractures and placement of integration points
Computing of enrichment functions to every enriched node
Computing of enrichment functions to every integration point of enriched elements
Attribution of fracture pressure degrees of freedom
Placement of fracture integration points
Computing of fracture length and direction

Figure 4.6 – General flow of the general definition stage of the pre-processor

The update stage run the same procedure for the enrichment data in the newly fractured elements and nodes, while the data in elements and nodes that had no change is kept the same.

A special procedure is considered in the geometry pre-processor in specific cases, when the continuous region is impermeable. As leak-off does not occur and no injection or fixed pressure points exist inside the fracture, there is no border or point for fluid to leave or enter it. Therefore, because the fluid is incompressible, the fracture cannot have any volumetric deformation to comply with the continuity equation. This way, all the displacements between the natural fracture faces may be highly compromised before intersection with hydraulic fractures occur. A way to overcome this problem is to define a criterion of fracture pressure activation, which allows the fracture pressure degrees of freedom to be deactivated at the beginning of the simulation and then be activated. Three types of criterion are implemented:

- The fracture pressure in a segment is activated as soon as that segment is in contact with another segment which has its pressure activated;
- The fracture pressure in a segment is activated as soon as a limit value of a pre-defined grandness – fracture aperture, fracture normal stress, fracture shear stress, fracture relative stress;
- A mixture of the previous two criteria, i.e., the pressure in a segment is activated when that segment is in contact with another segment with activated pressure and if a certain limit value is reached.

4.2.4. UEL algorithm

This is an essential step for any FEM simulation, as it is where the physical behaviour of the governing equations is represented through the construction of the elemental matrices that compose the global system of equations. These matrices are obtained by computing the integrals presented in Eqs. (3.38) to (3.40) and Eq. (3.43). Auxiliary procedures are also used to compute the non-linear terms, such as the elastoplastic stiffness matrix for material constitutive behaviour or the fracture longitudinal transmissibility.

A general flow of the coded UEL subroutine is presented in Figure 4.7. As stated previously, this procedure is run for every user element in every iteration of the simulation. Therefore, the first step of the routine is to select, both from the input files or the pre-processor, only the data relevant to that particular element. Then, two domains are integrated separately: the continuous region and the fracture.

First, the procedure loops over all the integration points in the continuous area. For each, it builds the shape functions and their derivatives (N and B, respectively), for both standard and enriched degrees of freedom, and also for both displacements and pore-pressures. Then, the code computes the stresses and the material stiffness matrix, which is elastic in the continuous region (see Figure 4.8). These matrices are then used to compute the contribution of the integration point for each of the area integrals, K, Q and H present in Eq. (3.43).

Second, the procedure loops over all the integration points in the fractures. Evidently, this part of the procedure is not run if there are not fractures in the element. For each fracture integration point, the routine builds the shape functions, their derivatives and the jump functions (N, B and $[\![N]\!]$, respectively), considering the contribution of the fracture pressure degrees of freedom. After, the code computes stresses and the material stiffness matrix, which may be elastoplastic in the fracture region (see Figure 4.8). The fracture longitudinal transmissibility is then computed, by applying the cubic law is used. It should be noted that the cubic law uses the fracture aperture to compute the transmissibility. Considering that, even when in contact, fractures present a larger transmissibility due to its roughness, a hydraulic aperture is used. At the beginning of a simulation, the hydraulic aperture is different from the mechanical aperture, taking a value defined by the user in the input files. All the previously mentioned matrices are then used to compute the contribution of the integration point for each of the fracture integrals, T, L, Q_{aP_F} and $H_{P_F P_F}$ present in Eq. (3.43).

Finally, all the computed integrals are inserted in the jacobian and the right-hand-side matrices and all the rows and columns related with deactivated degrees of freedom are zeroed.

It should be noted that every time the procedure is run, it stores and updates significant state variables related with every integration point, namely the stress tensor and plastic deformations.

Figure 4.8 presents the general steps taken to compute the stresses and the material stiffness matrix in every integration point (from continuous or fracture region), based on the material constitutive behaviour defined by the user.

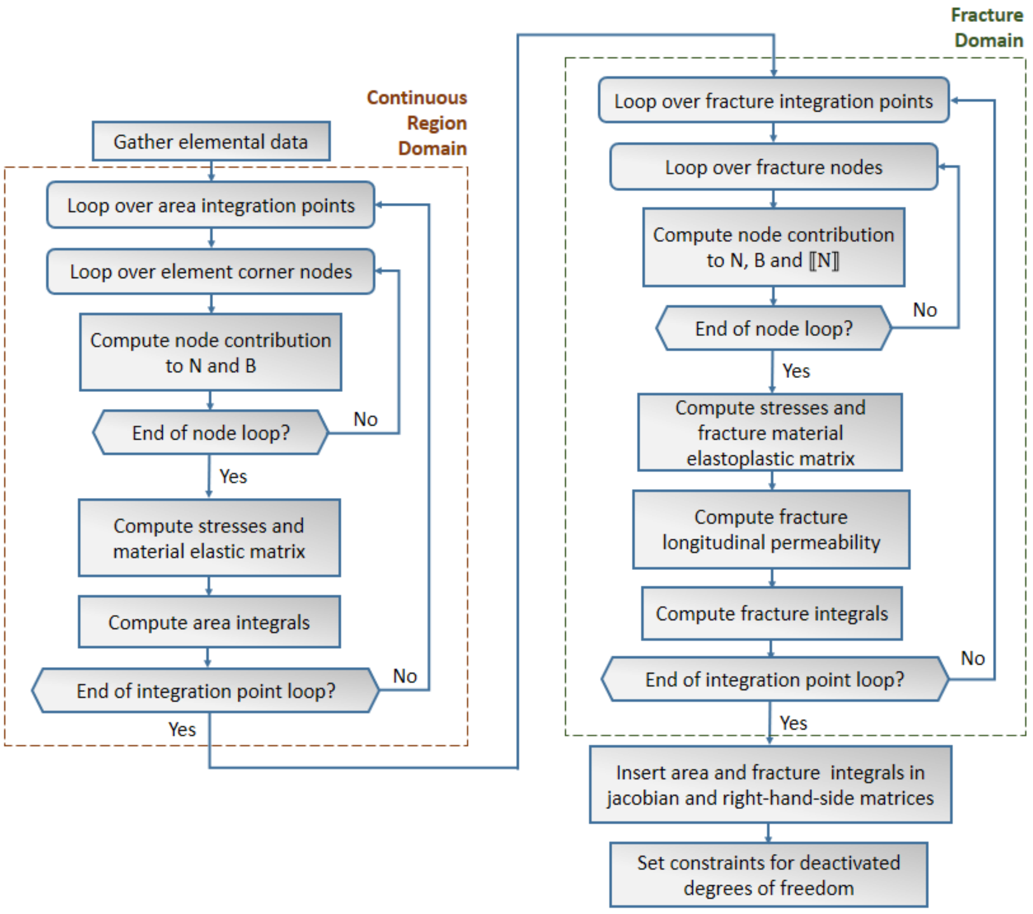

Continuous
Region
Domain
Gather elemental data
Loop over area integration points
Loop over element corner nodes
Compute node contribution to N and B
End of node loop?
No
Yes
Compute stresses and material elastic matrix
Compute area integrals
End of integration point loop?
No
Yes
Fracture
Domain
Loop over fracture integration points
Loop over fracture nodes
Compute node contribution to N, B and ⟦N⟧
End of node loop?
No
Yes
Compute stresses and fracture material elastoplastic matrix
Compute fracture longitudinal permeability
Compute fracture integrals
End of integration point loop?
No
Yes
Insert area and fracture integrals in jacobian and right-hand-side matrices
Set constraints for deactivated degrees of freedom

Figure 4.7 – General flow of the UEL subroutine

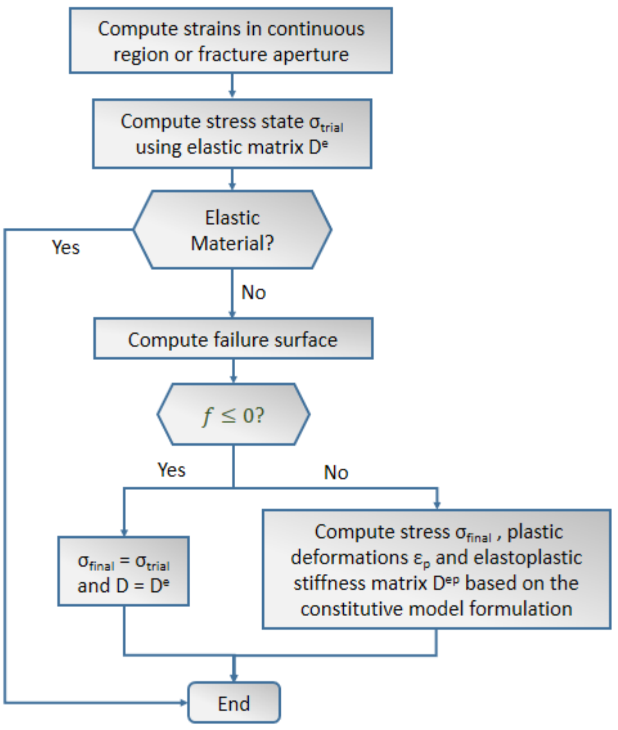

Compute strains in continuous region or fracture aperture
Compute stress state σ_{trial} using elastic matrix D^e
Elastic Material?
Yes
No
Compute failure surface
$f \leq 0$?
Yes
No
$\sigma_{final} = \sigma_{trial}$ and $D = D^e$
Compute stress σ_{final}, plastic deformations ε_p and elastoplastic stiffness matrix D^{ep} based on the constitutive model formulation
End

Figure 4.8 – General flow of the material constitutive subroutine

4.2.5. Fracture geometry post-processor

At the end of each increment, the algorithm checks the occurrence of fracture propagation. The verification is made using a user-defined propagation criterion. If a fracture propagates, then both the direction and length of propagation are computed. As seen in Figure 4.9, the post processor starts by defining in which regions of the model the propagation should be checked. Then, the propagation criterion is verified in each region. For every region where propagation occurs, the direction of the propagating segment is defined based on the direction criterion. Finally, the length of the propagating segment is computed. With the computed direction and length, the new coordinates for the propagating segments are obtained.

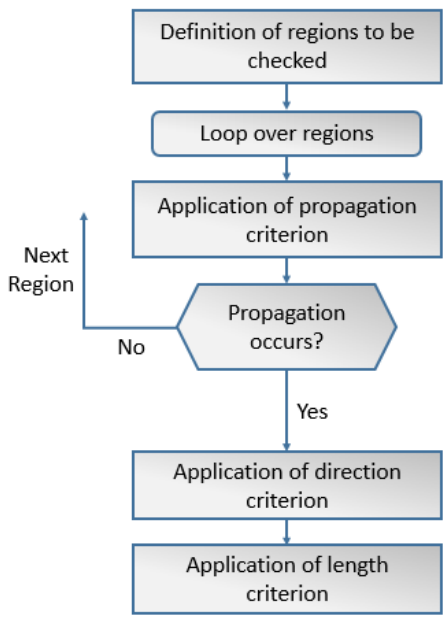

Definition of regions to be checked
Loop over regions
Application of propagation criterion
Next Region
Propagation occurs?
No
Yes
Application of direction criterion
Application of length criterion

Figure 4.9 – General flow of the fracture geometry post-processor

The adopted propagation criterion is based on the average minimum principal stress at the integration points of a region that is perpendicular to the fracture tip, as seen in Figure 4.10. The region may have different shapes. However, in this implementation, the region may be square or rectangular, depending on the user’s choice. If it is square, its side is equal to the dimension of the average. If it is rectangular, then it's dimensions are equal to the mesh average element.

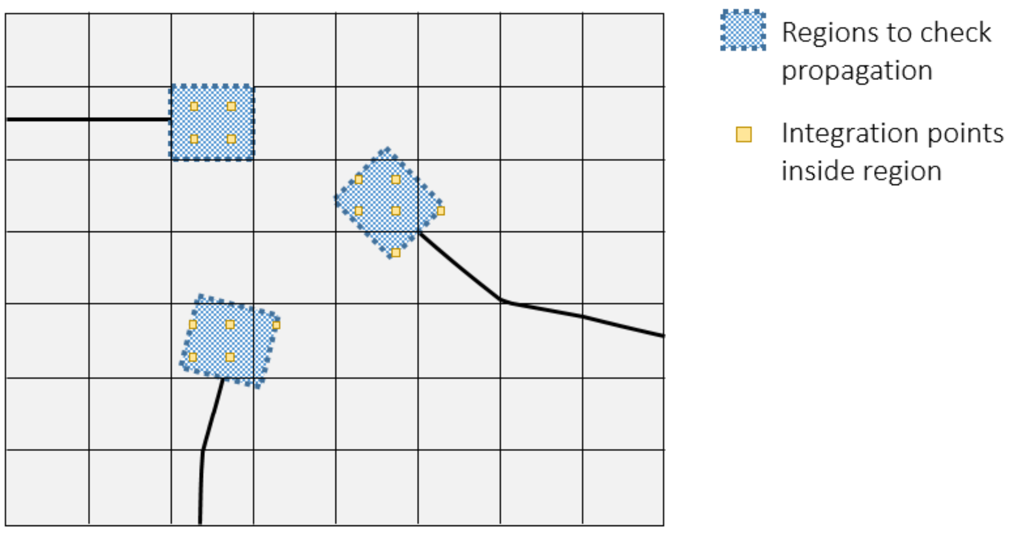

Regions to check propagation
Integration points inside region

Figure 4.10 – Examples of regions where propagation is checked

A specific treatment is made when a fracture tip is close to another fracture. Considering that stress fields are usually different at each side of a fracture, it would not be correct to pick the stresses in the opposite side of the fracture tip. This way, the region is redefined considering only the part that is of interest to the fracture tip, as seen in Figure 4.11.

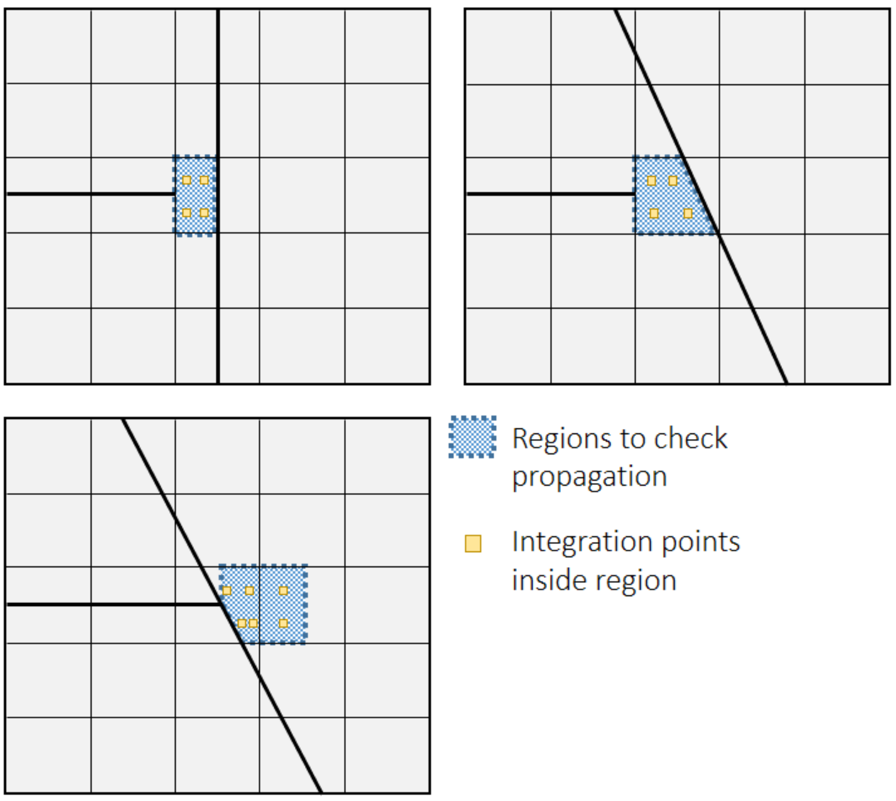

Regions to check propagation
Integration points inside region

Figure 4.11 – Examples of regions close to other fractures

Once the borders of the region are defined, the following criterion is applied

$$\overline{\sigma_3} > \sigma_t \tag{4.1}$$

Where $\overline{\sigma_3}$ is the average of the minimum principal stresses in the region's integration points and σ_t is the tensile strength of the rock. The direction criterion used is also based on the average of the principal stresses. Propagation occurs perpendicularly to the average of the minimum principal stress in the computed region.

Finally, the length of a propagating segment is such that propagation always extends up to the next element border. Consequently, only one element is allowed to propagate per increment. This is a similar approach as the one used in the Abaqus built-in XFEM elements. Despite the fact that this approach may lead to a decrease of the crack tip speed, as mentioned by Song, Areias and Belytschko (2006), this problem can be overcome by using sufficiently refined meshes.

4.2.6. Element topology

As mentioned previously, the Abaqus user element subroutine allows the implementation of any kind of finite element in the code. In this work, only one element was coded. However, its implementation is flexible enough to model different situations: standard porous finite element with hydro mechanical coupling, enriched porous finite element with coupling between the hydro mechanical porous region and the fracture fluid pressure, enriched porous finite element with multiple fractures and intersections with coupling between the hydro mechanical porous region and the fracture fluid pressure.

This flexibility is achieved by allocating degrees of freedom to all the mentioned situations, activating and deactivating them during the simulation depending on the modelling needs. For example, if there is need to simulate an impermeable element with one fracture, all the pore pressure degrees of freedom and intersection degrees of freedom should be deactivated. Once Abaqus does not allow run time deactivation, this is achieved by zeroing every coefficient related to those degrees of freedom.

The basis of the user element is a Q4 plane strain linear segment, with the corner nodes storing the standard displacement and pore pressure degrees of freedom. Additionally, the same corner nodes also store the displacement and pore pressure enriched degrees of freedom.

These degrees of freedom would be enough to model a multi fractured porous medium if there was no need to consider the fracture fluid pressure. However, considering the importance of working with the fracture fluid pressure as a variable (as explained in Chapter 3.1), additional degrees of freedom are added to consider it.

By principle, it would be correct to place the fracture pressure degrees of freedom coincident with the fracture segments extremities. However, as seen in Figure 4.12, that is not possible with Abaqus. As known, XFEM is a technique that allows simulation of fracture propagation without prior knowledge of the fracture path, i.e. each new fracture segment is positioned as the simulation runs. Consequently, at the beginning of the simulation it is not possible to know which elements will me intersected by fractures or to state where the fracture pressure degrees of freedom will be. As Abaqus does not allow the placement of new nodes in elements while the simulation runs, it is impossible to guarantee that the fracture pressure degrees of freedom will be positioned coincident with the fracture segments.

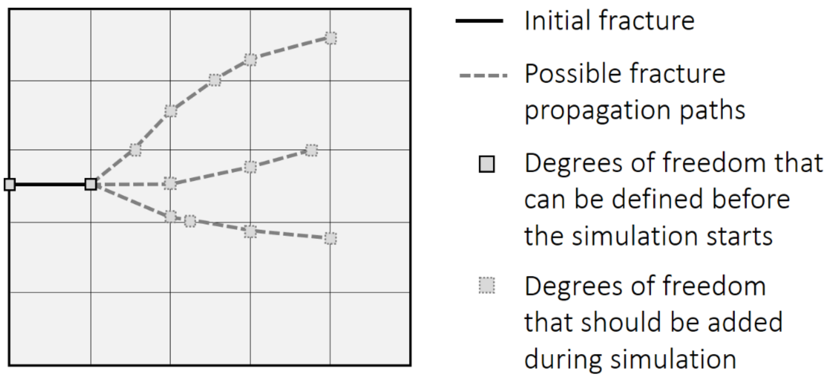

Initial fracture
Possible fracture propagation paths
Degrees of freedom that can be defined before the simulation starts
Degrees of freedom that should be added during simulation

Figure 4.12 – Possible positions of fracture pressure degrees of freedom in possible fracture propagation segments

To overcome this situation, the following workaround is implemented (see Figure 4.13):

- 9 nodes are considered in every element since the beginning of the simulation – the already existing 4 corner nodes + 4 element border middle nodes + 1 node at the element centroid

- Every time a fracture propagates into a new element, an algorithm in the pre-processor defines which nodes should store the new fracture variables
- If other fractures propagate into the same element (creating an intersection), the algorithm stores the new fracture variables in other nodes that are still available

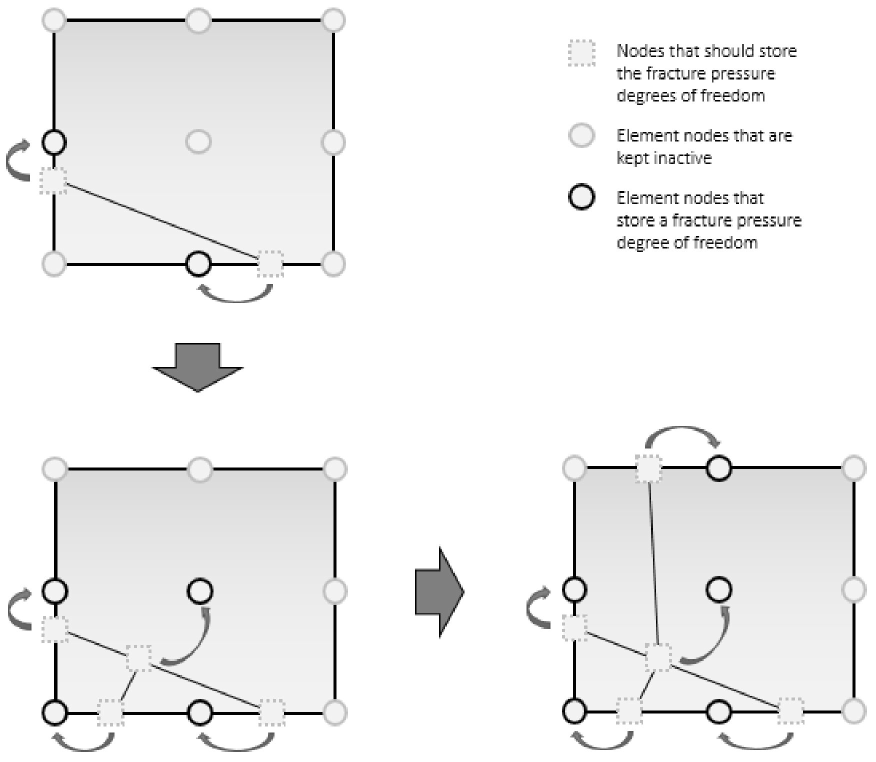

Nodes that should store the fracture pressure degrees of freedom
Element nodes that are kept inactive
Element nodes that store a fracture pressure degree of freedom

Figure 4.13 – Storage of fracture pressure degrees of freedom

4.2.7. Numerical integration

4.2.7.1. Introduction

As stated previously, the Finite Element Method demands the problem sub-domains (i.e. elements) to integrate smooth and continuous functions. In such cases as the discretization with interface elements, the integration on both sides of the fracture is made in separate continuous elements and the fracture domain integration is made directly in the interface elements that represent that fracture. In XFEM, the fracture domain is within the continuous region, leading to the need of performing two types of integrations in the same element. In 2D the fracture domain (*Γ*) requires line integration and the element domain (*Ω*) requires area integration.

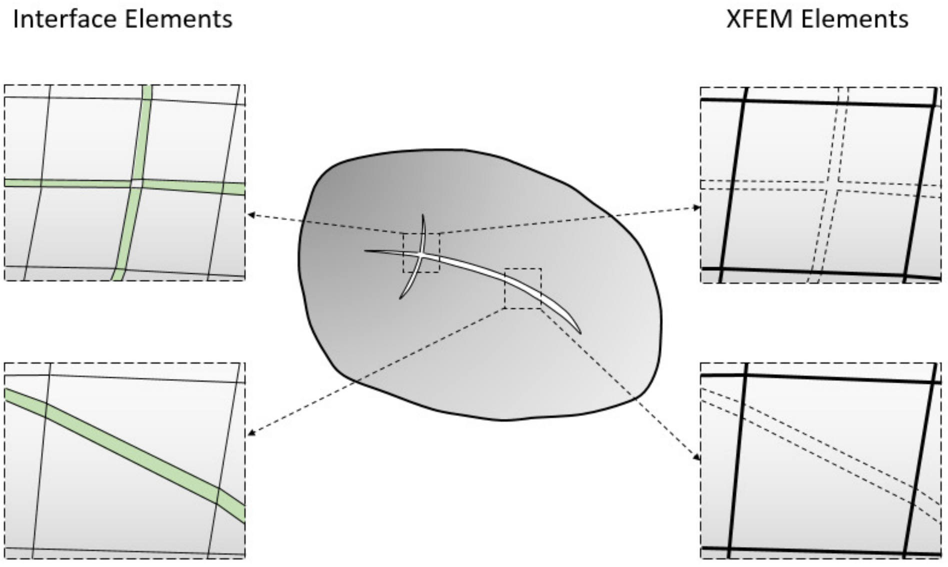


Figure 4.14 – Difference of element definition between Interface elements and XFEM elements. On the left, black continuous lines represent element borders, grey hatches represent continuous elements and green hatches represent interface elements. On the right, black continuous lines represent element borders, grey hatches represent continuous elements and dashed lines represent the fracture inside the element domain

4.2.7.2. Continuous Region

In this work, the integration over continuous regions, i.e. *Ω* in 2D elements, is performed by using the Gauss Method. In elements where all the enriched degrees of freedom are deactivated, i.e. non-fractured elements, the integration may be performed following the conventional techniques, such as presented in Potts and Zdravković (1999).

For enriched elements, the explicit consideration of fractures inside the element domain requires the implementation of non-standard techniques for element integration. The integrand in the element domain is no longer continuous with continuous derivative (C^1 class) due to the presence of the enrichment functions (see Figure 3.7 or Figure 3.9). However, functions that develop on both sides of the fracture are C^1 class and therefore the standard Gauss integration technique applies. Thus, sub-regions within the element where continuity applies may be integrated separately and the results be summed. Eq. (4.2) shows a Gauss integration procedure applied to the element sub-regions $nsubreg$. $iIPsub$ is the number of integration points in the sub-region, W the Gaussian weight and f the function to be integrated.

$$\iint_{-1}^{1} f(x,y)dxdy = \sum_{j=1}^{nSubReg} \sum_{i=1}^{kIPsub} W_{i,j} \, . \, f\left(x_{i,j}, y_{i,j}\right) \tag{4.2}$$

The subdivision process starts by cutting the element in sub-regions delimited by the fracture. Then the following rules apply:

- If the sub-region has four sides, Gauss integration for quadrilateral elements apply, as in the sub-regions defined in Figure 4.15a) and b);
- If the sub-region has three sides, Gauss integration for triangular elements apply, as seen in the lower left sub-region in Figure 4.15c);
- If the sub-region has five sides or more, further subdivision defines triangles with vertices coinciding with the original sub-region vertices and its centroid. In these triangles, Gauss integration for triangular elements applies, as seen in the upper right sub-regions in Figure 4.15c);

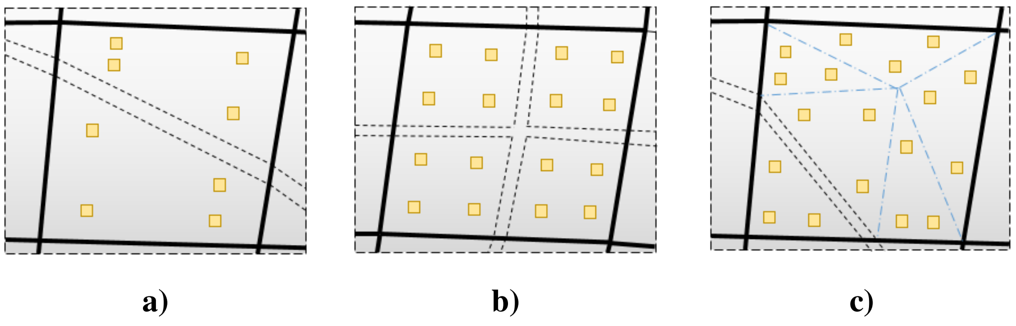


Figure 4.15 – Examples of integration points position in sub-regions

For both non-fractured elements and sub-regions, the Gauss integration order is defined by the user, varying on a range from 1 to 7.

4.2.7.3. Discontinuous Region

The integrals over fracture regions, i.e. Γ_d, are computed along fictitious line elements that coincide with the fracture position. Although these elements do not exist in the mesh topology, their implementation is critical to represent the fracture hydraulic and constitutive behaviour. The integration performed over these elements also uses the Gauss integration rule. It is worth pointing out that a fracture intersected by another fracture may show a different behaviour on each side of the intersection. Therefore, the placement of Gauss points must consider not only the fracture position but also its intersections. Figure 4.16 shows some examples of the positioning of integration points along the discontinuity. In this research, only a second order integration scheme is used.

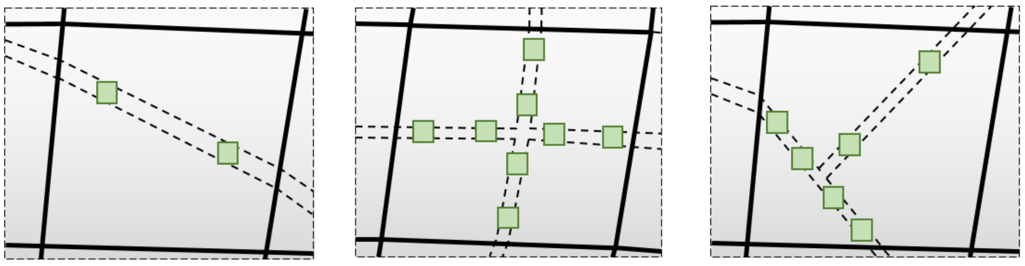


Figure 4.16 – Examples of integration points position in fractures

4.2.8. Limitations of the implementation

Abaqus is a powerful and very versatile tool, both for the built in and external user implementations. Nevertheless, the use of a commercial software where only a part of the code algorithm is reachable by the user brings naturally some limitations.

An already mentioned limitation of Abaqus is the impossibility to use the graphical interface both for input and output when using user elements. This issue which was overcome by the additional codes and software referred in Chapter 4.2.1.

The main limitation of the implementation is the way Abaqus deals with degrees of freedom of user elements. The degrees of freedom that are access by a user element must be defined in the input before the simulation starts. As stated before, the principle of XFEM develops around the concept of activation of new degrees of freedom in the run. To guarantee that all possible degrees of freedom will be available during the simulation, they need to be previously declared in the input. This means that many elements that are never fractured during a simulation have stored within them the deactivated degrees of freedom. This has obviously a negative effect in the calculation time and memory consumption.

Other related limitation is the number of degrees of freedom that Abaqus allows for user elements, which is limited to 30 per node. This is overcome by attributing the same enrichment degree of freedom to different fractures, which is not problematic as long as different fractures do not share the same enriched nodes. However, when modelling intersection between fractures there are nodes that have enriched degrees of freedom from different fractures, so these fractures must have different enrichment degrees of freedom. This to say that with a limited number of degrees of freedom per node the number of fractures in the model is not limited, but the number of intersections inside one element is.

The degrees of freedom of user elements may be attributed to any physical grandness, as the differential equations are defined in the code. However, every time Abaqus runs a step, a keyword related with the type of calculation must be defined. For example, static problems, where only displacement (position 1to 3) and rotation degrees of freedom (position 4 to 6) are active, or consolidation, where both displacement (position 1 to 3) and pore pressure (position 8) degrees of freedom are active. Although the implemented code has only intent of computing displacements and fluid pressures, the need of extra degrees of freedom for the enrichments demands a type of calculation that allows the maximum number of degrees of freedom possible. This is achieved by choosing a coupled displacement-temperature calculation. This way, the degrees of freedom 1 to 6 (displacement and rotation) and 7 and 11-30 (temperature) are available.

Finally, it must be highlighted that the computational geometry functions used in the fracture geometry pre and post-processor allow only the use of regular meshes.