

Application of the 3-D Hydro-Mechanical Model GEOFRACT in Enhanced Geothermal Systems

by

Alessandra Vecchiarelli

Laurea (Bachelor) of Engineering in Environmental Engineering
Sapienza, University of Rome, Italy (2005)

Laurea specialistica (M.Eng) of Engineering in Environmental Engineering
Sapienza, University of Rome, Italy (2008)

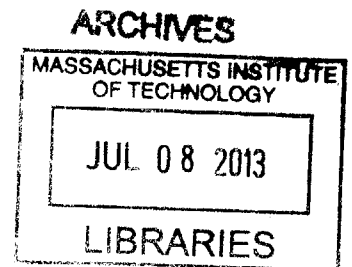
Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Civil and Environmental Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013



© 2013 Massachusetts Institute of Technology. All rights reserved.

Signature of Author.....

Department of Civil and Environmental Engineering
May 10, 2013

Certified by.....

Herbert H. Einstein
Professor of Civil and Environmental Engineering
Thesis Supervisor

A handwritten signature in black ink, appearing to be "H. Einstein", written over the printed name.

Accepted by.....

Heidi M. Nepf
Chair, Departmental Committee for Graduate Students

A handwritten signature in black ink, appearing to be "Heidi M. Nepf", written over the printed name.

Application of the 3-D Hydro-Mechanical Model GEOFRAC in Enhanced Geothermal Systems

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Civil and Environmental Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ABSTRACT

GEOFRAC is a three-dimensional, geology-based, geometric-mechanical, hierarchical, stochastic model of natural rock fracture systems. The main characteristic of GEOFRAC is that it is based on statistical input representing fracture patterns in the field in form of the fracture intensity P_{32} (fracture area per volume) and the best estimate fracture size $E[A]$. Recent developments in GEOFRAC allow the user to calculate the flow in a fractured medium. For this purpose the fractures are modeled as parallel plates and the flow rate can be calculated using the Poiseuille equation. This thesis explores the possibility of the application of GEOFRAC to model a geothermal reservoir. After modeling the fracture flow system of the reservoir, it is possible to obtain the production flow rate. A parametric study was conducted in order to check the sensitivity of the output of the model. An attempt to explain how aperture, width and rotation (orientation distribution) of the fractures influence the resulting flow rate in the production well is presented. GEOFRAC is a structured MATLAB code composed of more than 100 functions. A GUI was created in order to make GEOFRAC more accessible to the users. Future improvements are the keys for a powerful tool that will let GEOFRAC to be used to optimize the location of the injection and production wells in a geothermal system.

ACKNOWLEDGEMENTS

First and foremost I would like to thank my research advisor, Professor Einstein, for giving me the tremendous opportunity to be part of his research group. Your trust in my capabilities has always inspired me to continue. I really learned a lot from you the first time I came to MIT as a visiting student, and it has been a privilege to come back again as an MIT student under your supervision.

I would like to acknowledge the geotechnical professors: Dr. Jen, Dr. Germaine and Prof. Whittle. I have learned so much in the past few years from your classes. It has been my great privilege to learn the best from the best.

Thanks to all the people on the third floor, especially my officemates, Bruno, Steve and the “new entry” Zenzi who have really made it a fun environment. Even in this very last tough period you have always asked me how I was and helped me to go on with your comforting words.

To my closest friends and colleagues Gery and Shehab, thanks for sharing the pains and joys together with me. Thanks for always being there for me and for being such wonderful friends. I am going to miss you so much.

To my colleagues and gardening partner Amy, thanks for sharing your experience and teaching me how to take care of our plot. We cared so much that we even got little bunnies which we called “our babies”.

Thanks to the Italian community and the MITALY board Chris, Giancarlo and Federico for helping me to feel at home by speaking Italian and organizing Italian cultural events. I am so happy that we will continue to share experiences over the next few years.

I would like to thank my family, especially my parents, Giorgio and Margherita Vecchiarelli, for their never ending support and encouragement throughout my life. I

would not be where I am today without you standing behind me, cheering me on. Thank you, Dad, for all of your support and for giving me the opportunity to accomplish my goals. Thank you, Mom, for taking care of me 24/7 in the past years and being supportive of me. Thanks to my brother for being so close to me, even when we are on different continents. I love you all very much.

To my husband, Marcello Scarnecchia, if you hadn't decided to quit your job in Italy to come with me and let me follow my dreams, I would probably not be here today. The past two years were very difficult to us, especially the last 5 months being so far from each other, but we managed it very well and I love you more every day. I hope that our life will be always an adventure and that we will continue to have fun together.

I dedicate this thesis to my dad and my mom who love me and believe in me. They have been working very hard every single day to not let me miss anything.

APPLICATION OF THE 3D HYDRO-MECHANICAL MODEL GEOFRAC IN ENHANCED GEOTHERMAL SYSTEM

TABLE OF CONTENTS

1 INTRODUCTION

2 A REVIEW OF PREVIOUS MODELING OF GEOTHERMAL RESERVOIR

2.1 Mathematical background

2.1.1 Fracture system model

2.1.2 Flow system model

2.2 Software in commerce

2.2.1 TOGH2

2.2.2 FALCON

2.2.3 LEAPFROG

2.2.4 FRACMAN

2.2.5 Conclusions

3 GEOFRAC: 3-D HYDRO MECHANICAL MODEL

3.1 Introduction

3.2 Basic concept

3.2.1 Fracture system model

3.2.2 Flow system model

4 PARAMETRIC STUDY

4.1 Introduction

4.2 Parametric analysis results

4.2.1 Output analysis function of the aperture model

4.2.2 Output analysis function of the Fisher's parameter

4.2.3 Effect of fractures translation

4.3 Conclusions

5 BOREHOLE INTERSECTION

5.1 Introduction

5.2 Intersection algorithm

5.2 Introduction to optimization process

6 GEOFRAC GRAPHICAL USER INTERFACE

6.1 Introduction

6.2 Input parameter

6.2.1 Geometric inputs

6.2.2 Stochastic inputs

6.2.3 Simulation

6.2.4 Flow input

6.3 Results

APPENDIX 1 – GEOFRAC: GUI file list

APPENDIX 2 – GEOFRAC: GUI source code

7 CONCLUSIONS AND FUTURE RESEARCH

7.1 Summary and conclusions

7.2 Future research

REFERENCES

LIST OF FIGURES

Figure 2.1 - Representation of a fractured rock mass shown in (a), by FDM or FEM shown in (b), BEM shown in (c), and DEM shown in (d). (Jing, 2003)

Figure 2.2 - (a) Regular quadrilateral grid for the FDM and (b) irregular quadrilateral grid for the FDM (Jing, 2003)

Figure 2.3 - Staggered grid used for the 2-D finite-difference scheme (Coates et al. 1995)

Figure 2.4 - Illustrative meshes for fracture analysis with BEM: (a) sub-domain, direct BEM; (b) single domain, dual BEM (Jing, 2003)

Figure 2.4 - Physical model of implemented DEM (Deng et al., 2011)

Figure 2.5 - Three-dimensional orthogonal model (Dershowitz and Einstein, 1988)

Figure 2.6 - Comparison of (a) the general Baecher model with (b) the Enhanced Baecher (Staub, 2002)

Figure 2.7 - “Generation of Veneziano joint system model” (Einstein, 1993)

Figure 2.8 - (a) 3-D fractal Box algorithm, and (b) 3-D geometric model (Dershowitz et al, 1998)

Figure 2.9 - 3D geometric Levy-Lee fractal model (Dershowitz et al, 1998)

Figure 2.110 - Hybrid DFN/EPM Model Framework (Dershowitz, 2006)

Figure 2.111 - Stochastically Generated Water Conducting Fracture Population (WCF) Throughout Model Region (Dershowitz, 2006)

Figure 2.13 - Three dimensional fracture intensity measures (Dershowitz and Herda, 1992)

Figure 2.14 - Vertical borehole in an objective volume (Zhang, Einstein, 2000)

Figure 2.15 - Truncated lognormal distribution (Ivanova et al. 2012)

Figure 2.16 - Typical computational mesh (Travis, 1984)

Figure 2.17 - Steady state distribution of: (a) fracture saturation; (b) matrix saturation; (c) fracture water flux; and (d) matrix water flux $q_a=42.5$ mm/yr (Illman et al., 2005)

Figure 2.18 - Three-dimensional mesh for the simplest fracture flow problems under consideration (Podgorney et al., 2012)

Figure 2.19 - Temperature (in fracture domain) and pressure along the center of the reservoir matrix domain for tow simulations. (Podgorney et al., 2012)

Figure 3.1- Fracture represented as a polygon with a pole and a radius (Ivanova, 1995)

Figure 3.2 - Generation of a fracture set with the GEOFRAC model. Primary process (a), secondary process (b), tertiary process (c).

Figure 3.3 - Expected fracture connectivity, C , for given $E[A]$ and $P32$ (Ivanova et al., 2012)

Figure 3.4- Schematic representation of linear flow between parallel plates

Figure 3.5 – Mean fracture width between fractures intersection

Figure 3.6 - Representation of the fracture intersections (Sousa, 2013)

Figure 3.7 - Score components between two fractures intersection

Figure 3.8 - Score components: fracture intersects more than one fracture

Figure 3.9 - Intersection of two paths; representation of branches and nodes.

Figure 3.10 - Series of fractures modeled as parallel plates

Figure 3.11 - Highest score path between two fractures that intersect the two boundaries

Figure 3.12 - Highest score path between different pairs of fractures

Figure 4.12 - Flow rate for $h=0.005$ m for no rotation and random rotation of the fractures

Figure 4.13 -Flow rate for $h=0.01$ m for no rotation and random rotation of the fractures

Figure 4.14 - Fracture set poles. Orientation distribution: Univariate Fisher $\kappa =1$

Figure 4.15 - Fracture set poles. Orientation distribution: Univariate Fisher $\kappa =40$

Figure 4.16 – Flow rate (Q_{out}) values for $\kappa =1$

Figure 4.17 - Flow rate (Q_{out}) for $\kappa =40$

Figure 4.7 – Fracture paths system for simulation with $\kappa =1$, $h=0.005$ rotation

Figure 4.8 - Fracture paths system for simulation with $\kappa =1$, $h=0.01$ rotation

Figure 4.9 - Fracture paths system for simulation with $\kappa =1$, $h=0.005$ no rotation

Figure 4.10 - Fracture paths system for simulation with $\kappa =1$, $h=0.01$ no rotation

Figure 4.11- Fracture paths system for simulation with $\kappa =40$, $h=0.005$ rotation

Figure 4.12 - Fracture paths system for simulation with $\kappa =40$, $h=0.01$ rotation

Figure 4.13- Fracture paths system for simulation with $\kappa =40$, $h=0.005$ no rotation

Figure 4.14 - Fracture paths system for simulation with $\kappa =40$, $h=0.01$ no rotation

Figure 4.15 – Schematic representation of the translation between fractures

Figure 4.16 - Translation of the fractures vs Q_{out}

Figure 5.1 - Fracture systems intercepted by the two wells

Figure 5.2 Schematic representation of geometric inputs for the Intersect Borehole function

Figure 5.3- Matlab representation of intersection between a cylinder and spheres

Figure 5.4 - Schematic representation of intersection between the borehole and one vertex of the fracture (polygon)

Figure 5.5 - Schematic representation of a fracture (polygon) inscribed into the borehole

Figure 5.6- Schematic representation of an intersection between one side of a fracture and the cylinder

Figure 5.7- Schematic representation of intersection between a fracture (plane) and a borehole (cylinder)

Figure 6.18 – GEOFRAC GUI (user's interface)

Figure 6.19 - Geometric dimensions of the reservoir to be modeled

Figure 6.20 – Coordinates of the controlled volume according to GEOFRAC

Figure 6.21 - Section of the stochastic inputs in the GUI.

Figure 6.22 - Simulation parameter: number of simulations and type of simulation

Figure 6.23 - Flow parameters in GEOFRAC

Figure 6.24 - Flow system in the controlled volume

Figure 6.25- Example of representation of the Qout results

LIST OF TABLES

Table 2.1 - Main review of the DFN model, as regard to their applicability, advantages and limitations (Staub, 2002)

Table 4.2- Values of Q_{out} (m³/s) for $h = 0.005$ m

Table 4.3 - Values of Q_{out} (m³/s) for $h = 0.01$ m

Table 4.3 - Max and min values of the translation of the fractures

Table 6.4- Types of orientation distributions in GEOFRAC and parameters that needs to be set

Table 6.5 - Types of methods for the fracture aperture and parameters that needs to be set

Table 6.6- Excel file with the summary of some important results

CHAPTER 1

INTRODUCTION

In deep geothermal energy projects naturally and artificially induced fractures in rock are used to circulate a fluid (usually water) to extract heat; this heat is then either used directly or converted to electric energy. MIT has developed a stochastic fracture pattern model GEOFRAC (Ivanova, 1995; Ivanova et al., 2012). This is based on statistical input on fracture patterns from the field. The statistical input is in form of the fracture intensity P_{32} (fracture area per volume) and the best estimate fracture size. P_{32} can be obtained from borehole spacing information on observations and outcrops and the approach by Dershowitz and Herda (1992). Best estimate fracture size can be obtained from fracture trace lengths on outcrops with suitable bias corrections as developed by Zhang et al. (2002). Distribution and estimates of fracture size can also be obtained subjectively. GEOFRAC has been applied and tested by estimating the fracture intensity and estimated fracture size from tunnel records and from borehole logs. In the research case here presented, GEOFRAC predictions were satisfactorily applied for geothermal basin characterization. Since its original development, GEOFRAC has been made more effective by basing it on Matlab and it has been expanded by including an intersection algorithm and, most recently, a flow model. GEOFRAC belongs to the category of

Discrete-Fracture Network models. In this type of model the porous medium is not represented and all flow is restricted to the fractures. Fractures are represented by polygons in three dimensions. Both the fracture - and flow model have been tested and a parametric study was conducted in order to check the sensitivity of the output results to the inputs.

In Chapter 2 I briefly describe the existing theoretical models of geothermal reservoir simulation with emphasis on hydro-mechanical models. I also introduce the available commercial software tools and their application on large scale. Chapter 3 describes the basic concepts of GEOFRAC, a three-dimensional discrete fracture pattern model. In Chapter 4 the parametric analysis used to capture the sensitivity of the input parameter of GEOFRAC is presented and discussed. Chapter 5 introduces a new algorithm in GEOFRAC that can be used to model intersections between fractures and injection and production wells. In Chapter 6 the use of the GUI to run GEOFRAC is introduced and explained. The conclusions in Chapter 7 are intended to explain the overall results and to briefly discuss future research.

CHAPTER 2

A REVIEW OF PREVIOUS

MODELING OF GEOTHERMAL

RESERVOIR

The development of simulation model for enhanced geothermal reservoirs requires predicting the capacity of hydraulically induced reservoirs. Geothermal reservoir modeling in turn requires an adequate mathematical representation of the physical and chemical processes during the long-term heat extraction period. During the last 20 years the use of computer modeling of geothermal areas has become standard practice.

The intent of this chapter is to describe available models on the basis of the geological, geometric, mechanical (both solid and fluid) and thermal conditions.

2.1 FRACTURE SYSTEM MODELS

Modeling is an essential phase in studying the fundamental processes occurring in rocks and for rock engineering design. Modeling rock masses represents, however, a very

complex challenge; the most important reason is that the rock is a natural geological material. A rock mass is a discontinuous, anisotropic, inhomogeneous and non-elastic medium; it is subjected to stresses often by tectonic movements, uplift, subsidence, glaciation and tidal cycles. A rock mass is also a fractured and porous medium containing fluids in either liquid or gas phases, for example, water, oil, natural gas and air, under different in situ conditions of temperature and fluid pressures. The combination of all these factors makes rock masses a difficult material for mathematical representation via numerical modeling. The difficulty increases when coupled thermal, hydraulic and mechanical processes need to be considered simultaneously.

Uncertainties are one of the main characteristics of rock mechanics. It is important to understand the uncertainties and assess them—so that it is possible to run models managing risks without being conservative.

The aim of this section is to present numerical methods that are currently used to model rock fractures. It will introduce each model with a brief description followed by application cases. After this section, some of the models used to simulate particular characteristics of a rock mass such as the aperture of fractures and the intensity of the fractures are presented.

2.1.1 Numerical methods

The most commonly applied numerical methods in rock mechanics are:

Continuum methods

- Finite Difference Method (FDM)
- Finite Element Method (FEM)
- Boundary Element Method (BEM)

Discontinuum methods

- Discrete Element Method (DEM)
- Discrete Fracture Network (DFN) methods

Hybrid continuum/discontinuum models

- Hybrid FEM/BEM
- Hybrid DEM/DEM
- Hybrid FEM/DEM
- Other hybrid models

I will briefly describe these methods and their application in rock mechanisms.

The first category of numerical models consists of the continuum methods. Continuity is a macroscopic concept. The continuum assumption implies that, the material cannot be split or broken into pieces. All material points originally in the neighborhood of a certain point in the problem domain remain in the same neighborhood throughout the

deformation or transport process. Of course, at the microscopic scale, all materials are discrete systems. However, representing the microscopic components individually is mathematically complicated and often unnecessary in practice (Jing, 2003).

Figure 2.1 shows how to model a fractured rock mass (Figure 2.1 a) using a continuum method such as FDM or FEM (Figure 2.1 b), the BEM (Figure 2.1 c) (Jing, 2003). The description of these methods will be given in the next sections.

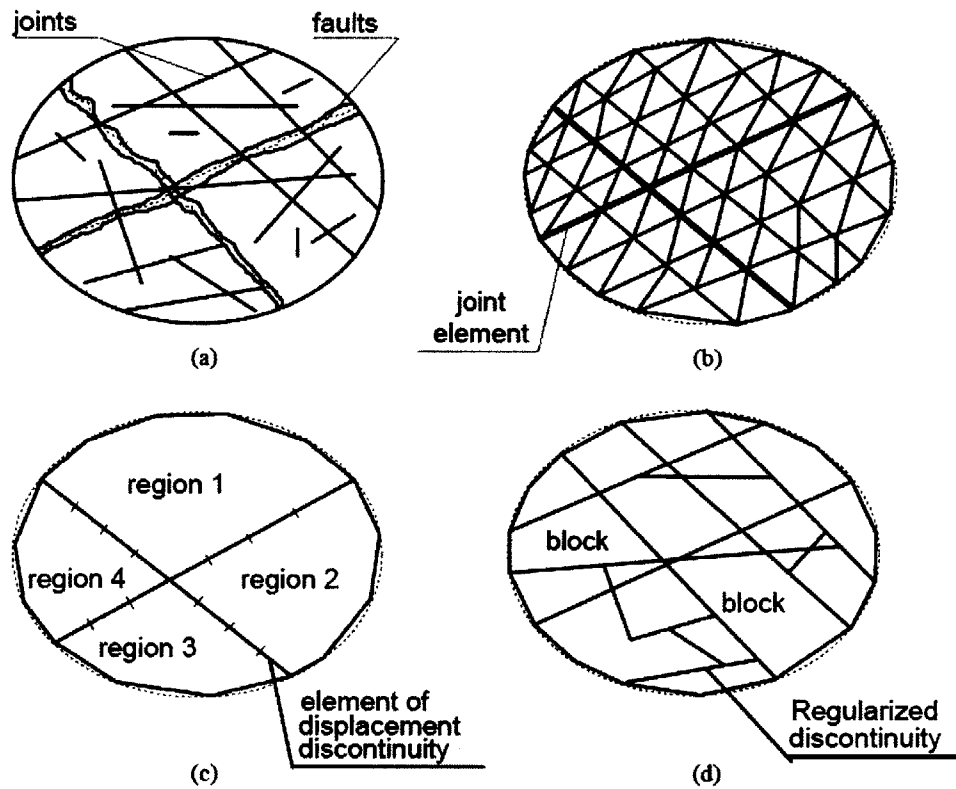


Figure 2.1 - Representation of a fractured rock mass shown in (a), by FDM or FEM shown in (b), BEM shown in (c), and DEM shown in (d). (Jing, 2003)

Discrete element methods are numerical procedures for simulating the complete behavior of systems of discrete, interacting bodies. Discrete Element Methods (DEM) and Discrete Fracture Networks (DFN) will be described after the continuum method.

The combination of the continuum and discrete models produces a very interesting group of so-called hybrid models. Hybrid models are frequently used in rock engineering, for flow and stress/deformation problems of fractured rocks. The main types of hybrid models are the hybrid BEM/FEM, DEM/BEM models.

Finite Difference Methods

The FDM approximates the governing PDEs by replacing partial derivatives with differences in regular (Figure 2.2 a) or irregular grids (Figure 2.2 b) imposed over the problem domain. The original PDEs are transformed into a system of algebraic equations in terms of unknowns at grid points. After imposing the necessary initial and boundary conditions the solution of the system equations is obtained.

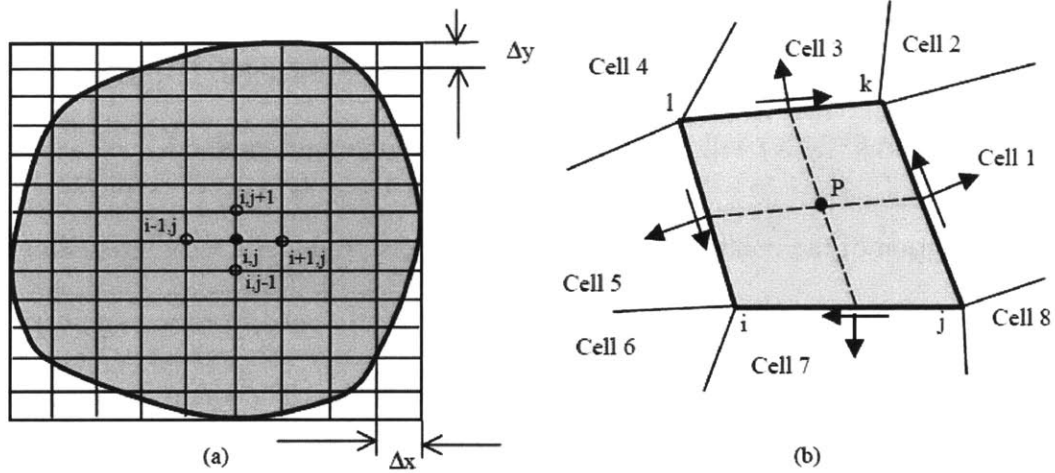


Figure 2.2 - (a) Regular quadrilateral grid for the FDM and (b) irregular quadrilateral grid for the FDM (Jing, 2003)

Coates and Schoenberg (1995) use the Finite Difference Method in order to model faults and fractures. What they try to do is to use the seismic propagation to detect slip surfaces. Figure 2.3 shows how they use the staggered grid and the locations at which the different components, for example, of the stress and velocity, are defined. The use of the FDM was successful and all equations used are well presented in their paper. The issue that they also reported in the conclusion is that it is not so easy to assess whether a slip surface is a realistic model of a fault.

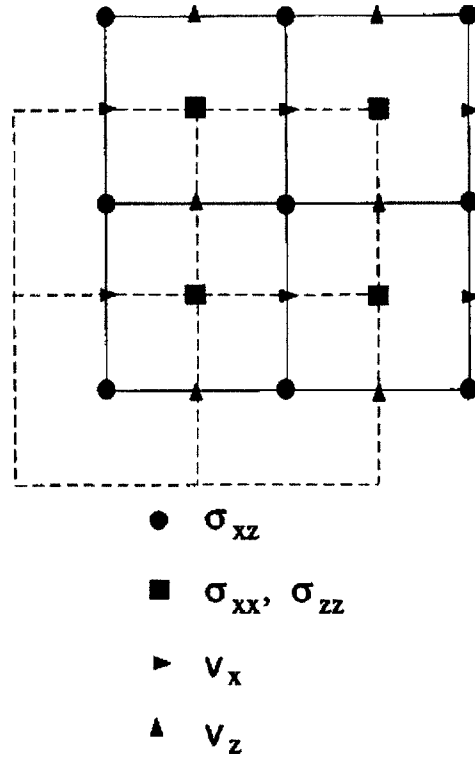


Figure 2.3 - Staggered grid used for the 2-D finite-difference scheme (Coates et al. 1995)

One of the best known commercially available FD codes is FLAC. FLAC is an advanced two-dimensional continuum model for geotechnical analysis of rock, soil, and structural support. Many researchers have used FLAC for studies such as stability of a slope and of a rock mass. (Shen et al., 2012; Apuani et al., 2005).

Finite Element Method

Finite element modeling is a well-established numerical technique that allows one to address the influence of the complexities that arise from non-linear behavior in geological

deformations. Finite element models are used in addressing a broad variety of geological problems ranging from folding and fracturing of rocks (e.g., Zhang Y. et al., 2000) to tectonics (Kwon, 2004; Kwon et al., 2007).

The FEM requires the division of the problem domain into sub-domains; i.e. elements of smaller sizes and standard shapes such as triangles, quadrilaterals, tetrahedrals, etc. with a fixed number of nodes at the vertices and/or on the sides (Figure 2.1 b). Polynomials are used to approximate the behavior of the PDEs at the element level and generate the local algebraic equations representing the behavior of the elements.

Zhang Y. et al. (2000) compare the results of a numerical modeling of single-layer folding using the Finite Element FLAC and Finite- Element MARC (Zhang Y. et al., 1996; Mancktelow, 1999). Numerical models of single-layer folds obtained using FLAC are consistent with those using the MARC for the same material properties and boundary conditions. The differences in the results reported by Zhang et al. (1996) and Mancktelow (1999) were not due to the different computer codes used in the two studies. The explanation lies in the different strain rates employed.

Boundary Element Method

The BEM solves linear partial differential equations that have been formulated as integral equations (i.e., in boundary integral form) (Figure 2.1 c). The BEM requires discretization at the boundary of the solution domains, reducing the problem dimensions by one and greatly simplifying the input requirements. The information required in the solution domain is separately calculated from the information on the boundary, which is

obtained by solution of a boundary integral equation, instead of direct solution of the PDEs, as in the FDM and FEM (Jing, 2003). The BEM, as FDM and FEM, can be used to solve both dynamic and static problems.

In order to use this method for fracture analysis, the fractures must be assumed to have two opposite surfaces. Denote Γ_c as the path of the fractures in the domain Ω with its two opposite surfaces represented by Γ_c^+ and Γ_c^- (Figure 2.4). Two techniques were proposed to model the domain. The first is to divide the problem domain into multiple sub-domains with fractures along their interfaces, (Figure 2.4 a). The stiffness matrix contributed by opposite surfaces of the same fracture will belong to different sub-domain stiffness matrices, in this way the singularity of the global matrix is avoided (Jing, 2003). The second is to apply displacement boundary equations at one surface of a fracture element and traction boundary equations at its opposite surface, although the two opposing surfaces occupy practically the same space in the model (Figure 2.4 b).

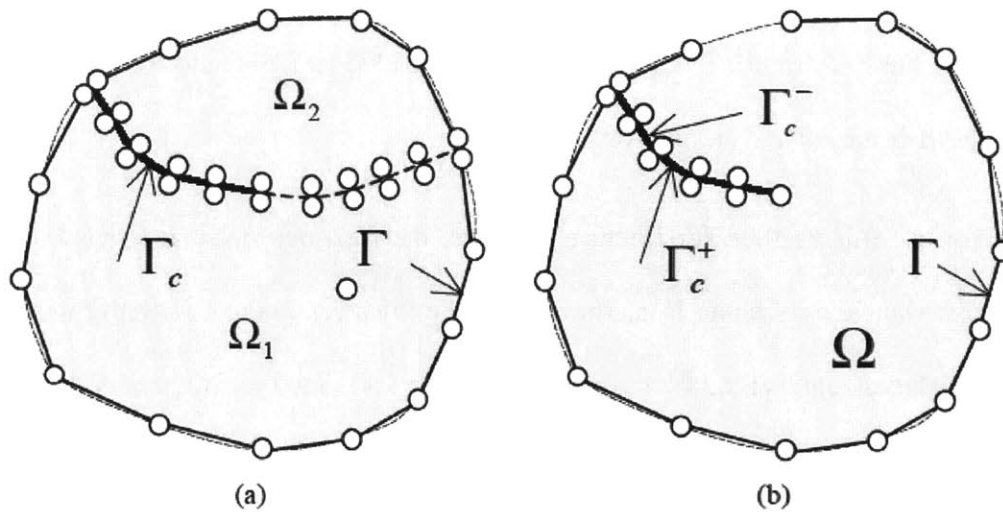


Figure 2.4 - Illustrative meshes for fracture analysis with BEM: (a) sub-domain, direct BEM; (b) single domain, dual BEM (Jing, 2003)

Discrete Element Method (DEM)

One of the original fields of DEM is rock mechanics. The first studies were conducted by Cundall (1971). The method has seen a wide variety of applications in rock mechanics, soil mechanics, granular materials, material processing, and fluid mechanics. This method can be used to represent block geometry and internal deformation of blocks (Figure 2.1 d).

Deng et al. (2011) state in their paper that DEMs can directly mimic rock and thus exhibit a rich set of emergent behaviors that correspond very well to real rock. Deng et al. (2011) implemented the DEM to handle rock deformation and fracturing processes. Rock is viewed, for instance, as a circular/spherical particle cluster with finite mass, and its mechanical performance is represented by the stiffness and strength of particles or bonds

between particles (Figure 2.5). The solid rock is treated as a cemented granular material of complex-shaped grains.

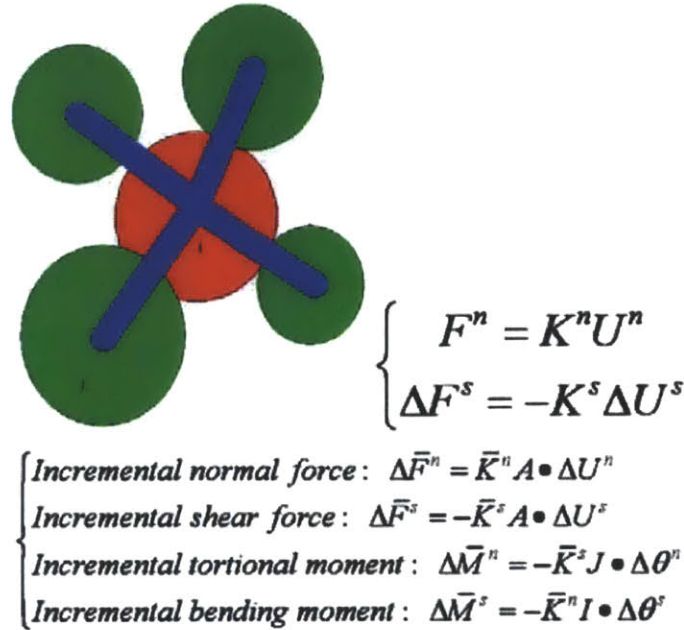


Figure 2.5 - Physical model of implemented DEM (Deng et al., 2011)

Discrete Fracture Network (DFN) methods

Among the methods for modeling fracture flow systems, the DFN approach is one of the most accurate, but also the most difficult to implement, as stated by many researchers. The DFN method is a special discrete model that can consider fluid flow and transport processes in fractured rock masses through a system of connected fractures (Jing, 2003). Like the DEM, this method was created from a need to represent more realistic fracture system geometries in 2-D and 3-D. Up to now they are widely used in applications to problems of fractured rocks, and they are an irreplaceable tool for modeling fluid flow

and transport phenomena. Table 2.1 shows a summary of the main fracture models based on DFN (Staub, 2002), and Figures 2.6 through 2.10 show some of the models presented in the table.

Due to its computational complexity these methods restrict fluid flow to the fractures and consider the surrounding rock as impermeable. With the progress in numerical techniques researchers are trying to model very complex systems of fractures and also take into account the fact that fractures can exchange fluid with the surrounding rock matrix (Reichenberg et al., 2006).

Table 2.1 - Main review of the DFN model, as regard to their applicability, advantages and limitations (Staub, 2002)

Model	Concept	Applicability	Advantages	Limitations
Orthogonal	Fracture network simulated from 3 sets of unbounded orthogonal joints	rock masses with completely defined rectangular rock blocks / mostly to hydrology	Simple geometry and treatment of data	Planar assumption, limitation in the variation of fracture orientation
Baecher disk	Generate fracture network from fracture centres that are distributed uniformly in space	Homogeneous rocks	Few field data available / Accurate in rock mechanics and hydraulics when a little is known	Do not simulate terminations of fractures, fractures must be planar / Do not account for complex features of fracture populations
Enhanced Baecher	Generate fractures from fracture centres located at random points in space. Intersections are calculated with pre-existing fractures	Fractured rock masses in which joint terminations are observed	Suited for simulation of connectivity of natural fracture population / Multiple intersections per fracture are possible	Fracture size distribution is not preserved; joints must be planar
BART	Same principle as for the Enhanced Baecher model, except that the centre of fracture terminating at intersections is generated from point on fracture intersection	Fractured rock masses in which fracture terminations are observed	Quick simulation / Fracture size distribution is preserved; spatial correlation in the simulated fracture population	fractures must be planar
Veneziano	Fracture network generated in 3 stochastic processes based on Poisson plane and Poisson lines	Suited for 100% persistent and unbounded fractures	Polygonal shapes are often observed in nature. More appropriate than orthogonal model for most cases, specially in case of coplanarity	Often fail to construct blocks / Intersections of fractures do not often match joint edges / Complex 3-D model
Dershowitz	Fracture network generated from 2 stochastic processes based on Poisson plane	Accurate for systems which exhibit distinct rock blocks, bounded polygonal fractures, and orientation dispersion	Model distinct rock blocks of various shape, flexibility in the distribution of fracture orientations / Joint intersection at joint edges	Can generate a large number of smaller polygons / Not so well fitted for coplanar fractures
Mosaic Tessellation	Deterministic and/or stochastic generation of the blocks, then definition of the fracture planes	Fracture systems resulting from a process of block formation (jointing in columnar basalts)	Manage non co-planarity, creation of the blocks first	Not so accurate in cases that do not display polyhedral blocks and polygonal fractures; blocks created first; indirect modelling of location, orientation and shape of fractures
Poisson Rectangle	Same concept as Enhanced Baecher except that fractures are rectangular	Same as Enhanced Baecher	Same as Enhanced Baecher	Specific conceptual model / Require a good knowledge of the rock mass geometry
Geostatistical	Generate fractures according to a specified variogram	Describe the spatial behaviour of regionalised variables of the fracture network	Account for a good spatial correlation	The size of the sampling area must be consequent compared to the study area
War Zone	Simulate higher densities of fractures between two major subparallel fractures	Simulation of fracture network in shear zones and in the surrounding rock mass	Binary model / identify "fracture zone" and "non-fracture zone"	Specific conceptual model / Require a good knowledge of the rock mass geometry
Non-Planar Zone	Generate fractures along a non-planar user defined surface	Simulation of fracture network along specific features (deformation zones,...)	Enhance rock zones with specific geometrical properties / binary model	Require a good knowledge of the rock mass geometry
Levy-Lee Fractal	Generate clusters of smaller fractures around wider fractures	In combination with geostatistical analysis: Hierarchical Fracture Trace Model	Accounts for the chronology of fracture formation / Ability to generate a non-stationary fracture system with a set of parameters that remain constant throughout the generation system	Do not consider the size and shape of the blocks delimited by the simulated fractures / Definition of the most appropriate fractal dimension
Nearest-neighbour	Fractures are organised into primary, secondary and tertiary groups, and are generated in this sequence	Can account for the generation of fracture network according to the theory of fracture genesis	Generate clusters of fractures around the primary group / More explicit than Levy-Lee model if fractures can be classified	Must have enough data to assess the different groups and chronology

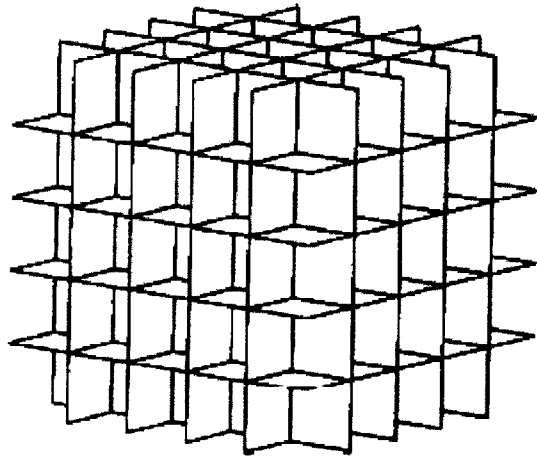


Figure 2.6 - Three-dimensional orthogonal model (Dershowitz and Einstein, 1988)

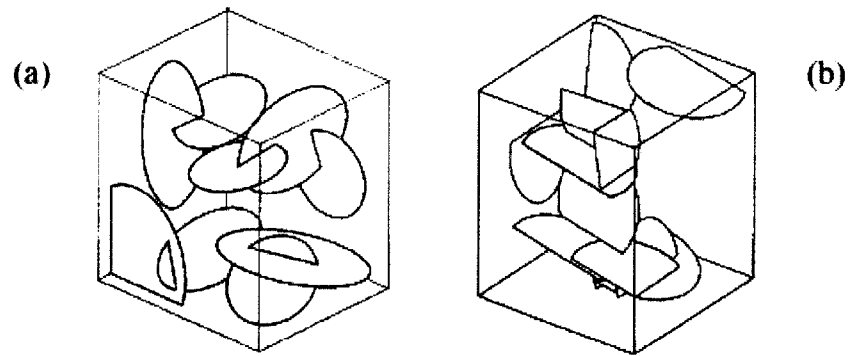


Figure 2.7 - Comparison of (a) the general Baecher model with (b) the Enhanced Baecher (Staub, 2002)

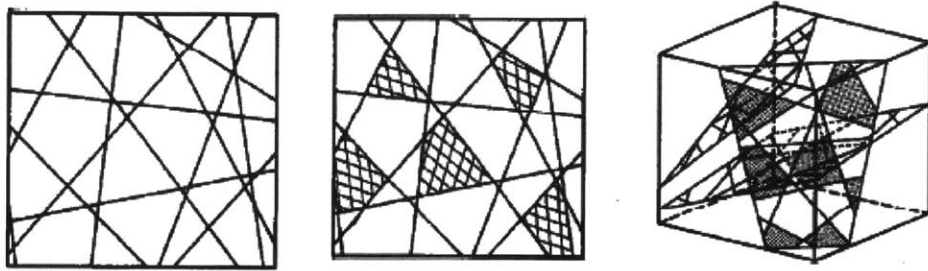


Figure 2.8 - "Generation of Veneziano joint system model" (Einstein, 1993)

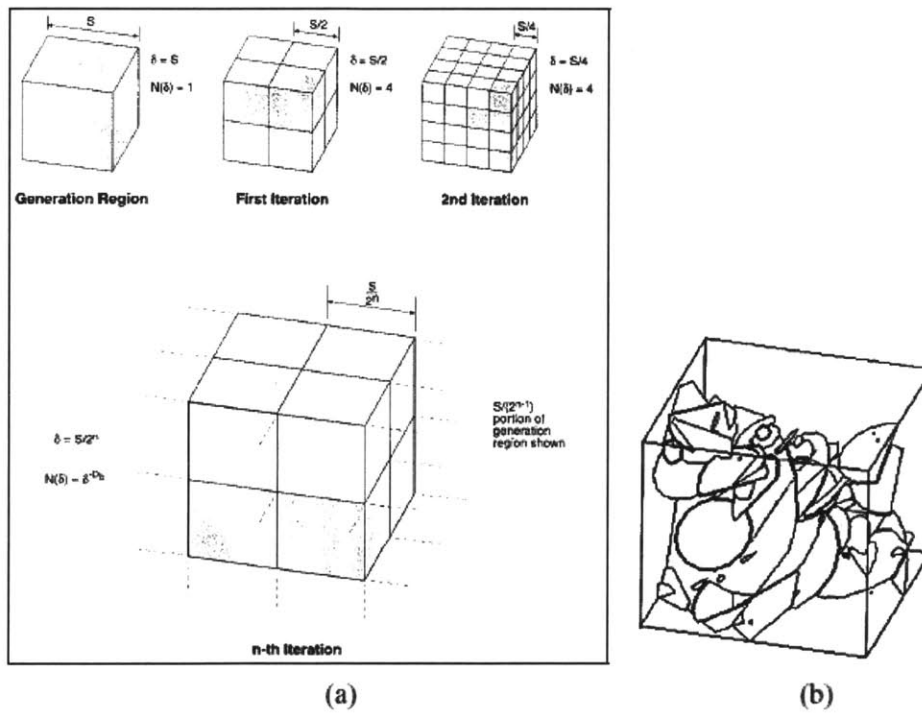


Figure 2.9 - (a) 3-D fractal Box algorithm, and (b) 3-D geometric model (Dershowitz et al, 1998)

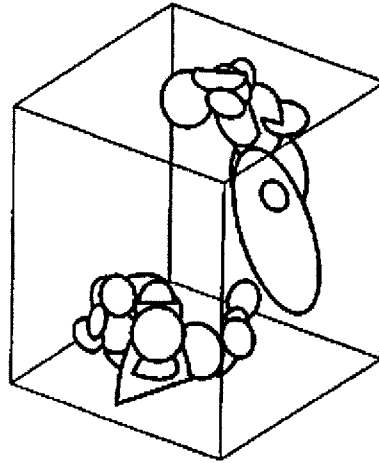


Figure 2.10 - 3D geometric Levy-Lee fractal model (Dershowitz et al, 1998)

Hybrid methods

Dershowitz (2006) describes the development of a hybrid model using DFN and EPM. The EPM (Equivalent Porous Media) volume elements are integrated with the DFN triangular elements. In the case presented in the paper the hybrid model was used to model shaft sinking at an underground laboratory. The schematic representation is shown in Figure 2.11.

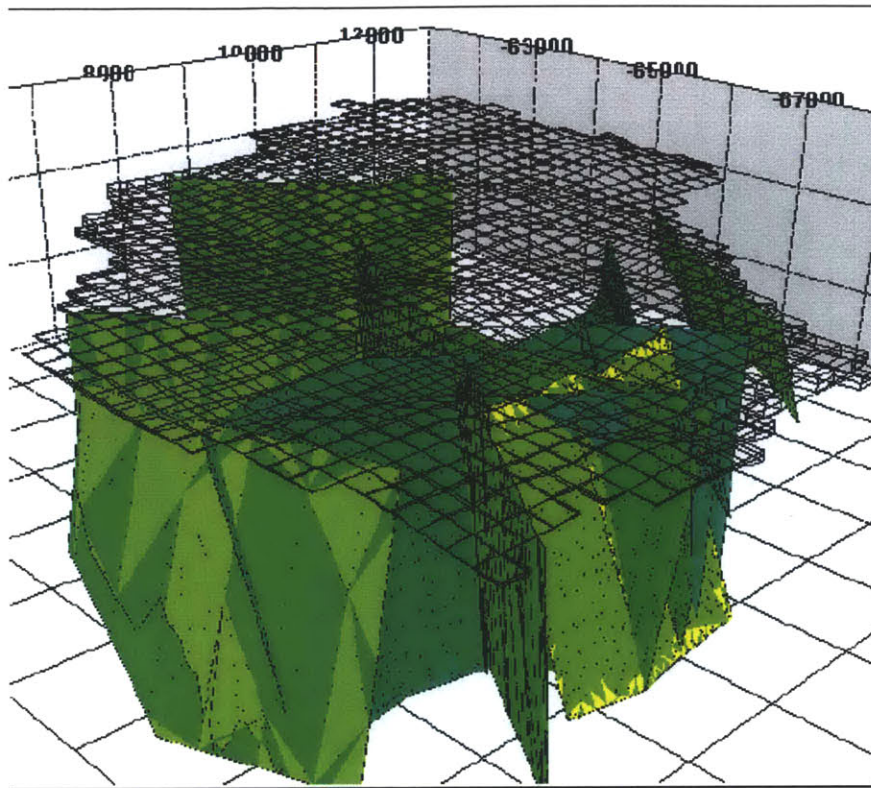


Figure 2.11 - Hybrid DFN/EPM Model Framework (Dershowitz, 2006)

The hybrid DFN/EPM model first implements the sedimentary strata using EPM volumetric elements (wire frame in Figure 2.11), with properties derived from well testing. Then the EPM volumetric elements are linked to DFN elements and the identified major faults are implemented using tessellated surfaces (green in Figure 2.11). Finally water conducting fractures are stochastically generated and implemented as DFN elements, with geometry and properties, based on interpretation of hydro-physical flow logs, packer tests, and borehole image logs (Figure 2.12) (Dershowitz,2006).

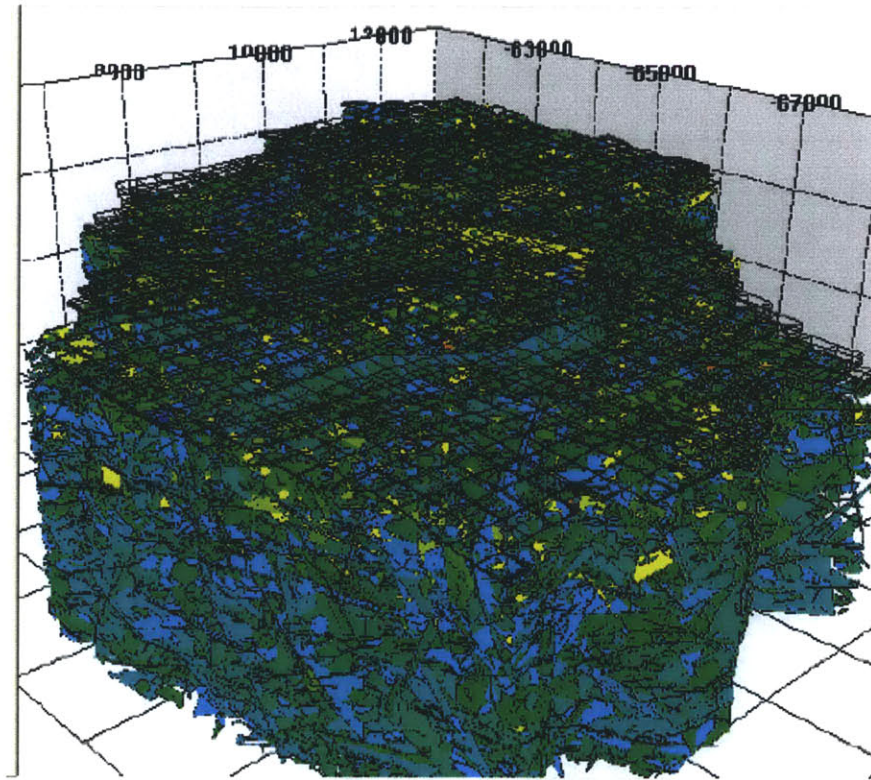


Figure 2.12 - Stochastically Generated Water Conducting Fracture Population (WCF) Throughout Model Region (Dershowitz, 2006)

2.1.2 Interpretation of fracture intensity

Dershowitz and Herda (1992) introduce in their paper a class of fracture intensity measures in 1-D, 2-D and 3-D. In three dimensions fracture intensity can be defined in terms of P31, the number of fractures in a volume; P32, the surface area of discontinuities per unit volume or P33; the volume of fractures in a volume (Figure 2.13).

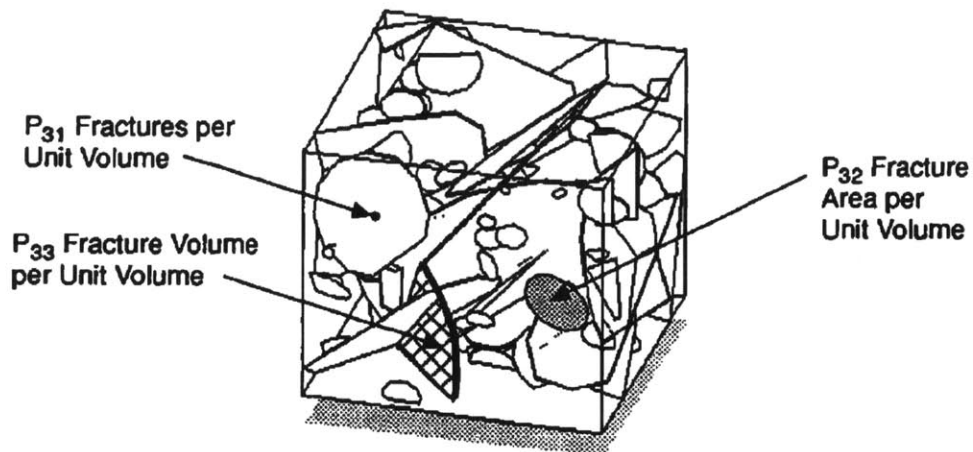


Figure 2.13 - Three dimensional fracture intensity measures (Dershowitz and Herda, 1992)

The most useful measure of intensity for three-dimensional fracture modeling is P_{32} , since it does not reflect any orientation effect. The size distribution and the number of discontinuities are needed for calculating intensity. Zhang and Einstein (2000) present in their paper methods for estimating the size distribution and the number of discontinuities. The discontinuity size distribution can be inferred from the trace data sampled in circular windows by using the stereological relationship between the true trace length distribution and the discontinuity diameter distribution for area (or window) sampling (Warburton, 1980). Zhang and Einstein (2000) present a method for estimating the true trace length distribution which is affected by bias when measured. Several types of bias occur during sampling: e.g., orientation bias, size bias, truncation bias, censoring bias. They analyze the biases using statistical tools for lognormal, negative exponential and gamma distributions of discontinuity diameters. In order to estimate the total number of discontinuities the approach of Mauldon and Mauldon (1997) is used. This approach

estimates the probability that a discontinuity with its centroid in the objective volume will intersect the wall of a borehole (Figure 2.14).

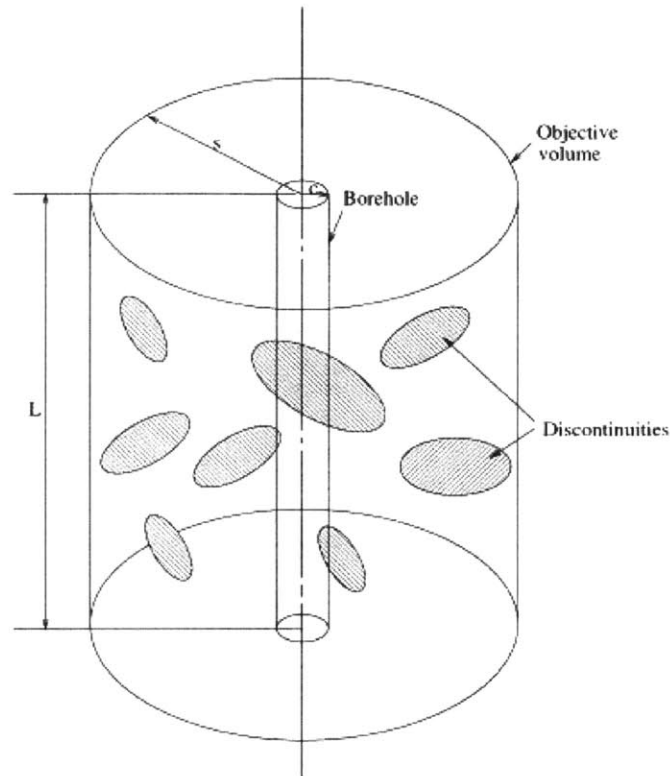


Figure 2.14 - Vertical borehole in an objective volume (Zhang, Einstein, 2000)

In order to describe both the intensity and the orientation distribution several tensor methods have been described. Those described by Oda (1982) and Kawamoto (1988) seems to take advantage of the concept of P_{32} and has a clear physical meaning.

P_{32} is one of the principal inputs in GEOFRAC, so its correct estimation is very important. P_{32} is used as intensity of Poisson planes in the Controlled Volume in the

primary stochastic process in GEOFRAC. More information about the generation of planes in GEOFRAC is presented in the next chapter.

2.1.3 Models for the aperture of the fractures

Fluid flow in fractures is strongly dependent on the fracture aperture. For this reason it is very important, especially for geothermal applications, to have a model that well represents the geometric characteristics of the fractures. The relation between aperture and flow is commonly expressed in terms of the parallel plate laminar flow solution (Poiseuille equation) through a cubic law (*Equation 2.1*):

$$q = \frac{h^3 \Delta P}{12 \mu \Delta L} \quad \text{Equation 2.1}$$

where

h: aperture of the fractures (m)

ΔP : pressure drop (Pa)

μ : fluid dynamic viscosity (Pa s)

ΔL : length of flow (m)

using a hydraulic aperture related to the mean mechanical fracture aperture. The hydraulic aperture is a non-linear function of effective normal stress, the fracture morphology, material properties and its history.

It is important to emphasize that the terminology used by different authors can be confusing. In fact, some authors refer to the aperture as the “width” or “opening”. In this thesis I will always use “aperture” for the distance between two separate fractures surfaces.

Many researchers such as Stone (1984), Vermilye et al. (1995) and Johnston and Mccaffrey (1996), through observation of fracture properties from field mapping, have proposed a power-law correlation between aperture h and fracture length l (Equation 2.2).

$$l = ah^b \quad \text{Equation 2.2}$$

where b varies between 0.6 and 1 and a varies between 20 and 2000. Length and aperture are in mm.

Tezuka and Watanabe (2000) used the Vermilye and Scholz (1995) equation to model the fracture network of the Hijiori hot dry rock reservoir. The aperture (in meters) is defined as shown in Equation 2.3:

$$a = \alpha * \sqrt{r} \quad \text{Equation 2.3}$$

where a is the fracture aperture, r is the fracture radius, and α is the factor that controls the relationship between the aperture and the radius. They conducted a sensitivity analysis for both the parameter α and r_{max} in order to find the most appropriate values. After comparison between a simulated flow and field observations, the values that they chose for the two parameters were:

$$\alpha = 4.0 * 10^{-3}$$

$$r_{max} = 200 \text{ m}$$

Ivanova et al. (2012) assume that fracture aperture (in meters) can be related with fracture length by a power-law function (Equation 2.4):

$$h = \alpha(2R_e)^\beta \quad \text{Equation 2.4}$$

where R_e (in meters) is the equivalent radius of the sphere that circumscribes the fracture (polygon), h is the aperture, and a and b are coefficients that depend on the site geology and that can be found in the literature (Vermilye and Scholz, 1995; Stone, 1984; Vermilye et al., 1995; Johnston and Mccaffrey, 1996).

Other researchers, such as Dverstop and Andersson, 1989; Cacas, et al., 1990 assume that the hydraulic aperture of fractures, h (in meters), follows a lognormal distribution, which can be written as:

$$f(h) = \frac{1}{h\sigma\sqrt{2\pi}} e^{-\frac{(\ln h - \mu)^2}{2\sigma^2}}, 0 \leq h \leq \infty \quad \text{Equation 2.5}$$

where $f(h)$ is the lognormal distribution of the aperture, h , with parameters μ and σ .

Ivanova et al. (2012) referring to these studies, implemented in their model this probabilistic approach but assuming that it as a truncated lognormal distribution that follows these relation:

$$f_{TR}(h) = \frac{f(h)}{\int_{h_{min}}^{h_{max}} f(h)d(h)}, h_{min} \leq h \leq h_{max} \quad \text{Equation 2.6}$$

Where h_{\min} and h_{\max} (in meters) are the minimum and the maximum aperture values. This relation is presented in Figure 2.15.

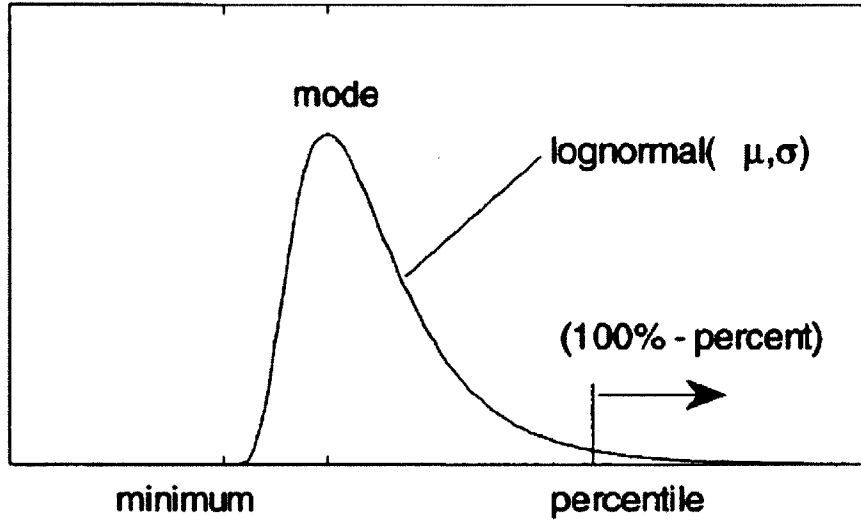


Figure 2.15 - Truncated lognormal distribution (Ivanova et al. 2012)

2. 2 FLOW SYSTEM MODELS

This section summarizes the numerical models describing fluid flow in fractured porous media. As reported by Diodato, 1994, four conceptual models have dominated the research:

- 1) Explicit discrete fractures
- 2) Dual continuum
- 3) Discrete fracture networks
- 4) Single equivalent continuum

The **Explicit Discrete Fracture model** allows one to explicitly represent fluid potential gradients and fluxes between fractures and porous media with minimal non-physical parameterization. As Diodato (1994) explains the data acquisition with this model can become onerous where large numbers of fractures need to be represented.

Travis (1984) represents fractures with orthogonal orientation. An implicit finite difference formulation is used and solved iteratively. The region of interest is represented by a computational mesh of rectangular cells as shown in Figure 2.16. The rows, columns, and layers of the cells are not equally spaced. Some variables such as pressure, density, concentration, and saturation are evaluated at the cell centers; others (velocity components) are evaluated at cell interfaces.

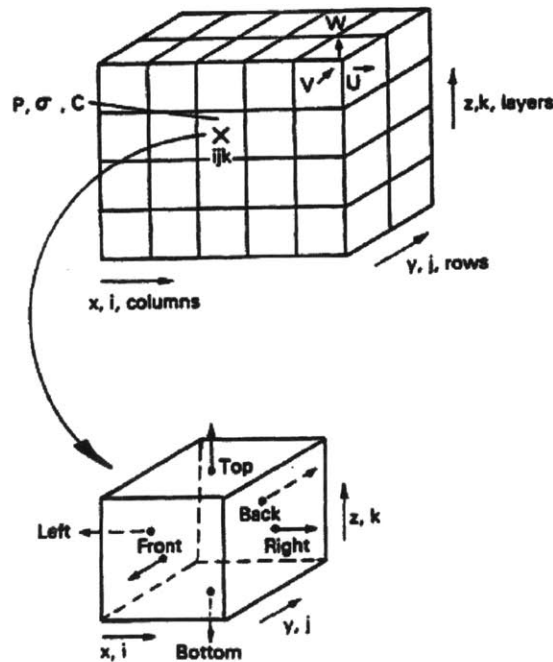


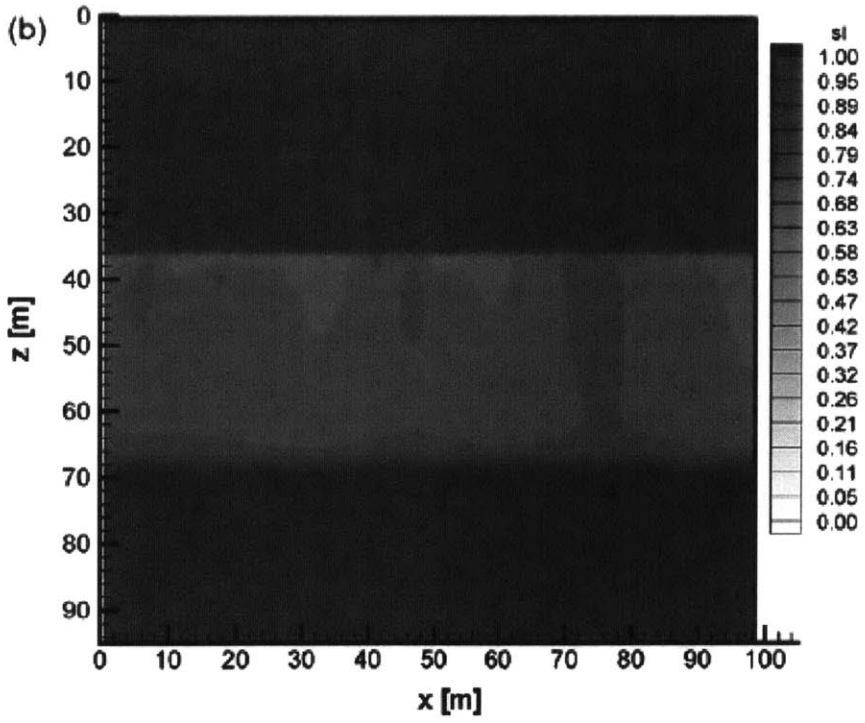
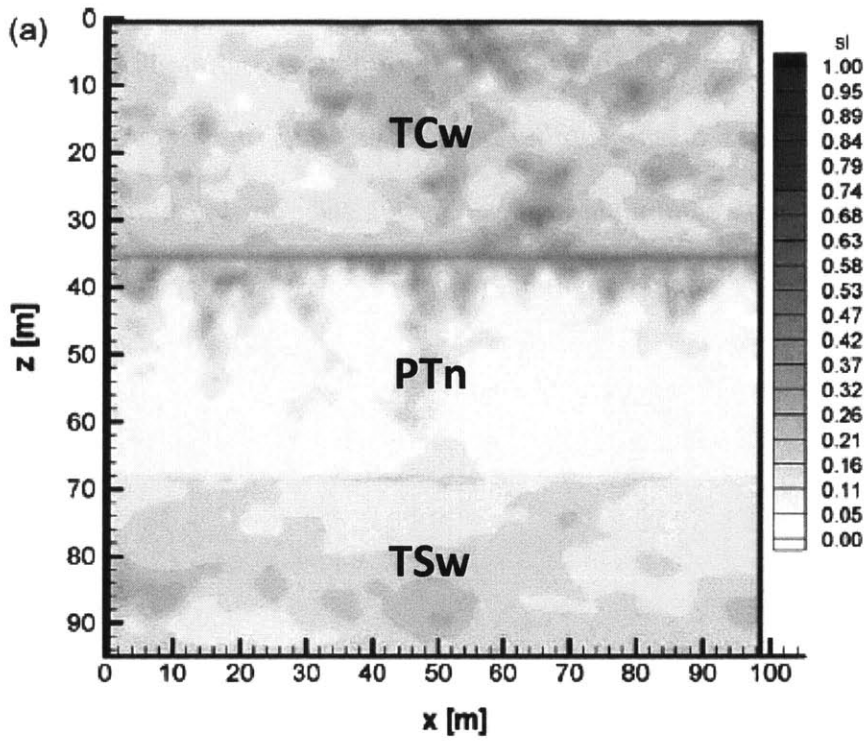
Figure 2.16 - Typical computational mesh (Travis, 1984)

Travis implemented an algorithm based on finite difference to create the TRACR3D code that was used to model time-dependent mass flow and chemical species transport in a three-dimensional, deformable, heterogeneous, reactive porous/fractured medium (Travis, 1984).

The **Dual-continuum** approaches are based on an idealized flow medium consisting of a primary porosity created by deposition and lithification and a secondary porosity created by fracturing, jointing, or dissolution (Warren and Root 1963). The first studies were introduced by Barenblatt et al. (1960) and later extended by Warren and Root (1963). The porous medium and the fractures are envisioned as two separate but overlapping continua. Fluid mass transfer between porous media and fractures occurs at the fracture-porous medium interfaces (Diodato, 1994).

More recent studies such as that by Illman et al. (2004) used a dual continuum two phase flow simulator called METRA to represent the matrix and the fractures as dual overlapping continua; liquid flux between continua are restricted by a uniform factor. Figure 2.17 shows an example of the steady state distribution of fracture saturation (Figure 2.17a), matrix saturation (Figure 2.17 b), fracture water flux (Figure 2.17 c), and matrix water flux (Figure 2.17 d) for a single realization of fracture permeability with $q_a=42.5$ mm/yr, where q_a is the water flux. In Figure 2.17 the materials, used in the test, are defined with the acronyms: Tiva Canyon Tuff (TCw), non-welded Paintbrush Tuff (PTn), and welded Topopah Spring Tuff (TSw). Illman et al., 2004 state: *“The distribution of saturation in the fracture continuum is highly variable in the nonwelded*

and welded units. The saturation is highest locally, where permeability in the fracture continuum is low and at the TCw/PTn boundary in the fracture continuum, where the contrast in permeability causes a permeability barrier. As water moves progressively from the fracture to the matrix continuum in the PTn unit, the variability in saturation decreases with depth in the PTn fracture continuum”.



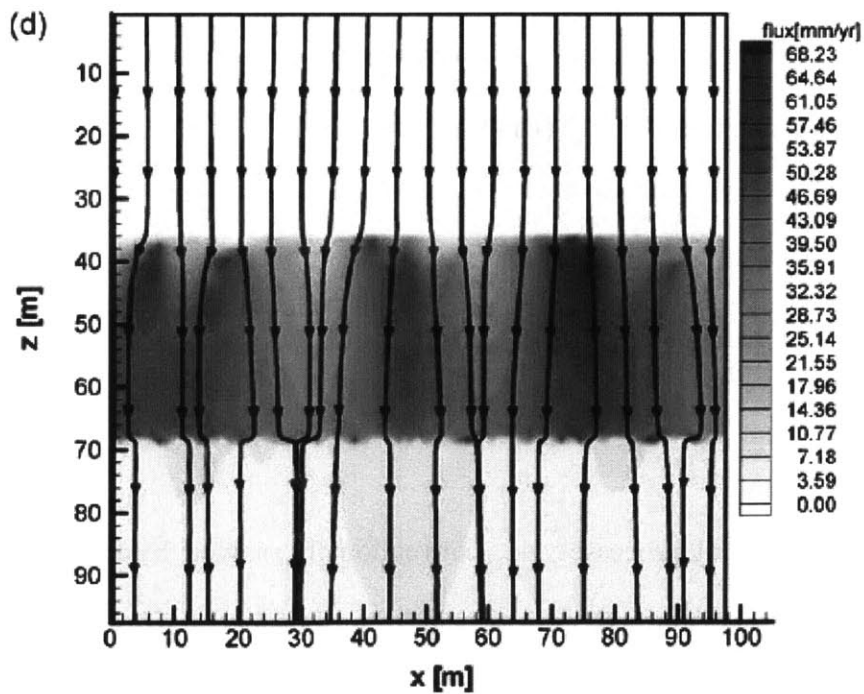
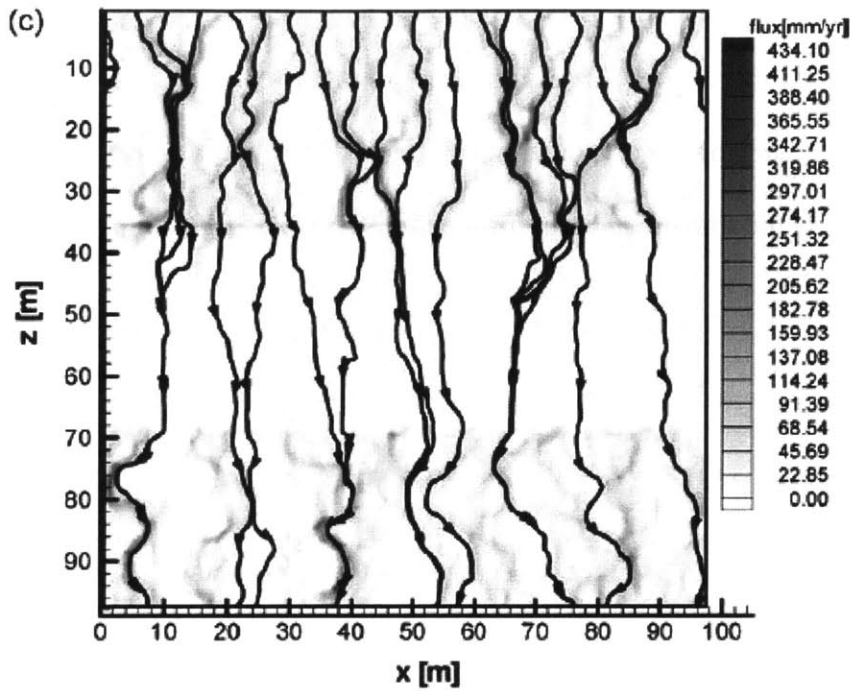


Figure 2.17 - Steady state distribution of: (a) fracture saturation; (b) matrix saturation; (c) fracture water flux; and (d) matrix water flux $q_a=42.5$ mm/yr (Illman et al., 2005)

In **Discrete-fracture-networks** all flow is restricted to the fractures. This idealization reduces computational resource requirements. Fractures are often represented as lines or planes in two or three dimensions. For contaminant transport, some network models allow for diffusion between the fracture and porous medium (Diodato, 1994).

The model developed at MIT that will be introduced in Chapter 3 belongs to this category. Fractures are represented by polygons in three-dimensions, the porous medium is not represented and all flow is restricted to the fractures.

The **Single Equivalent Continuum Formulation** assumes that the volume of interest is considered to be large enough that, on average, permeability is a sum of fracture and porous media permeability. Pruess et al. (1990) demonstrated that where the scales of integration are sufficiently large, the single equivalent continuum approximation will model well the conserving fluid mass. It may, however, be a poor predictor of spatial and temporal distributions of contaminant fluxes (Diodato, 1994).

2.3 SOFTWARE TOOLS IN USE AND THEIR APPLICATION

Thermal-hydrological-mechanical processes are, conventionally, solved by coupling a fracture system model with a subsurface flow and heat transfer model. The term ‘coupling’ implies that one process affects the initiation and progress of another. It is possible to use coupling processes using the same software, as the Rock Mechanics

Group at MIT is trying to create with GEOFRAC or using different software that model different part of the thermo-hydro-mechanical processes; for example, Rutquist et al. (2002) used FLAC, a rock mechanics simulator and TOUGH2, a widely used flow and heat transfer simulator, for their simulation. The two simulators run sequentially with the output from one code serving as input to the other one (Podgorney et al., 2010).

2.3.1 TOUGH2

TOUGH ("Transport Of Unsaturated Groundwater and Heat") was developed at the Lawrence Berkeley National Laboratory (LBNL) in the early 1980s primarily for geothermal reservoir engineering. Now it is widely used for many other applications for instance nuclear waste disposal, environmental assessment and remediation.

TOUGH2 is the basic simulator for non-isothermal multiphase flow in fractured porous media. The TOUGH2 simulator was developed for problems involving strongly heat-driven flow. It takes into account the fluid flow in both liquid and gaseous phases occurring under pressure, as well as viscous, and gravity forces according to Darcy's law. Interference between the phases is represented by means of relative permeability functions. The code includes Klinkenberg effects and binary diffusion in the gas phase and capillary and phase adsorption effects for the liquid phase. Heat transport occurs by means of conduction (with thermal conductivity dependent on water saturation), convection, and binary diffusion, which includes both sensible and latent heat.

2.3.2 FALCON

FALCON (**F**racturing **A**nd **L**iquid **C**ONvection) is a finite element based simulator solving fully coupled multiphase fluid flow, heat transport, rock deformation, and fracturing using a global implicit approach (Podgorney et al. 2012). It was developed and now improved by the Idaho National Laboratory group.

Podgorney et al. (2010) describe the initial code. The approach is to develop a physics based rock deformation and fracture propagation simulator by coupling a discrete element model (DEM) for fracturing with a continuum multiphase flow and heat transport model. In this approach, the continuum flow and heat transport equations are solved in an underlying finite element mesh with evolving porosity and permeability for each element that depends on the local structure of the discrete element network. As a first step in the development of the code, governing equations for single-phase flow and transport of heat are being coupled with linear elastic equations. The basic architecture of the code allows one to conveniently couple different processes and incorporate new physics, such as stress dependent permeability-porosity, phase change, implicit fracturing. The code in FALCON is developed using a parallel computational framework called MOOSE (**M**ultiphysics **O**bject **O**riented **S**imulation **E**nvironment) developed at the Idaho National Laboratory (INL).

Podgorney et al. (2012) present some results. Figure 2.18 shows the simplest problem geometries, a small fracture network consisting of two horizontal and one vertical fracture.

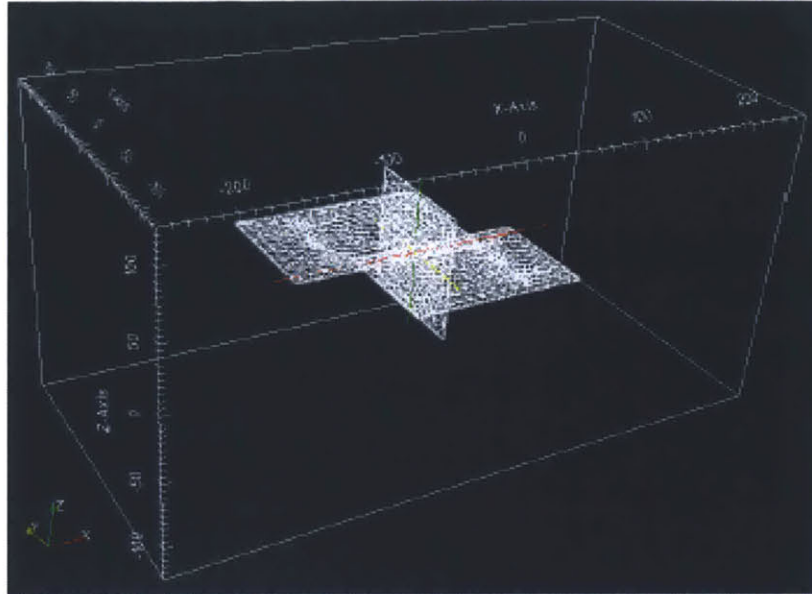


Figure 2.18 - Three-dimensional mesh for the simplest fracture flow problems under consideration (Podgorney et al., 2012)

Figure 2.19 shows the simulated temperature and pressure for two examples after several years of injection and production. Cold fluid is injected on the right side of the fracture domain, while production is on the left side. At early times, small changes are observed in the temperature field in the proximity of the production location.

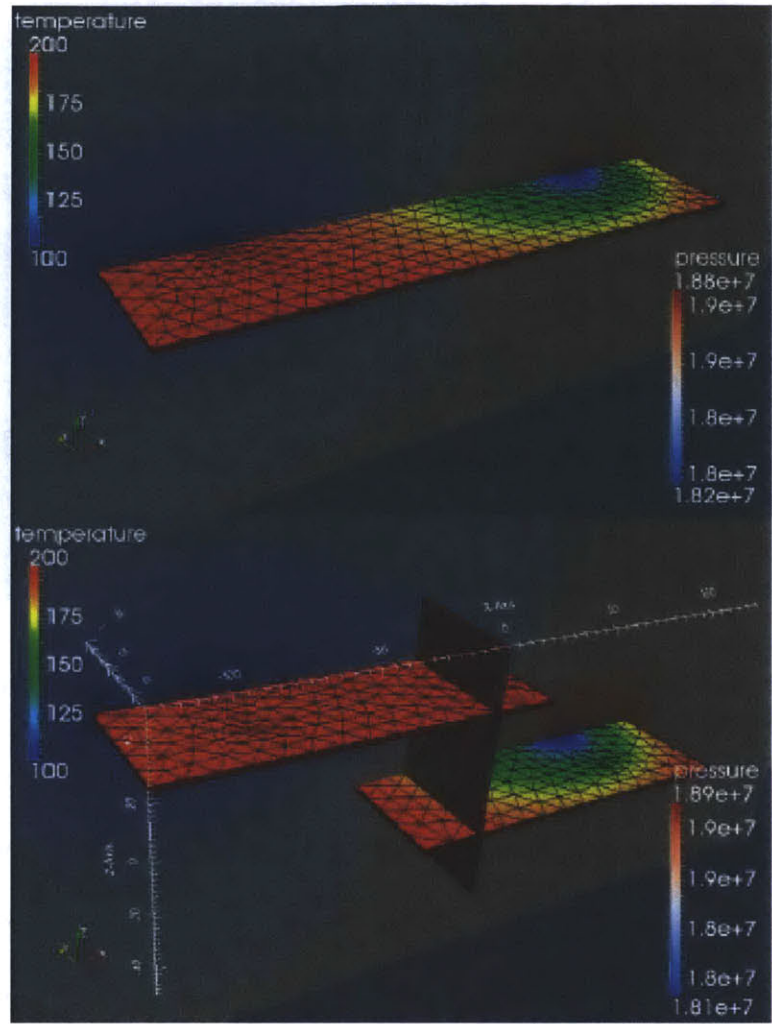


Figure 2.19 - Temperature (in fracture domain) and pressure along the center of the reservoir matrix domain for tow simulations. (Podgorney et al., 2012)

2.3.3 LEAPFROG

Leapfrog Geothermal is a 3D modeling tool that can be used in every stage of a project, from initial proof-of-concept to reservoir development and production. In 2010 ARANZ Geo together with the University of Auckland, Department of Engineering Science

Geothermal Group and Contact Energy Ltd used the core technology to develop a geothermal “product”.

Leapfrog Geothermal can be used for integration with Tough2 for flow modeling, regional geothermal resource evaluation, geothermal model review and maintenance, borehole planning for exploration, development and reservoir management, 3D fault.

2.3.4 FRACMAN

FracMan generates fractures in three dimensions within a given rock volume. Dershowitz et al, (1998) explain the definition of the input parameters and the theoretical background of Discrete Fracture Network introduced the DFN models into the FracMan. To take into account the variability of the input parameters, the DFN model is generated several times by means of Monte Carlo simulations, and the fracture population statistics analyzed for each simulation model (Staub I. et al., 2002). FracMan generates 3-D fracture network models to describe the pattern of faults, fractures, solution features and stratigraphic contacts in fractured rock.

2.3.5 CONCLUSIONS

The software tools presented above are very useful tools to be used in the field of geothermal energy. The disadvantage is that they need to be combined in order to obtain a complete fracture flow model simulation.

The program GEOFRAC, developed by the MIT Rock Mechanics Group and presented in Chapter 3, aims to model a geothermal reservoir as a complete and optimized tool. Particularly the well location optimization study that will be implemented as a next step, is quite innovative.

CHAPTER 3

GEOFRACT: 3-D HYDRO-MECHANICAL MODEL

3.1 INTRODUCTION

GEOFRACT is a three-dimensional, geology-based, geometric-mechanical, hierarchical, stochastic model of natural rock fracture systems (Ivanova, 1998). Fractures are represented as a network of interconnected polygons and are generated by the model through a sequence of stochastic processes (Ivanova et al., 2012). This is based on statistical input representing fracture patterns in the field in form of the fracture intensity P_{32} (fracture area per volume) and the best estimate fracture size $E[A]$. P_{32} can be obtained from spacing information in boreholes or from observations on outcrops using the approach by Dershowitz and Herda (1992). Best estimate fracture size $E[A]$ can be obtained from fracture trace lengths on outcrops with suitable bias corrections as developed by Zhang et al. (2002). Distributions of fracture size can also be obtained subjectively. GEOFRACT has been applied and tested by estimating the fracture intensity and estimated fracture size from tunnel records and from borehole logs (Ivanova et al. 2004, Einstein and Locsin 2012). Since its original development, GEOFRACT has been made more effective by basing it on Matlab, and it has been expanded by including an

intersection algorithm and, most recently, a flow model. Focus of my research was to apply GEOFRAC in the EGS (Enhanced Geothermal System) modeling field. In this chapter I will present the basic concept of GEOFRAC and I will introduce the applicability in the geothermal field. The algorithms developed to apply GEOFRAC for the geothermal area are explained in more details in the next chapters.

A parametric study was conducted in order to test the efficiency of this model and to analyze the sensitivity of the output flow rate to the parameters used as input in the model. Chapter 4 will present the results of the parametric study.

3.2 BASIC CONCEPT

3.2.1 Fracture system model

The fracture system model in GEOFRAC was developed by Ivanova (1998). The concept is a three-dimensional geometric-mechanical model that represents rock fracture systems. The model has the characteristics to be hierarchical, so that fractures are grouped into hierarchically related fracture sets; and it is stochastic, using statistical methods to generate the fracture system from available geologic information. Fractures in GEOFRAC are represented as polygons (Figure 3.1). As shown in the figure each polygon is characterized by a pole and a radius R_e that represent the radius of the equivalent circle that circumscribes the polygon.

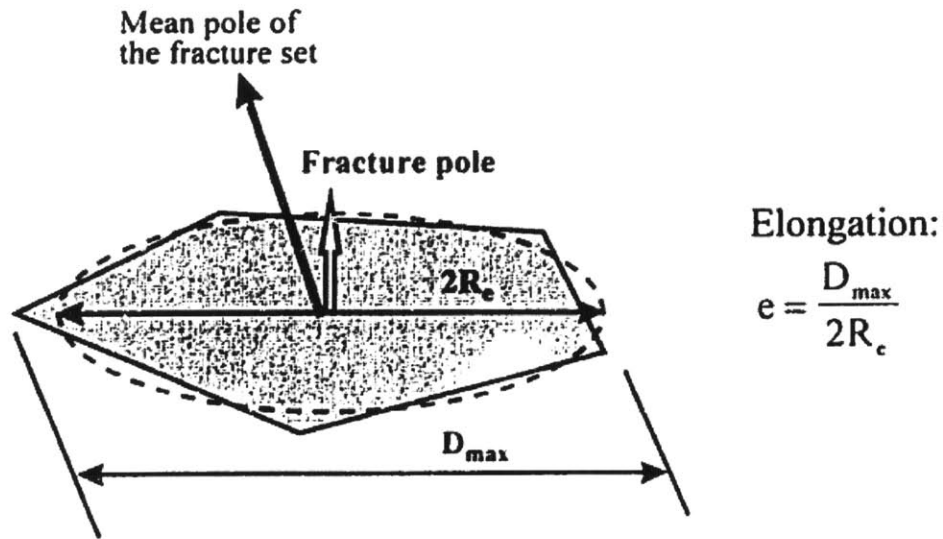


Figure 3.1- Fracture represented as a polygon with a pole and a radius (Ivanova, 1995)

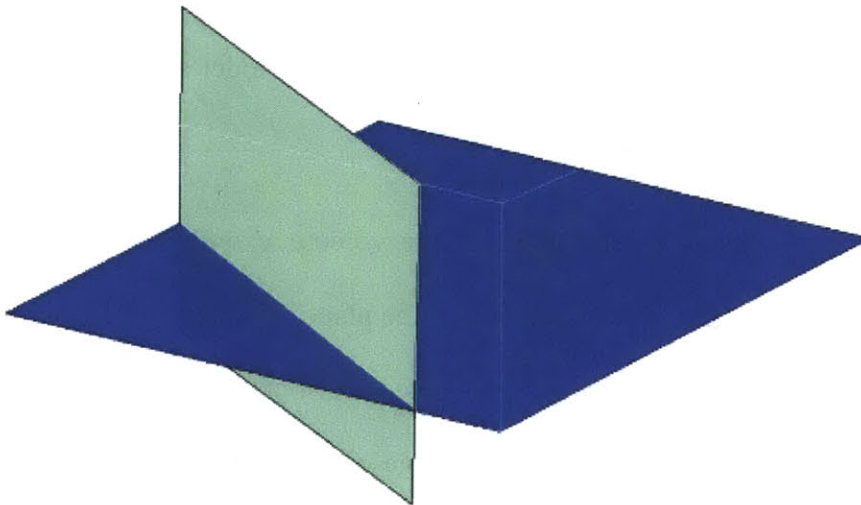
The desired mean fracture size $E[A]$ and fracture intensity P_{32} in a region V are given as input. GEOFRAC uses these inputs to generate the fracture system following a sequence of stochastic processes (for details on GEOFRAC and on the flow model see Ivanova et al. 2012, Sousa et al. 2012 and Sousa, 2013):

Primary Process: Fractures planes are generated in the volume V with a Poisson plane process of intensity μ where $\mu = P_{32}$. The orientation of the planes can be specified with the Fisher distribution. Recall that this is a single parameter distribution. Low values of the parameter κ simulate randomly generated planes; large κ will generate planes mostly parallel to each other.

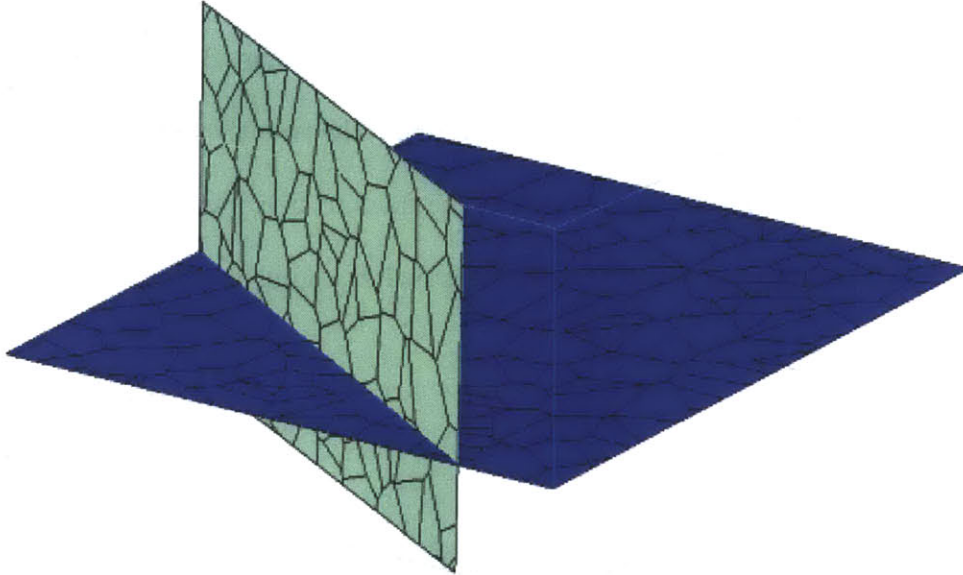
Secondary Process: A Poisson point process with intensity λ is generated on the planes and the fractures are created with a Delaunay-Voronoi tessellation. It represents fracture intensity variation by size and location.

Tertiary Process: Random translation and rotation of the fractures (polygons) are conducted to represent the local variation of fracture position and orientation of individual fractures. In the tertiary process a new algorithm was recently added to the model to allow the user to model fractures with or without random rotation. The parametric study presented in the Chapter 4 will compare results with rotation and no rotation of the fractures.

The generation process of the fractures is visualized in Figure 3.2.



a) Primary Process: Generation of Planes



b) Secondary Process: Division of planes into polygons



c) Tertiary Process: Random translation and rotation

Figure 3.2 - Generation of a fracture set with the GEOFRAC model. Primary process (a), secondary process (b), tertiary process (c) (Ivanova et al., 2012)

A parametric study was conducted in order to study the relationship between fractures intensity, size, and connectivity (Ivanova et al., 2012). Monte Carlo simulation was used in order to determine the mean fracture connectivity C , with the variation of P_{32} and $E[A]$. The results of the simulations are graphically presented in Figure 3.3. The plot shows the results for $\kappa=0$ (solid line) and for $\kappa=10$ (dotted line). The results suggest that the fracture connectivity C is a non-linear function of both the fracture size, measured by the expected mean area $E[A]$, and the fracture intensity, measured as cumulative fracture area per unit rock volume, P_{32} .

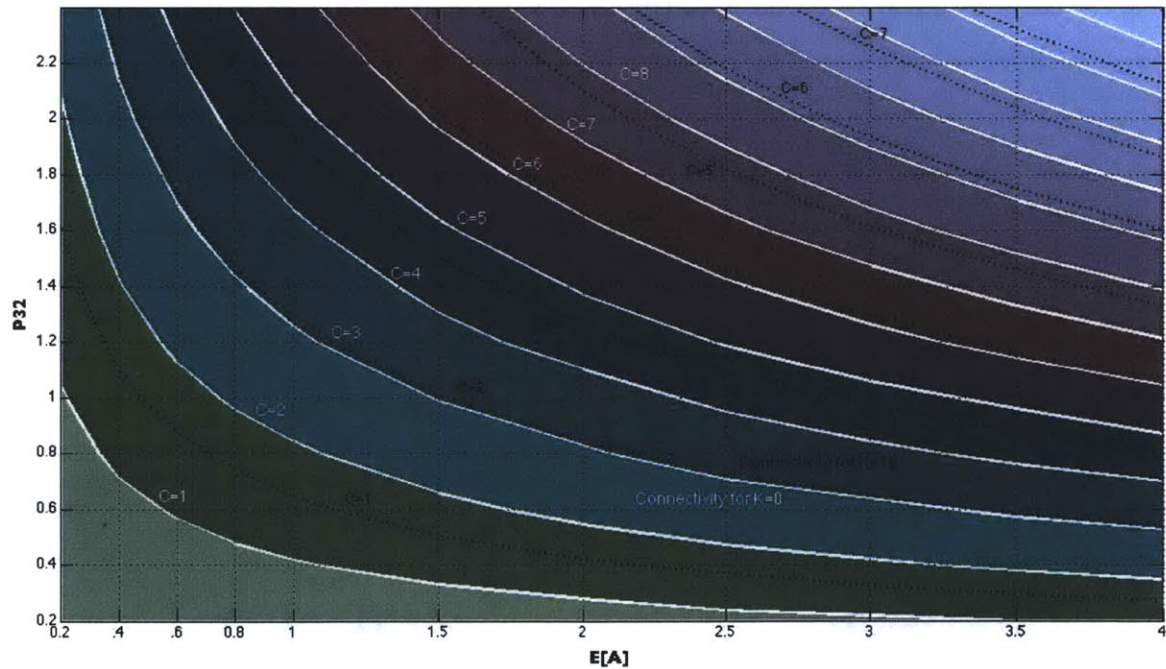


Figure 3.3 - Expected fracture connectivity, C , for given $E[A]$ and P_{32} (Ivanova et al., 2012)

3.2.2 Flow system model

In section 2.1.2 I described the 4 methods that can be used to represent fracture flow. The circulation model in GEOFRAC belongs to the category of Discrete-Fracture Networks. In this type of model the porous medium is not represented and all flow is restricted to the fractures. Fractures are often represented as lines or planes in two or three dimensions. In this specific case the fractures are represented by polygons in three dimensions. The flow model in GEOFRAC was developed by Sousa (Sousa, 2012).

The flow equations used to model the flow through the fractures are those of linear flow between parallel plates. The water flows only in the x direction between two parallel plates with the no-slip condition for viscous fluids forming the velocity profile in the y direction (Figure 3.4).

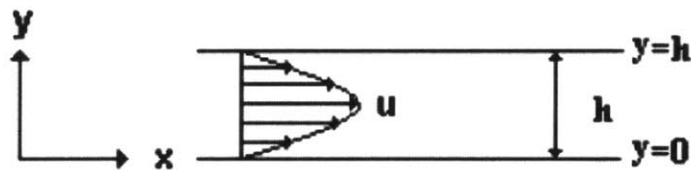


Figure 3.4- Schematic representation of linear flow between parallel plates

The water flow in fracture is assumed to be governed by the Poiseuille cubic law (Zimmerman and Bodvarsson, 1996) represented by the following equation:

$$q = \frac{h^3 \Delta P}{12 \mu \Delta L} \quad \text{Equation 3.1}$$

Where:

q is the volumetric flow rate (m^2/sec) per unit width;

h is the aperture of the fracture in m;

μ is the fluid dynamic viscosity in PA s;

ΔP is the pore pressure change in Pa after the flow travels through distance ΔL .

Considering the fracture width, $w(s)$ (see Figure 3.5) (in m, variable with length) the equation for flow between parallel plates is:

$$q = \frac{w(s)h^3 \Delta P}{12\mu \Delta L} \quad \text{Equation 3. 2}$$

The schematic representation of the fracture width is shown in Figure 3.5.

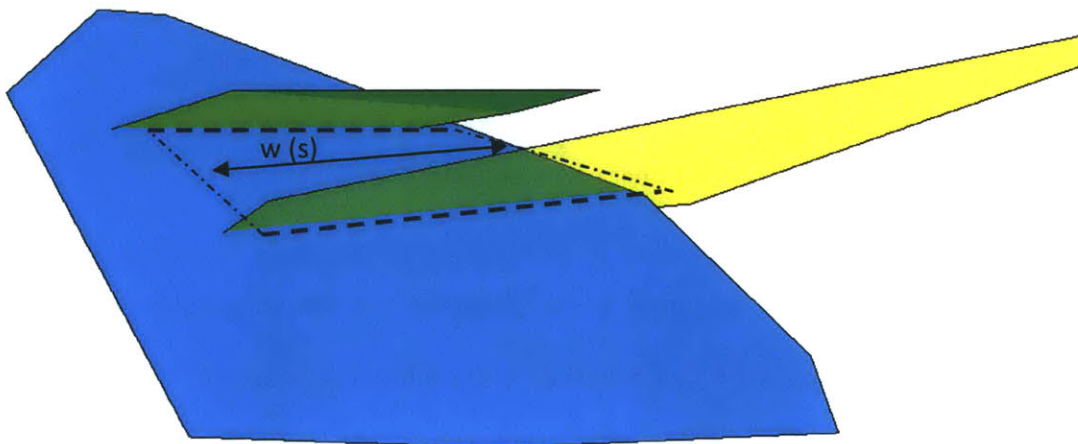


Figure 3.5 – Mean fracture width between fractures intersection

The fracture aperture in GEOFRAC can be modeled in three different ways:

- Deterministic approach using a power law relation between fracture length and aperture.
- A probabilistic approach based on the truncated lognormal distribution.
- A fixed value. This was used to perform the parametric study that will be presented in the Chapter 4.

These methods are explained in more detail in Chapter 6.

In order to calculate the geometric flow paths and the flow rate GEOFRAC follows seven steps:

- 1) Determination of the fractures that intersect the left boundary;
- 2) Determination of the fractures that intersect the right boundary;
- 3) Determination of the score of each fracture;
- 4) Determination of the highest score paths amongst all initial fractures (that intersect the left boundary) and all end fractures (that intersect the right boundary);
- 5) Determination of the intersections between the paths;
- 6) Determination of the flow through each path;
- 7) Determination of the total flow through the reservoir;

I will briefly describe the above steps that are more thoroughly explained in Sousa (2012).

Step 1 & 2 – Select and store the fractures that intersect the injection and the production boundaries

A function called `buildNodesList.m` was created. This function creates an $N \times 2$ matrix containing a list of edge connections.

Figure 3.6 shows an example of how the connections are considered and stored in the matrix

E:

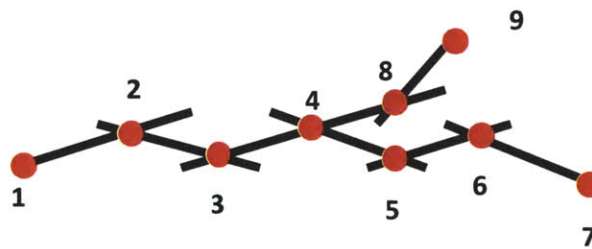


Figure 3.6 - Representation of the fracture intersections (Sousa, 2013)

The matrix containing the list of edge connections in this case would be as follows:

$$E = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \\ 3 & 8 \\ 4 & 8 \\ 4 & 5 \\ 8 & 9 \\ 5 & 6 \\ 6 & 7 \end{bmatrix}$$

Step 3 - Scoring system:

The model at this point assigns a score to each fracture. The scoring system is based on the concept that for the same drop of pressure the flow rate is proportional to $\frac{wh^3}{\Delta L}$.

The score for each fracture is calculated as:

$$SC = \frac{\bar{w}h^3}{\Delta L} \quad \text{Equation 3.3}$$

where:

\bar{w} is the mean width of the fracture in m;

h is the aperture of the fracture in m;

ΔL is the distance between fractures (See Figure 3.7).

The fractures with greater aperture and greater mean width will have a greater volumetric flow for the same drop of pressure.

Figure 3.8 illustrates the different geometric components of the scoring formula. The length of a “fracture path” corresponds to the distance between the middle points of the intersections between fractures. For example in Figure 3.7 the length ΔL of fracture 1 is the distance between the middle point of the intersection of fracture 1 and 3 and the middle point of the intersection of fracture 1 and 2. Figure 3.8 shows the score components when a fracture intersects more than one fracture.

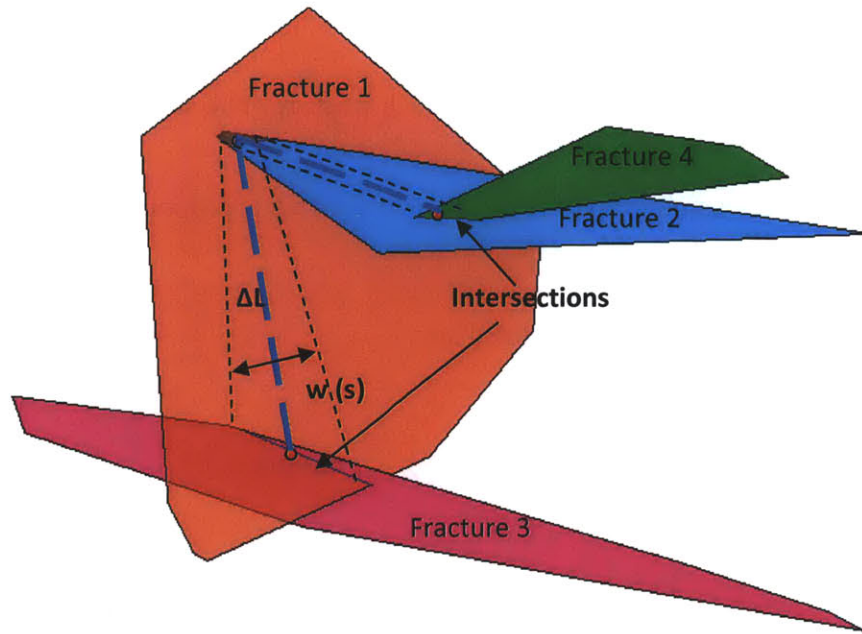


Figure 3.7 - Score components between two fracture intersections

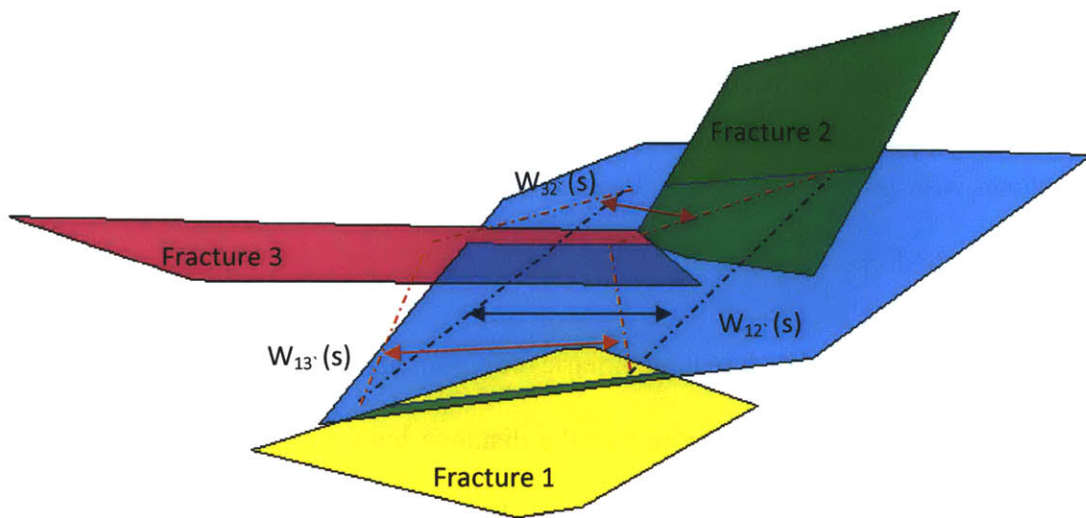


Figure 3.8 - Score components: fracture intersects more than one fracture

Step 4 - Highest score path

The model then finds the paths that have the highest score based on the score of the fractures. The highest score path(s) is calculated using the Dijkstra's algorithm (Dijkstra, 1959).

In GEOFRAC, the Dijkstra's algorithm is used to calculate the “highest score paths” (or most likely path) between all the fractures that intersect the injection boundary and all the fractures that intersect the production boundary. Figure 3.11 shows an example of a most likely overall path, i.e. the path with the highest score in a specific reservoir. Figure 3.12 shows three different highest score paths between initial fractures and different final fractures.

Step 5 – Intersection between paths

At this point the model generates a list of several branches, each one composed of numerous fractures and nodes that are the intersection between branches or the intersection between branches and one of the boundaries, as illustrated in Figure 3.9. Branches are then joined to form paths. Each path is composed of several fractures that can be represented schematically as shown in Figure 3.10. The path can be represented by an equivalent fracture width and equivalent aperture and a length that is equal to the sum of the lengths of all fractures.

The equivalent aperture can be calculated with Equation 3.4.

$$h_{eq} = \frac{1}{\sqrt[3]{\sum_{i=1}^n \frac{l_i}{l} \left(\frac{1}{h_i^3} \right)}} \quad \text{Equation 3.4}$$

where, l_i and h_i are the length and aperture of the i^{th} fracture ;

l is the total length of the series of fractures, i.e. the sum of all the fractures in the series.

The equivalent length of a path is the sum of the lengths of all fractures that constitute the path (Equation 3.5)

$$l_{eq} = \sum_{i=1}^n l_i \quad \text{Equation 3.5}$$

Where, l_i is the length of the i^{th} fracture

The equivalent width can be computed by a weighted average of all the fractures that are part of the path. This is represented by Equation 3.6.

$$w_{eq} = \frac{\sum w_i l_i}{\sum l_i} \quad \text{Equation 3.6}$$

Where

w_i is the width of the i^{th} fracture

l_i is the length of the i^{th} fracture

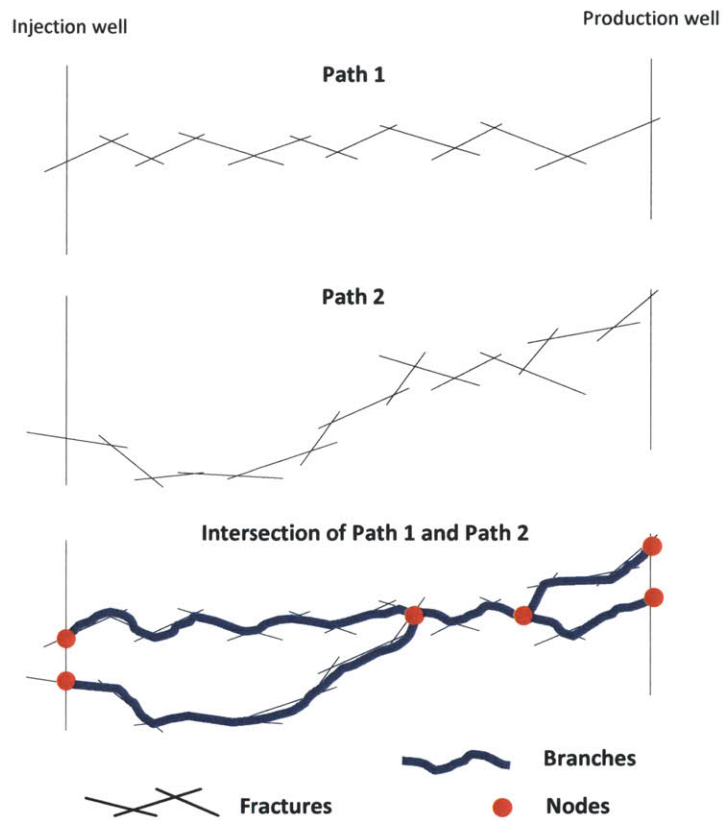


Figure 3.9 - Intersection of two paths; representation of branches and nodes.

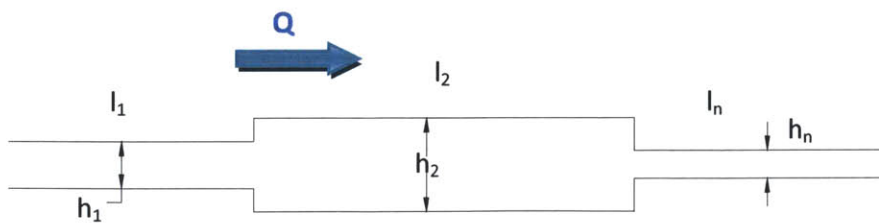
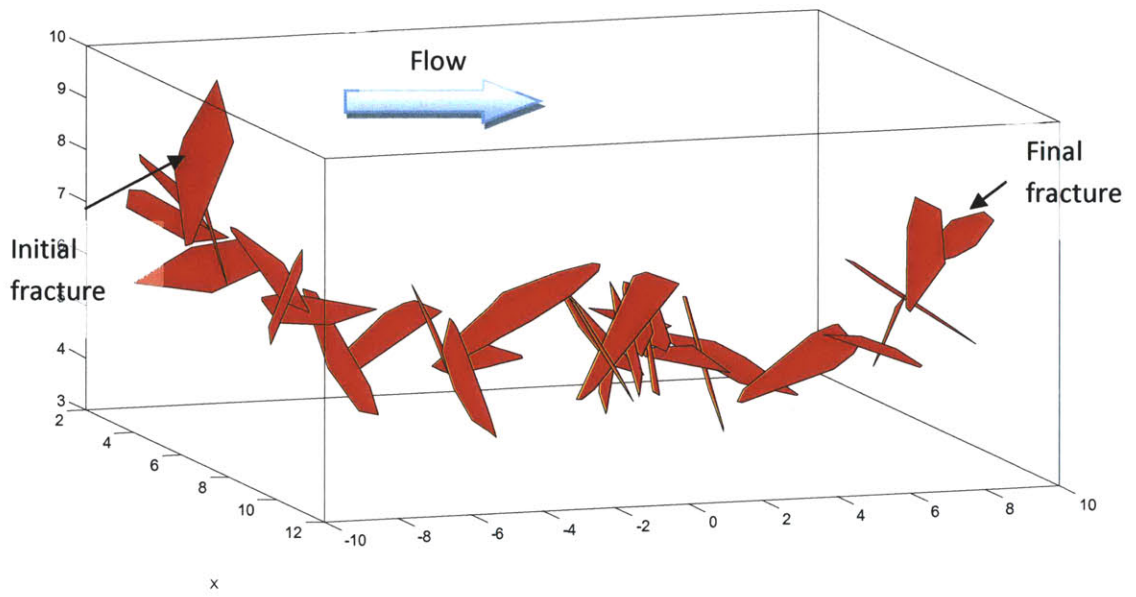
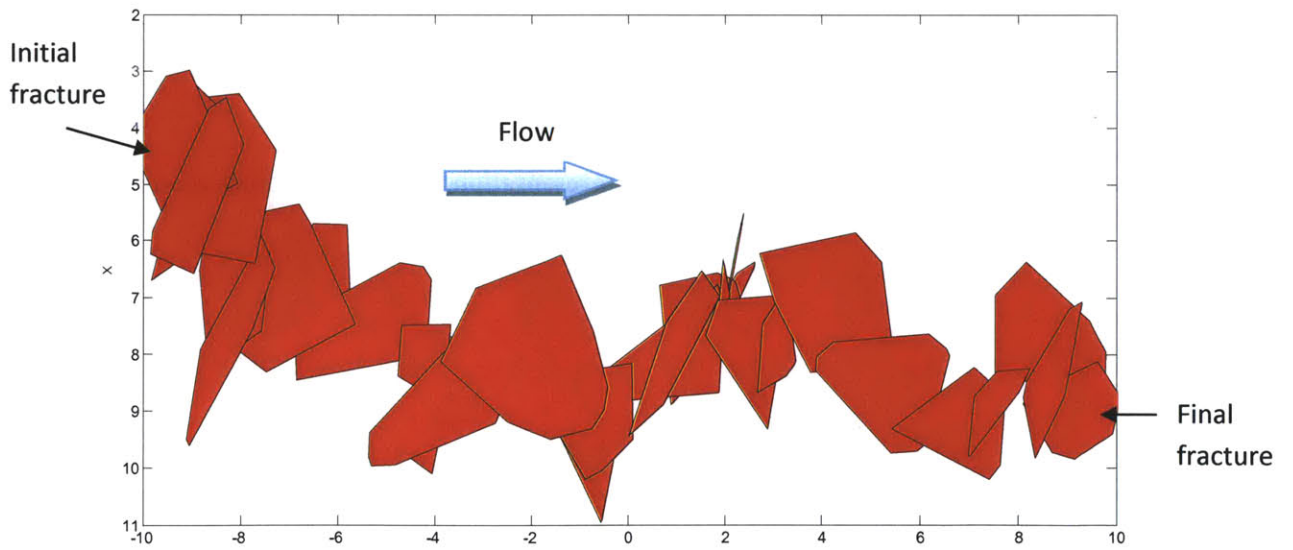


Figure 3.10 - Series of fractures modeled as parallel plates

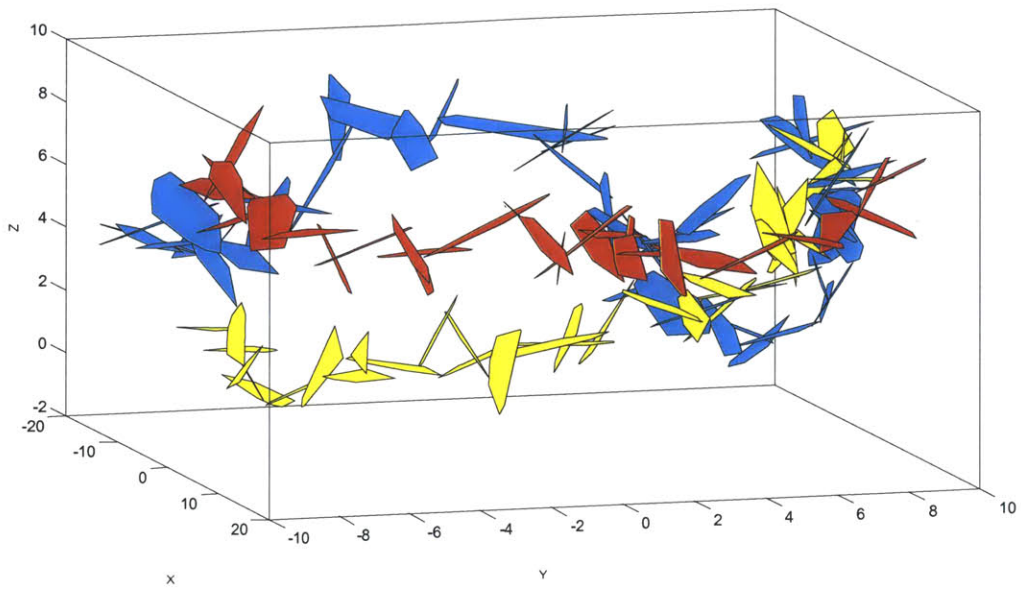


a) 3D view

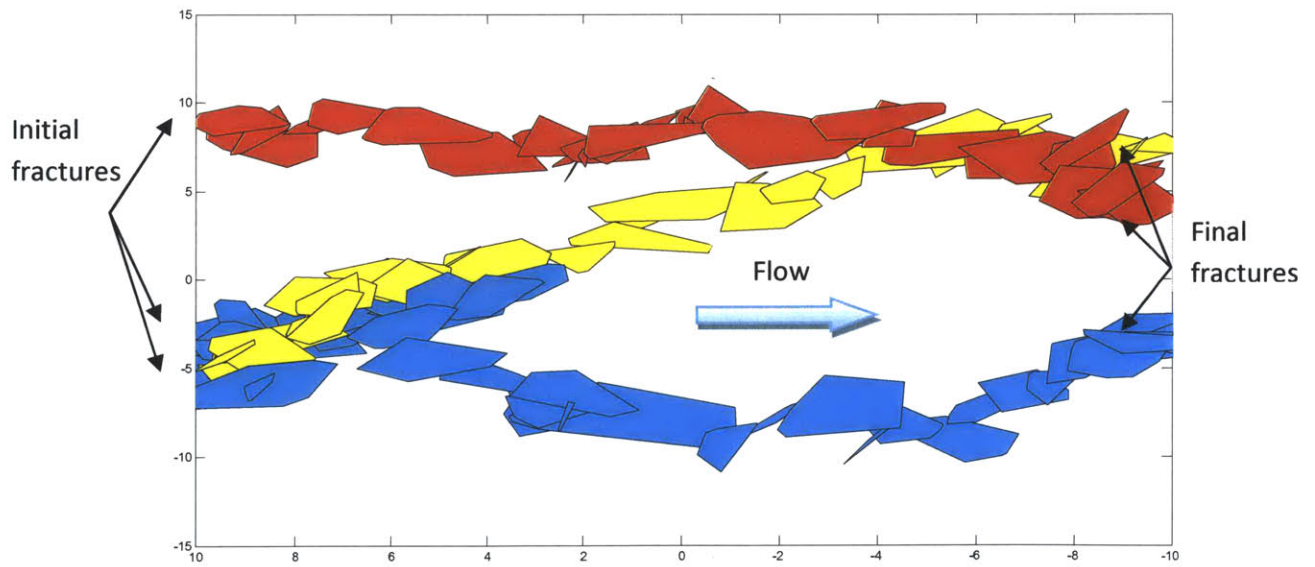


b) X-Y view

Figure 3.11 - Highest score path between two fractures that intersect the two boundaries



a) 3D view



B) X-Y view

Figure 3.12 - Highest score path between different pairs of fractures

After finding the best geometric solution (fracture system) to create the system of the branches the model calculates the output flow (equation 3.2) as the sum of the output flow from each path.

CHAPTER 4

PARAMETRIC STUDY

4.1 Introduction

The parametric analysis was conducted in order to check the consistency of the model and determine which parameters have the greatest effects on the final results.

The parametric analysis considers simplified conditions:

- A synthetic 20x20x10 m volume
- Injection and production wells are the left and right boundaries of the volume
- The water temperature is assumed to be 20°C, i.e. the dynamic viscosity is 1.002×10^{-3} Pa s.

These simplifications are both justified and necessary since the parametric study intends to verify the intrinsic correctness of GEOFRAC flow model.

The results of this analysis are presented in the next paragraphs and are sub-divided into the following sections:

- output analysis when varying the aperture parameter;

- output analysis when varying the Fisher parameter that affects the orientation of the planes during the primary process;
- output analysis when varying the rotation the fractures during the tertiary process.

Each section will conclude with a brief summary of the results obtained.

The following code will be used to identify the simulations done using the follow abbreviated indicators, in the charts as well as in some comments:

$$E[A] - P_{32} - \kappa - R - h$$

Where:

$E[A]$ = Expected area of the fracture (m^2)

P_{32} = Fracture intensity (fracture area/volume)

κ = Fisher parameter

$R=0$ 'no rotation' of the fractures in the tertiary process

$R=1$ 'random rotation' of the fractures in the tertiary process

h = Aperture of the fracture (m)

For example: the code 2-3-1-0-0.005 means: .

$E[A]=2 m^2$, $P_{32}=3$, $\kappa=1$, $R=0$ (no rotation), aperture= $0.005 m$

4.2 Parametric study results

4.2.1 Output analysis when varying the aperture parameter

GEOFRAC allows the user to generate the aperture (h) of the fractures using three models: a deterministic approach, in which the aperture is a function of the radius of the sphere that circumscribes the fracture (polygon); a probabilistic approach, which follows a truncated lognormal distribution, and a fixed value approach in which the users can fix the value of the aperture. This last approach was used for this analysis, in order to establish the sensitivity of the results and to confirm the direct relation between the aperture of the fractures and Q_{out} (m^3/s). Two cases are analyzed: $h = 0.005$ m and $h = 0.01$ m. Most of the other parameters are kept fixed in this particular sensitivity analysis:

$$E[A] = 2$$

$$P_{32} = 3$$

$$\kappa = 1$$

$$R = 1 \text{ (rotation) or } 0 \text{ (no rotation)}$$

For both cases shown in Figure 4.1 ($h = 0.005$ m) and Figure 4.2 ($h = 0.01$ m), the results for no rotation and random rotation of the polygons are reported in the same chart. Just 40 simulations were run; a number considered representative since the aim of this analysis is to study the trend of the results and not to evaluate the exact

value of Q_{out} . A detailed study of sample analysis is presented in Vecchiarelli and Li (2013).

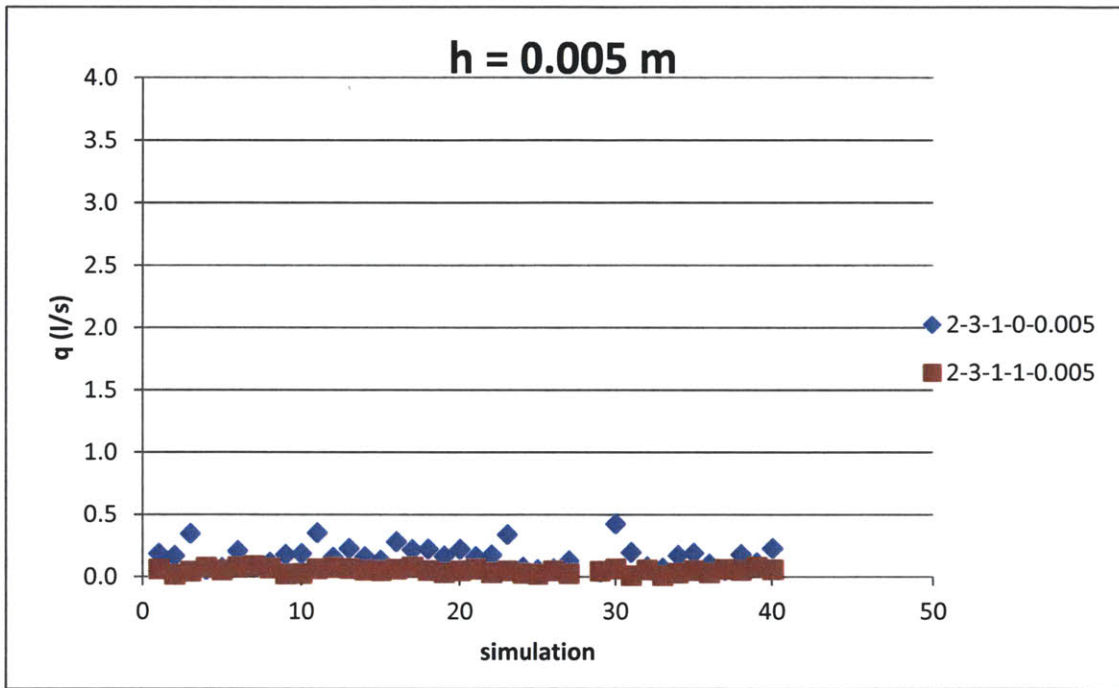


Figure 4.1 - Flow rate for $h=0.005$ m for no rotation and random rotation of the fractures

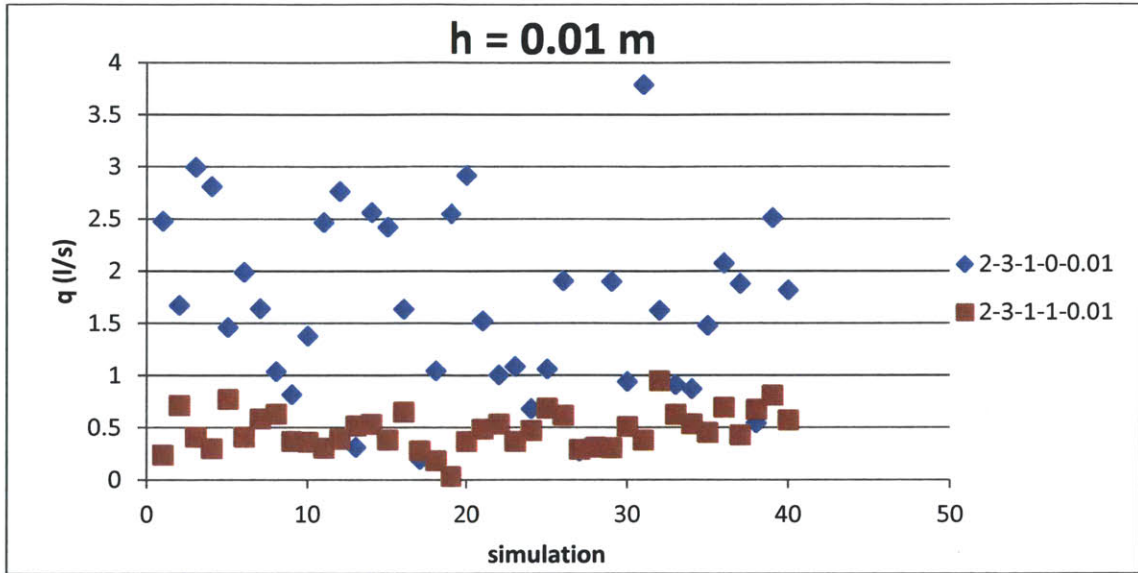


Figure 4.2 -Flow rate for h=0.01 m for no rotation and random rotation of the fractures

In Tables 4.1 and 4.2 the mean, the standard deviation and the coefficient of variation of the simulations are summarized.

Table 4.1- Values of Qout (m³/s) for h = 0.005 m

	Mean Qout (m ³ /s)	Standard Deviation Qout (m ³ /s)	Coefficient of variation Qout (m ³ /s)
no rotation	0.16	0.09	0.54
rotation	0.05	0.02	0.40

Table 4.2 - Values of Qout (m³/s) for h = 0.01 m

	Mean Qout (m ³ /s)	Standard Deviation Qout (m ³ /s)	Coefficient of variation Qout (m ³ /s)
no rotation	1.64	0.85	0.52
rotation	0.48	0.19	0.38

The results correspond well to the theory. The ratio of the cubic values of the aperture parameters chosen for this analysis is $0.01^3/0.005^3=8$. Qout for the rotation case for example is Qout= 5.16 m³/s for h=0.005 m and Qout=47.94 m³/s for h=0.01 m and producing a ratio of about 9 which is very close to 8. A similar ratio is obtained for the no rotation case. (The differences in absolute values for rotation and no rotation will be discussed later). The variability as expressed by the coefficient of variation is apparently not affected by the absolute value of h.

4.2.2 Output analysis when varying the Fisher parameter

In order to check the variability of the results of Qout, using different Fisher parameters, two values were selected for the analysis presented in this section: $\kappa =1$ and $\kappa=40$. These two values represent the extreme conditions with $\kappa=1$ representing

randomly generated planes (Figure 4.3) and $\kappa=40$ mostly parallel planes (Figure 4.4). The resulting Qout are plotted in Figures 4.5 and 4.6.

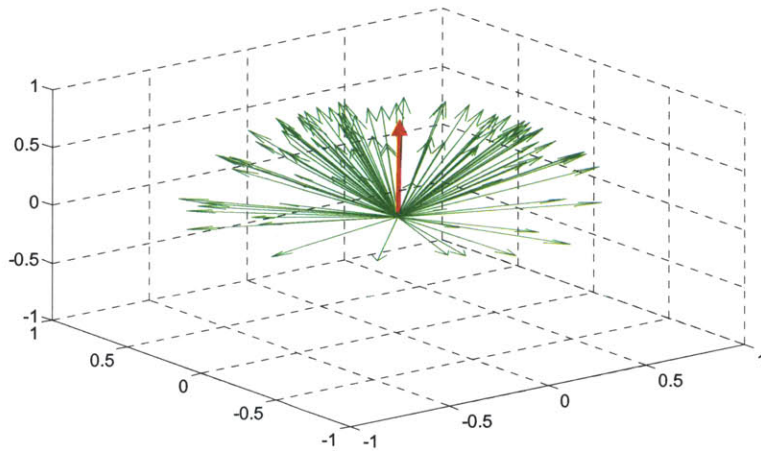


Figure 4.3 - Fracture set poles. Orientation distribution: Univariate Fisher $\kappa=1$

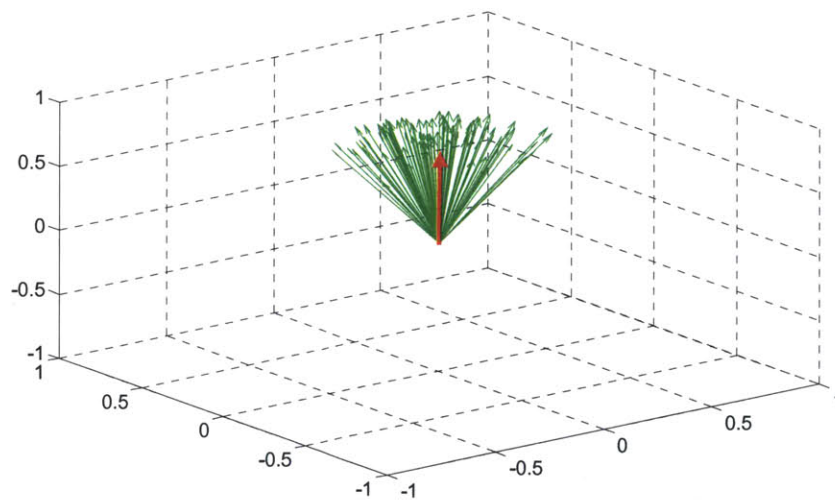


Figure 4.4 - Fracture set poles. Orientation distribution: Univariate Fisher $\kappa=40$

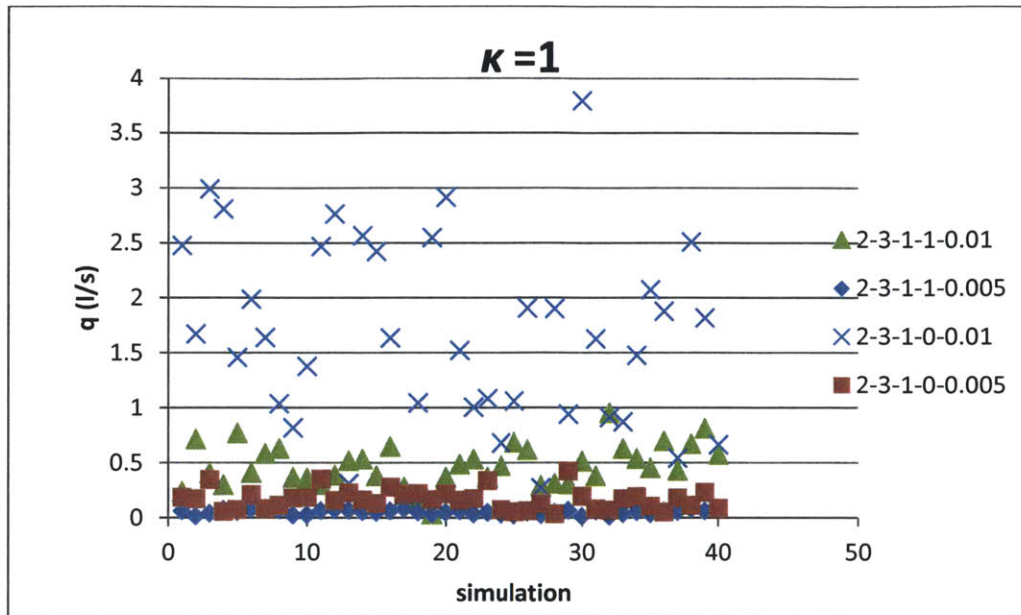


Figure 4.5 – Flow rate (Qout) values for $\kappa = 1$

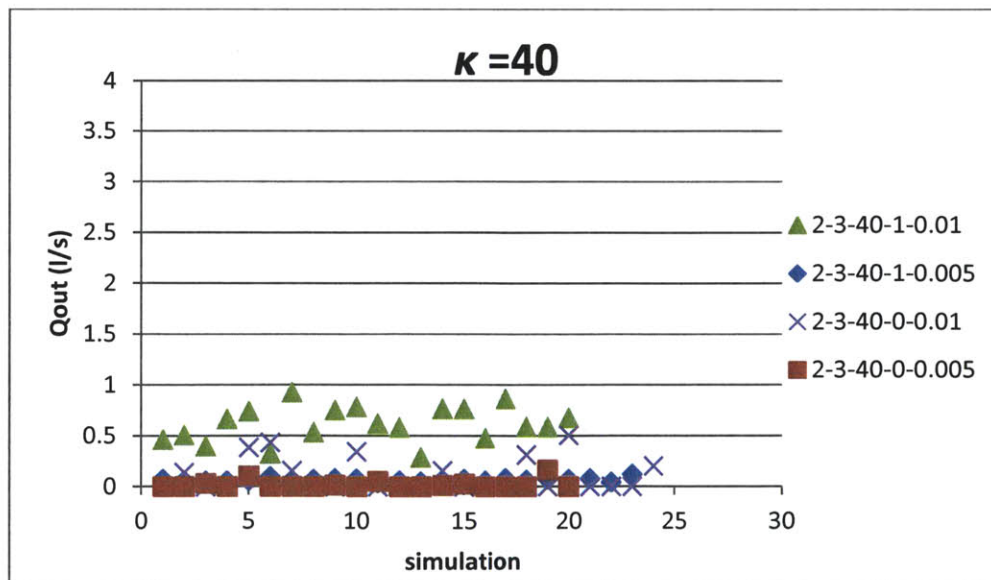


Figure 4.6 - Flow rate (Qout) for $\kappa = 40$

For $\kappa = 1$ (Figure 4.5) the largest values are obtained for $h=0.01$ m and no rotation. For $\kappa = 40$ this is different (Figure 4.6). In fact, for $\kappa = 40$ and $h=0.01$ the highest values of Q_{out} occur for random rotation. A possible explanation is as follows: the rotation of the fractures starting from almost parallel planes ($\kappa=40$) adds some randomness that increases the intersections between fractures generating more flow. On the other hand starting from random orientation of the planes ($\kappa=1$) the fractures intersect because of the orientations of the planes with no rotation. The rotation appears to remove some of these intersections. In order to better understand this behavior the number of paths and their physical location in the control volume are shown in the following figures (Figures 4.7 to 4.14) in which the variation of all parameters is investigated ($\kappa=1, \kappa = 40$; $h=0.01$ m, $h=0.005$ m; rotation, no rotation).

- Plots of the paths for $\kappa=1$

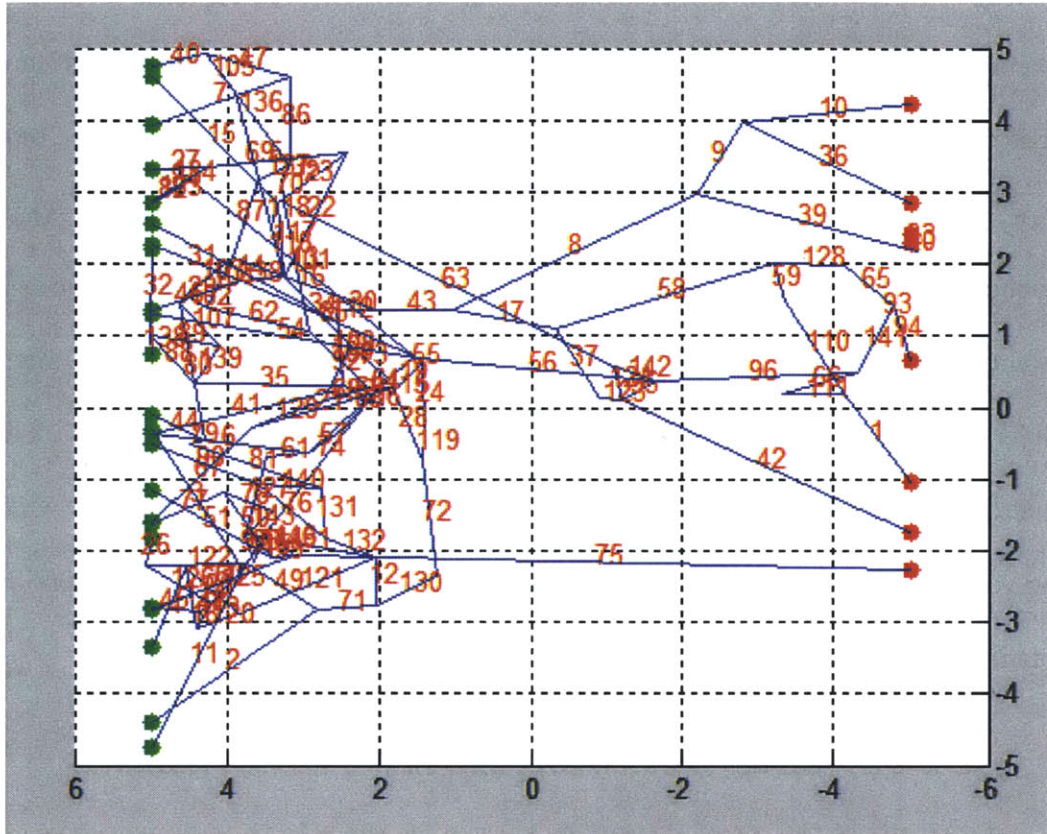


Figure 4.7 – Fracture paths system for simulation with $\kappa=1$, $h=0.005$ rotation

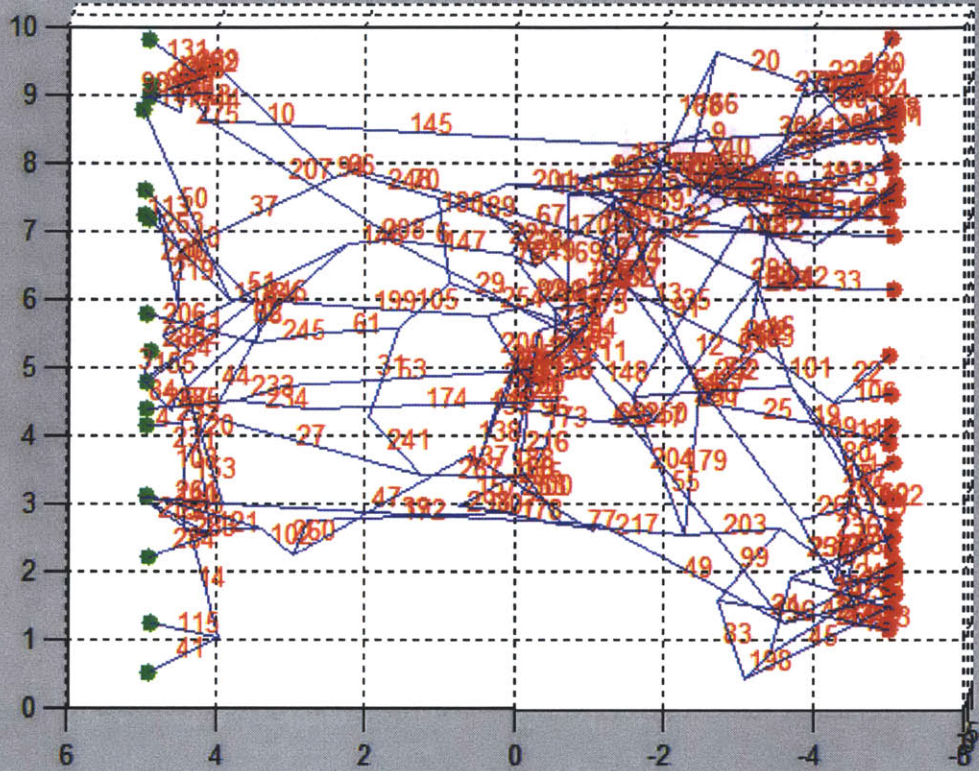


Figure 4.8 - Fracture paths system for simulation with $\kappa=1$, $h=0.01$ rotation

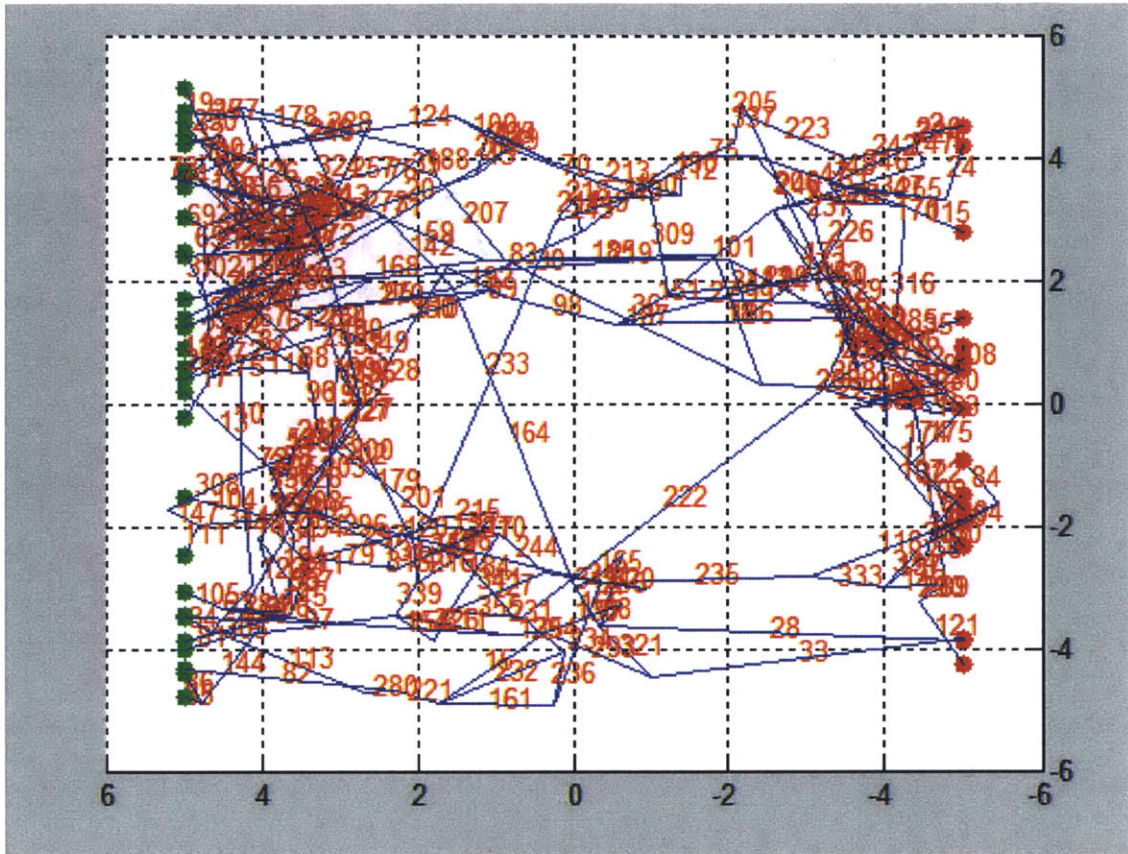


Figure 4.9 - Fracture paths system for simulation with $\kappa=1$, $h=0.005$ no rotation

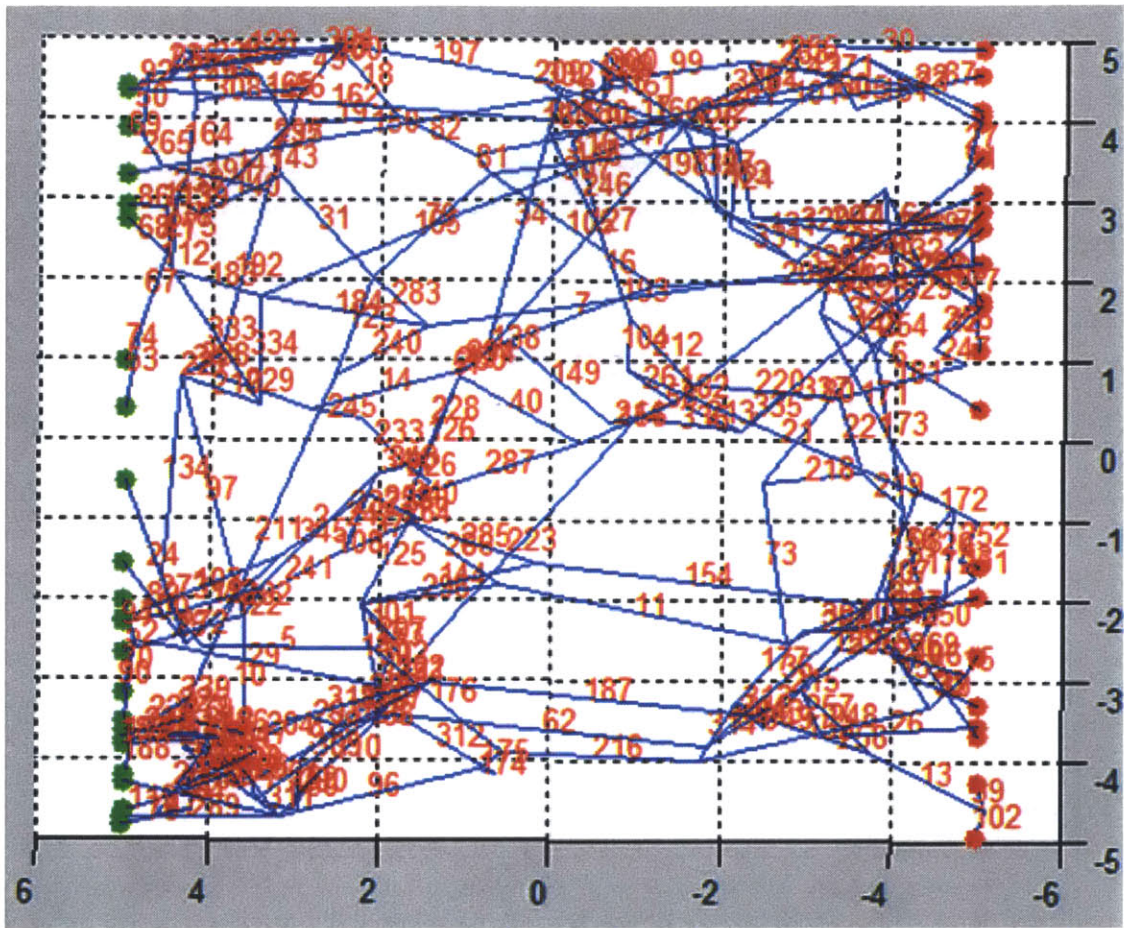


Figure 4.10 - Fracture paths system for simulation with $\kappa=1$, $h=0.01$ no rotation

- Plots of the paths $\kappa=40$

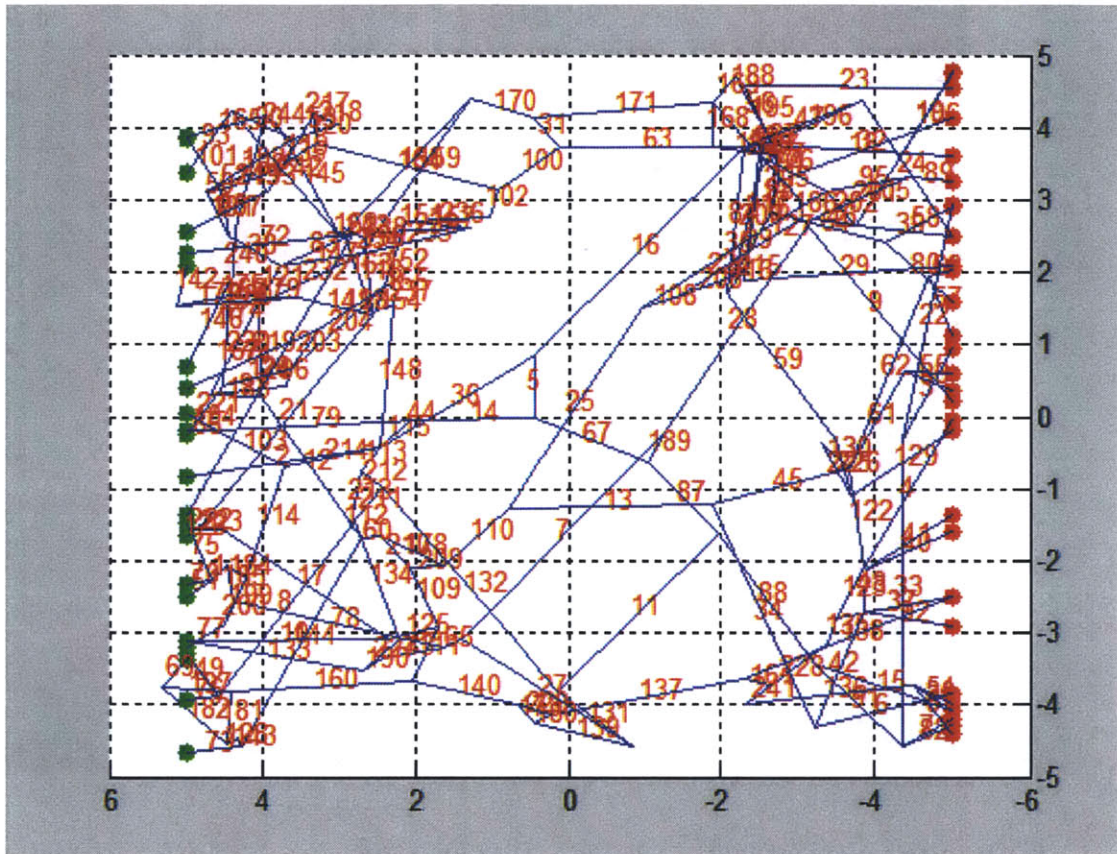


Figure 4.11- Fracture paths system for simulation with $\kappa=40$, $h=0.005$ rotation

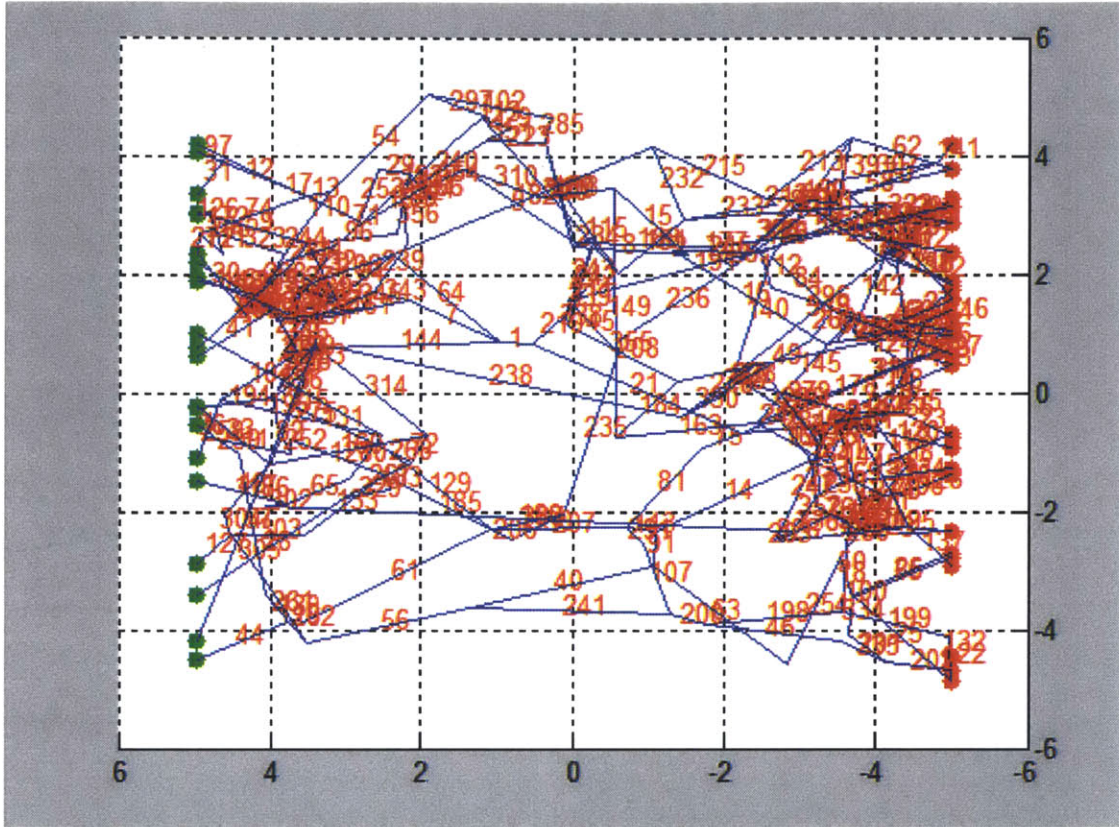


Figure 4.12 - Fracture paths system for simulation with $\kappa = 40$, $h = 0.01$ rotation

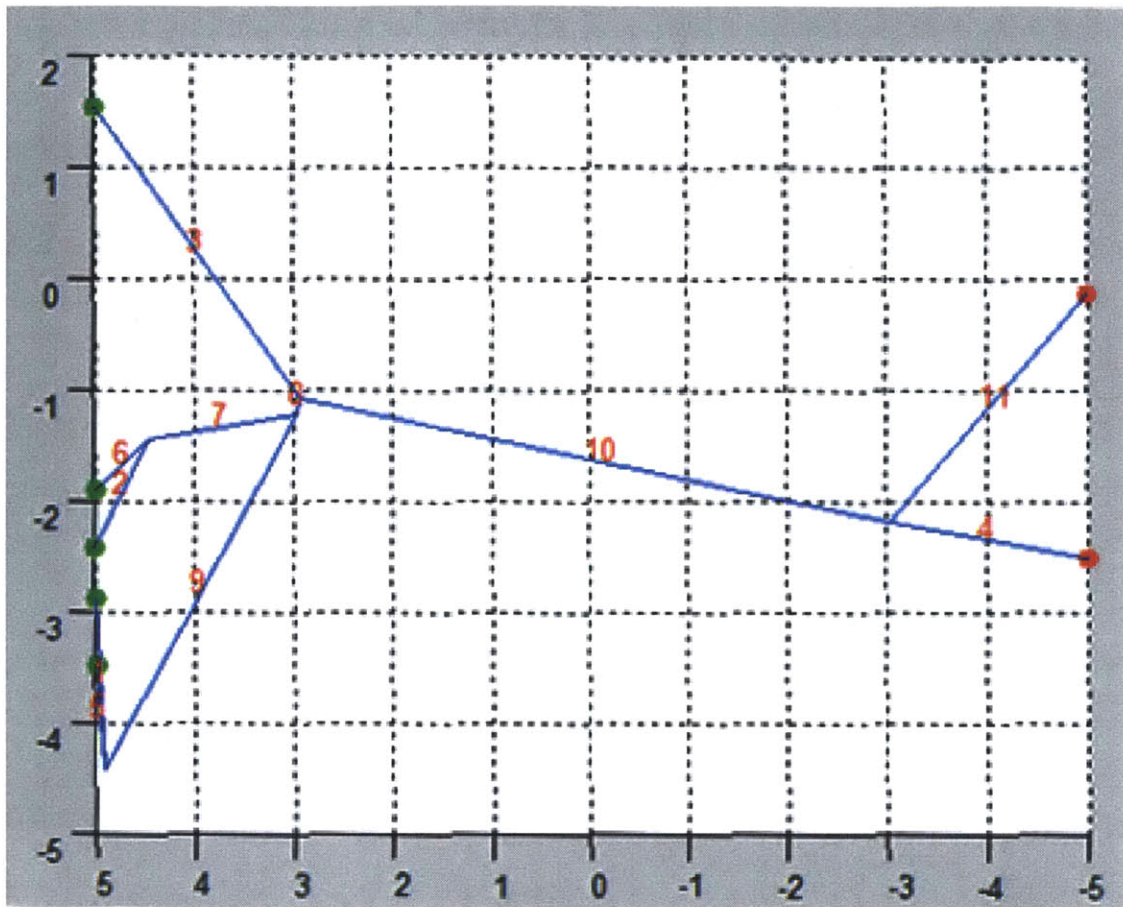


Figure 4.13- Fracture paths system for simulation with $\kappa=40$, $h=0.005$ no rotation

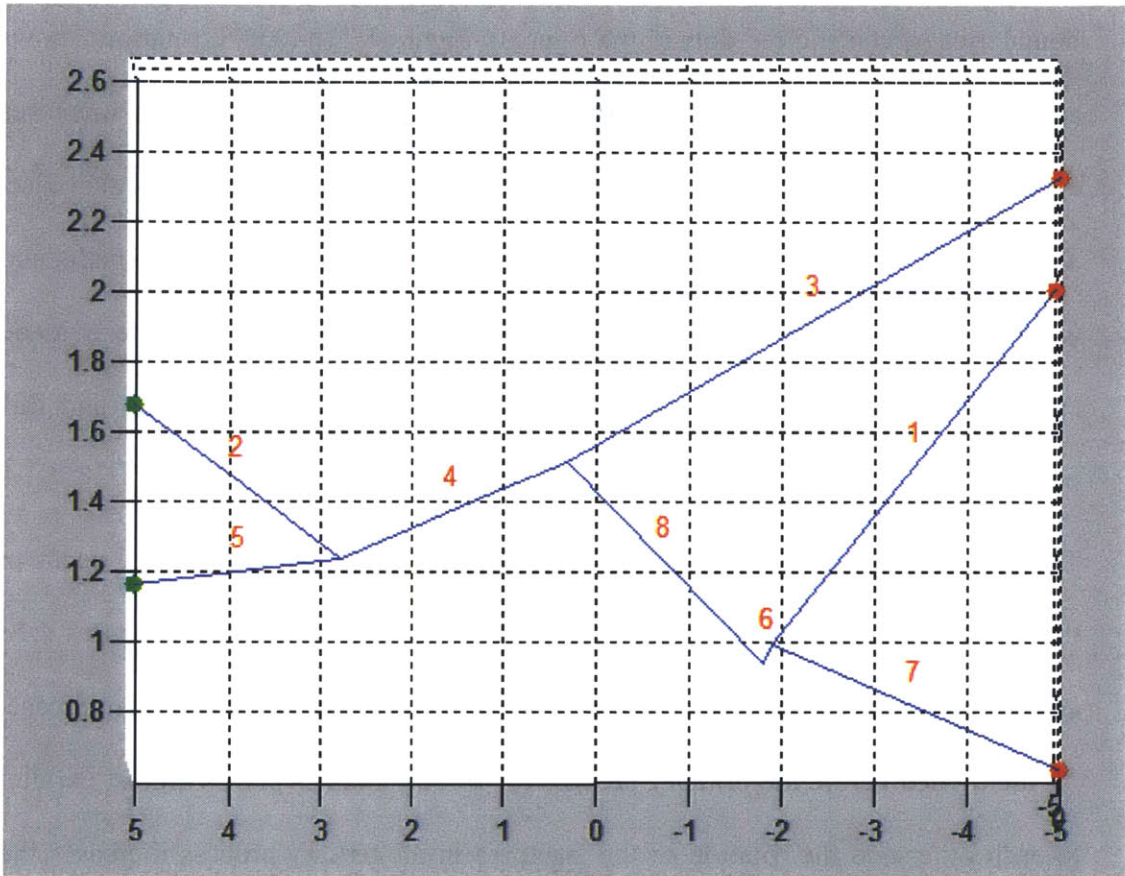


Figure 4.14 - Fracture paths system for simulation with $\kappa=40$, $h=0.01$ no rotation

The following interpretation can be offered: for $\kappa = 1$ (Figures 4.7 to 4.10) the number of paths in general is high. There are many fractures that intersect the two boundaries (green and red dots in the figures). Figure 4.7 ($h=0.005$, rotation) shows a somewhat odd behavior in that the number of paths decreases then increases again. This influences the flow, which in effect is smaller in this particular case. The $h=0.01$ m no rotation case (Figure 4.10) shows a higher number of paths and more branches compared to the other figures. This confirms the hypothesis made earlier that, with $\kappa=1$ the planes in the volume are randomly oriented, and this produces a large number of intersections between fractures.

The results for $\kappa =40$ paths are very consistent and clear. More branches occur in the rotation case (Figures 4.11 and 4.12) than in the no rotation case, (Figures 4.13 and 4.14) and as a consequence the number of paths is higher in the rotation case. As mentioned before the primary process for $\kappa=40$ generates planes almost parallel to each other and the rotation of the fractures in the tertiary process increases the probability of intersections.

4.2.3 Effect of fracture translation

In the tertiary process fractures are translated and can be rotated or not rotated. The effect of rotation was discussed above. It is worthwhile to investigate what effect translation may have. Figure 4.15 shows the possible overlaps of apertures if the fractures are translated and not rotated. Such an overlap can produce a fracture

path. However, when we investigated Q_{out} as a function of translation and rotation as shown in Figure 4.16 one can see that translation has no effect both in the rotation- and the no rotation case and that the Q_{out} for no rotation is higher (for reasons explained earlier). A possible explanation for the lacking effect of translation can be found with the numbers shown in Table 4.3. The minimum value of the translation is 0.016 m; so only few fractures with aperture $h=0.01$ m will overlap as shown in Figure 4.15.

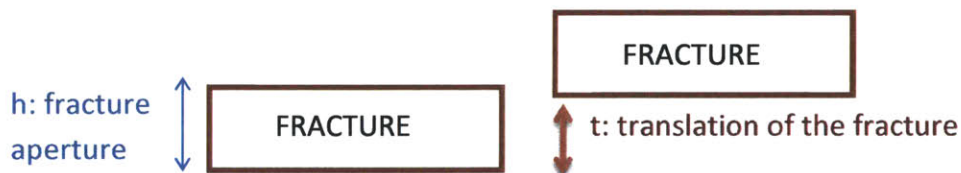


Figure 4.15 – Schematic representation of the translation between fractures

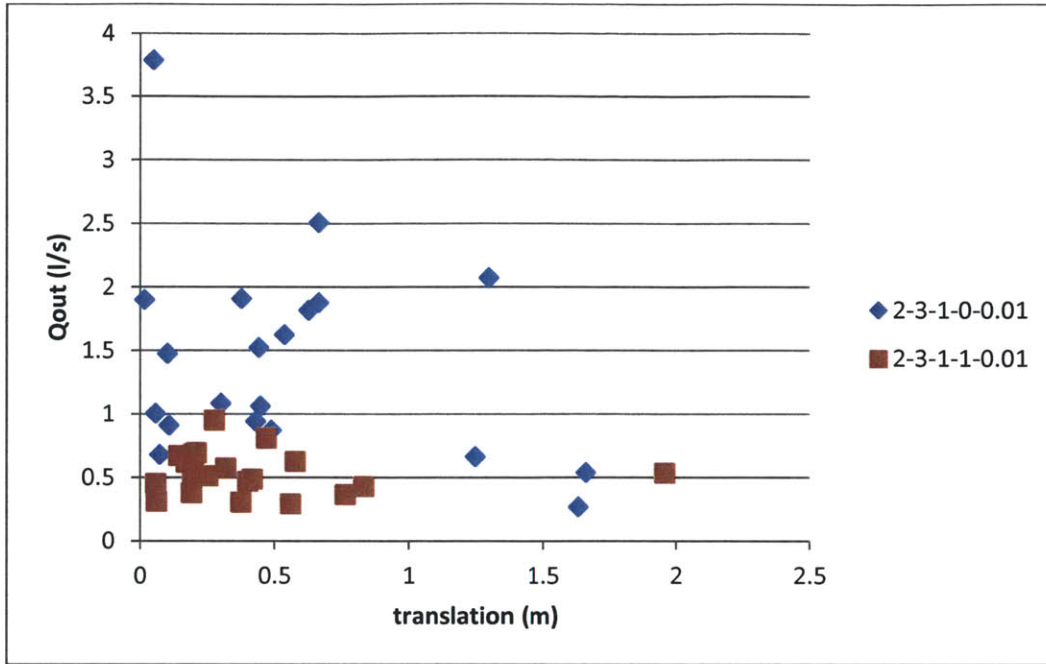


Figure 4.16 - Translation of the fractures vs Qout

Table 4.3 - Max and min values of the translation of the fractures

translation (m)	min	max
no rotation	0.016	1.659
rdm rotation	0.057	1.955

4.3 Conclusion

The parametric study demonstrates how aperture, the Fisher parameter and rotation of the fractures influence the production flow rate. As to be expected greater aperture produces greater flow. The effects of orientation are more complex as the effect of fracture plane orientation (Fisher parameter) and of rotation of individual fractures interact. For planes randomly generated the case with no rotation of the fractures and an aperture of 0.01 m generates greater flow than with rotation, while for parallel planes greater flow occurs in case of rotation of the fracture and aperture equal to 0.01 m.

This study of a simple synthetic case shows that the model is consistent but also that are some unexpected complexities affected by fracture orientation.

CHAPTER 5

BOREHOLE INTERSECTION

5.1 INTRODUCTION

Up to now the fracture and flow system was created and modeled starting from plane boundaries of the controlled volume (Figure 5.1). In order to apply GEOFRAC to model a geothermal reservoir, borehole¹ boundary conditions have to be implemented in the model. With this GEOFRAC allows the user to evaluate the flow just through the fractures that intersect the injection - and production wells. After the fracture system is modeled with plane boundaries, the model checks if fractures intersect the injection and the production wells (Figure 5.1).

¹ The term borehole will be used in this Chapter to represent geothermal wells

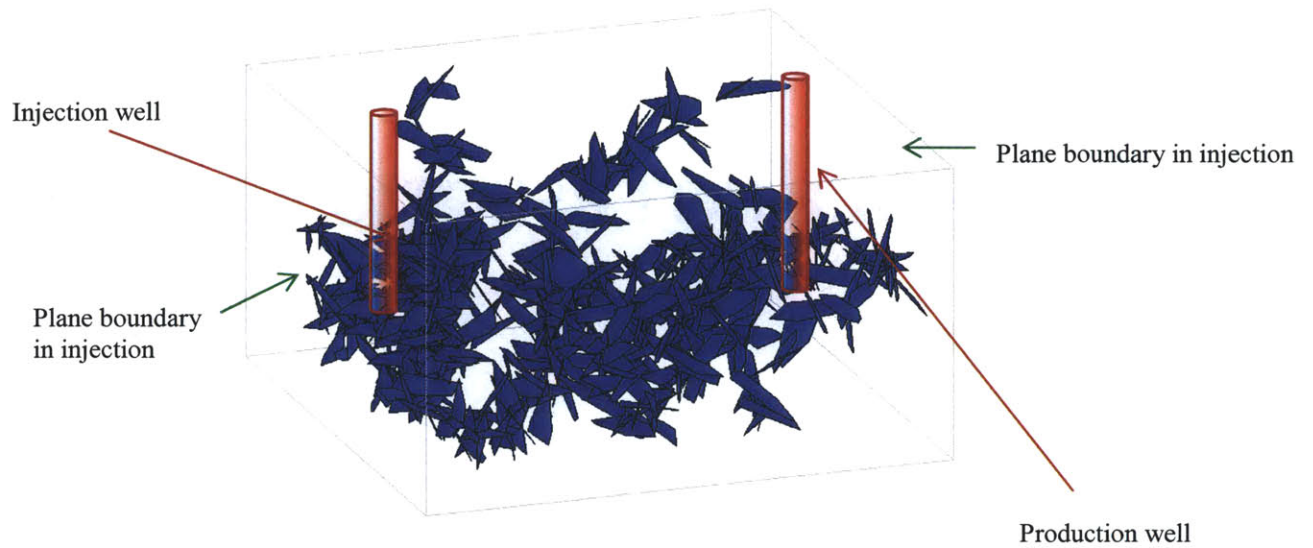


Figure 5.1 - Fracture systems intercepted by the two wells

5.2 INTERSECTION ALGORITHM

5.2.1 The MATLAB function *intersectBorehole*

In order to have the possibility to choose if the model considers the fractures system in the entire controlled volume or just the fractures that intersect the boreholes, a new function `intersectBorehole` was created.

```
function [frac, inter]=intersectBorehole(P0, d, rb, fractureSet)
```

This function takes the fracture set generated in the controlled volume, checks which fractures intersect the borehole (cylinder) and returns the intersections and the fractures that intersect the cylinder.

In more detail the inputs are:

PO: $[X_0, Y_0, Z_0]$, i.e. is the center of the borehole at the surface

d: depth of the borehole (m)

rb: radius of the borehole (m)

FRACTURESET: MATLAB structure folder containing all fractures and geometric characteristics. (Polygon vertex coordinates)

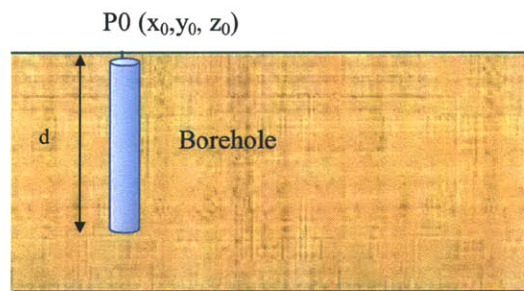


Figure 5.2 Schematic representation of geometric inputs for the Intersect Borehole function

The outputs are:

FRAC: array of fractures that intersect the borehole

INTER: array of intersections of the fractures with the borehole

From this point on GEOFRAC will take as input just the FRAC array in which fractures that intersect the borehole are stored.

In the Matlab file this intersection is done in three steps:

STEP 1: Clear polygons outside the interest volume (below the borehole zone)

From the computational point of view it is better to eliminate fractures that are below the borehole depth because one is certain that they will not intersect the borehole. This allows one to reduce the quantity of fractures that the code needs to check. The function that it is run at this point is `clearPolygonsBelowVolume`

```
[polygon, cs, rs]=clearPolygonsBelowVolume(polygon,center,P0,d);
```

Where

`polygon=fractureSet.polygonall;` this is the list of the fractures generated in the controlled volume

`center=fractureSet.Call;` it is the center of all fractures in the controlled volume

`cs, rs` are respectively the center and the radius of the spheres that enclose the polygons

See the end of this chapter for the code of the function `clearPolygonsBelowVolume`.

STEP2: Check which spheres that enclose polygons, resulting from step 1, intersect the borehole

In this step the intersections between the borehole and the spheres that enclose the polygons is checked and stored in C.

```
C = intersectBoreholeSphere(cb,cs,rb,rs);
```

See the end of this chapter for the function `intersectBoreholeSphere`

Figure 5.3 gives a visual representation in Matlab of an intersection between a cylinder and spheres.

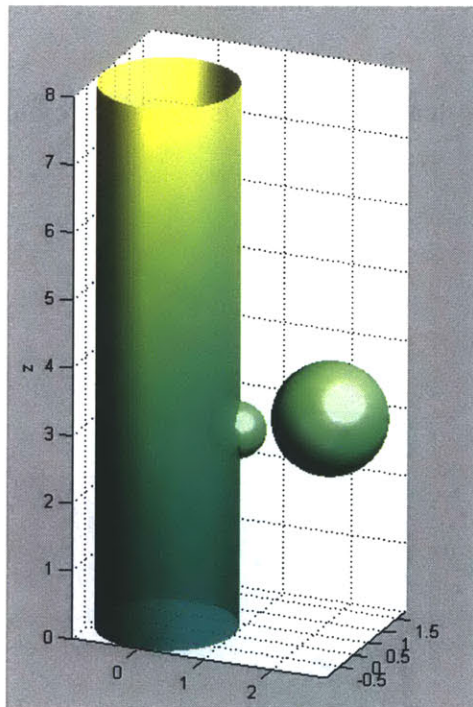


Figure 5.3- Matlab representation of intersection between a cylinder and spheres

STEP 3: Calculate if polygons (whose spheres intersect the borehole) actually intersect the well

In this step just the polygons, whose spheres intersect the borehole and that were stored in the preceding step are analyzed.

```
m=find(C==1);
```

$C==1$ means that there is intersection and this information is stored in m .

The intersection between a sphere and a cylinder can be a circle, a point, the empty set, or a special type of curve. In order to ensure any possible intersection between polygon and cylinder, the following cases are taken into account:

- CASE 1- General case If at least one vertex is inside the cylinder, the polygon intersects the borehole (Figure 5.4)

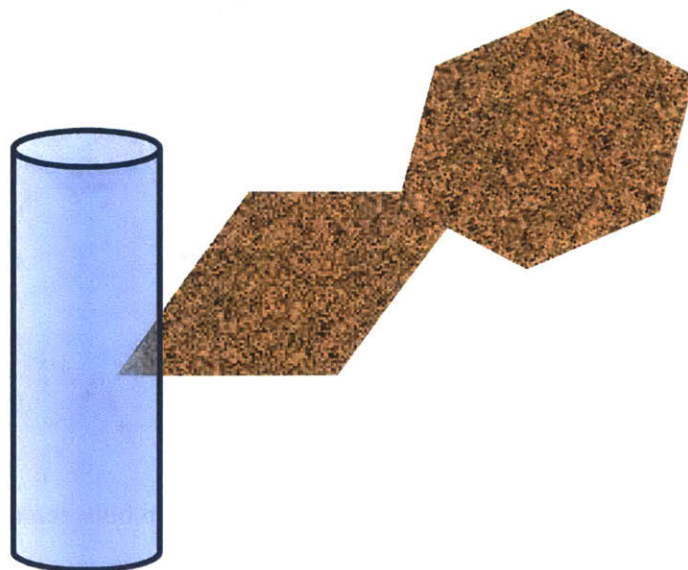


Figure 5.4 - Schematic representation of intersection between the borehole and one vertex of the fracture (polygon)

- CASE 2- Polygon inscribed into the cylinder. The way of modeling case 1 solves also the case of polygon inscribed into the borehole (Figure 5.5).

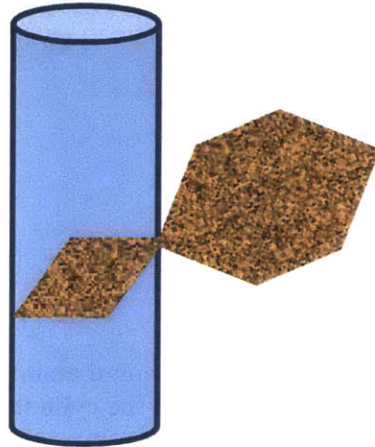


Figure 5.5 - Schematic representation of a fracture (polygon) inscribed into the borehole

- CASE 3 - All vertices outside the cylinder but a line between 2 vertices (i.g. v_1 and v_2 in Figure 5.6) intersects the cylinder.

At this point of the code just the vertexes coordinates are stored in FRACTURESET, so lines between vertices are created in the code, and then the intersections between lines and the cylinder are calculated.

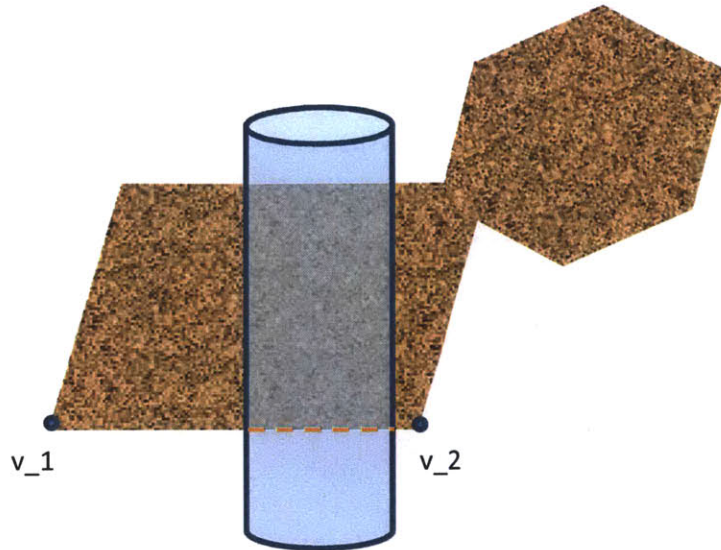


Figure 5.6- Schematic representation of an intersection between one side of a fracture and the cylinder

- CASE 4 - All vertices outside the cylinder, no line between vertices intersect the cylinder (Figure 5.7).

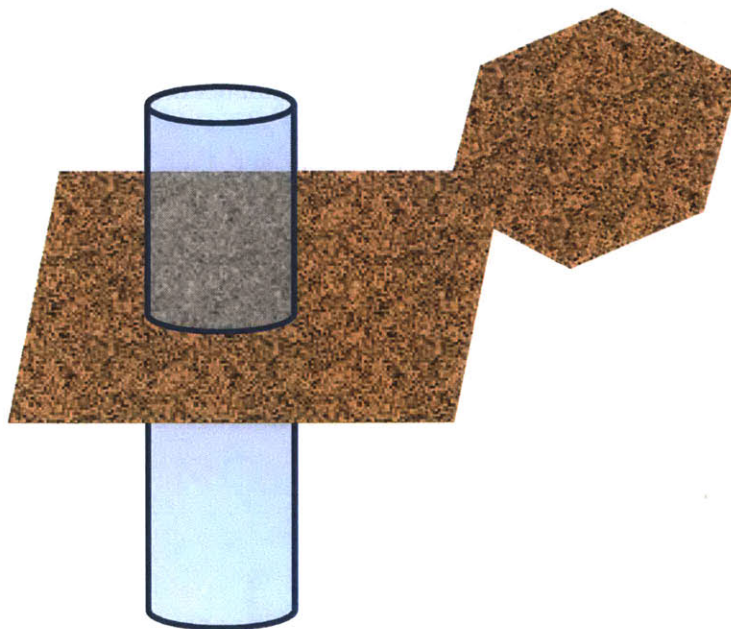


Figure 5.7- Schematic representation of intersection between a fracture (plane) and a borehole (cylinder)

MATLAB CODE – intersectBorehole

```
function [frac, inter]=intersectBorehole(P0, d,rb, fractureSet)

%INTERSECTBOREHOLE   Return intersection between fracture and borehole
%
%   [FRAC, INTER]=intersectBorehole(P0,D,RC, FRACTURESET)
%   Takes a fracture set, checks which fractures intersect the borehole
%   (cylinder) returns the intersections and the fractures that
intersect.
%   PO: [X0,Y0,Z0]
%   D:  depth of the borehole
%   RB: radius of the borehole
%   FRACTURESET: structure containing all fractures and geometric
%   characteristics
%   FRAC: array of fractures that intersect the borehole
%   INTER: array of intersections of the fractures with the borehole

%STEP 1: Clear polygons outside the interest volume

center=fractureSet.Call;
polygon=fractureSet.polygonall;

[polygon, cs, rs]=clearPolygonsBelowVolume(polygon,center,P0,d);
```

```

%STEP2: check which spheres that enclose polygons resulting from step 1
% intersect the borehole

C = intersectBoreholeSphere(P0,cs,rb,rs);

% STEP 3: Calculate if polygons (which spheres intersect the borehole)
% actually intersect the borehole

m=find(C==1);

%calculate P1: end point of the borehole, i.e. P0(z)-depth
P1(1:2)=P0(1:2);
P1(3)=P0(3)-d;

L = createLine3d(P0, P1);
% P0 = X0,Y0,Z0
% P1 = X0,Y0,Z0+depth;
%assign inter
inter=zeros(m,3);

if ~isempty(m)
for i=1:length(m)
    [inter(i,:)]= intersectLinePolygon3d(L, polygon{m(i)});
end

n=find(~isnan(inter(:,1)));
frac=m(n);
inter(isnan(inter(:,1)),:)=[];

```

```
else
    frac=[];
    inter=[];
end

end
```

MATLAB CODE - clearPolygonsBelowVolume

```
function [polygon, center, rs] = clearPolygonsBelowVolume (polygon,  
center, P0, depth)
```

This function returns the polygons, their center and radius, which are above a calculated depth.

Step 1- Determine the maximum Radius of all the spheres that enclose the polygons

```
ds=getDmax(polygon, center);  
Dmax=max(ds);  
rs=ds/2;  
rs=rs';
```

Step 2- A limit in depth is fixed in order to eliminate part of the volume in which sphere not intersect the borehole. All the spheres below this depth are not considerate.

```
lim=P0(3)-depth-Dmax;  
m=find(center(:,3)<lim);  
center(m,:)=[];  
polygon(m)=[];  
rs(m)=[];  
  
end
```

MATLAB CODE - intersectBoreholeSphere

```
function C = intersectBoreholeSphere(cb,cs,rb,rs)

% INTERSECTBOREHOLESPPHERE True if the borehole and spheres intersect
% C=INTERSECTBOREHOLESPPHERE(CB,CS,RS) returns a matrix containing 1
%   if
%   boreholes defined in array 1 and spheres defined in array 2
%   intersect and 0 otherwise.
%   CB is an array that contains the centers of spheres in array 1
%   CS is an array that contains the centers of spheres in array 2
%   RB is an array that contains the radius of spheres in array 1
%   RS is an array that contains the radius of spheres in array 2

%computes the distances between vectors in cB and cS
D=dist(cb(1:2)',cs(:,1:2)');

%computes the sums of the radius of rB and rS
sumR=(ones(length(rs),1)*rb')'+(ones(length(rb),1)*rs');

C = (D<sumR);

end
```


CHAPTER 6

GEOFRAC GRAPHICAL USER

INTERFACE

6.1 INTRODUCTION

Geofrac is a program written in the MATLAB computational environment, so one needs to run MATLAB in order to use GEOFRAC. GEOFRAC has been made accessible through the Graphical User Interface (GUI). The GUI allows the user to perform the complicated simulations of a geothermal basin without the need to understand the details of how the tasks are performed.

After starting MATLAB, one can use the GUI to enter input parameters, as described in more detail in section 6.2, and determine the computed fracture system and the output flow rate. The outputs that can be obtained and how they are represented are explained in section 6.3.

6.2 INPUT PARAMETERS

In the GEOFRAC folder a file called *RunGeofrac.m* opens the window shown in Figure

6.1. The GUI is organized in sections:

- Geometric inputs
- Stochastic inputs
- Simulation
- Flow input

The screenshot shows the GEOFRAC GUI with the following sections and parameters:

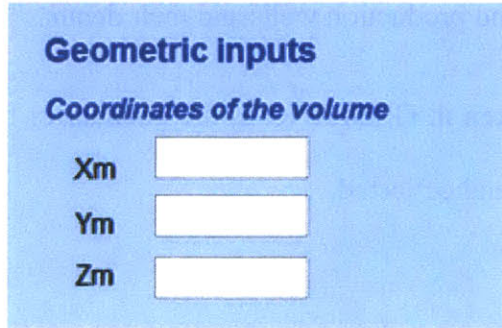
- Geometric inputs**
 - Coordinates of the volume*
 - Xm: 5
 - Ym: 5
 - Ztop: 10
- Simulation**
 - Number of simulation: 1
 - flow (dropdown menu)
- Flow input**
 - Pi: 4000 kPa
 - Pf: -1 kPa
 - Qout: 300 l/s
- Stochastic inputs**
 - P32: 2
 - E(A): 3
 - Rot: 0
 - Type of orientation distribution*
 - Bivariate Fisher: K1: 0.2, K2: 0.15
 - Uniform Max Phi: Max Phi: 3
 - Uniform
 - Uniform Fisher: K: 20
 - Aperture of the fractures*
 - Deterministic
 - Probabilistic: Alpha: 0.01, Beta: 1, Mode: 0.1, h max: 0.15, h min: 0.002

A **RUN GEOFRAC** button is located at the bottom right of the interface.

Figure 6.1 – GEOFRAC GUI (user's interface)

6.2.1 Geometric inputs

The geometric inputs (Figure 6.2) defines the reservoir dimensions.



Geometric Inputs
Coordinates of the volume

Xm

Ym

Zm

Figure 6.2 - Geometric dimensions of the reservoir to be modeled

Figure 6.3 shows how the coordinates are placed in the space. The coordinates of the controlled volume in GEOFRAC are defined in the following way:

Xm: half of the length of the area of interest (m)

Ym: half of the width of the area of interest (m)

Ztop: depth of the area of interest (m)

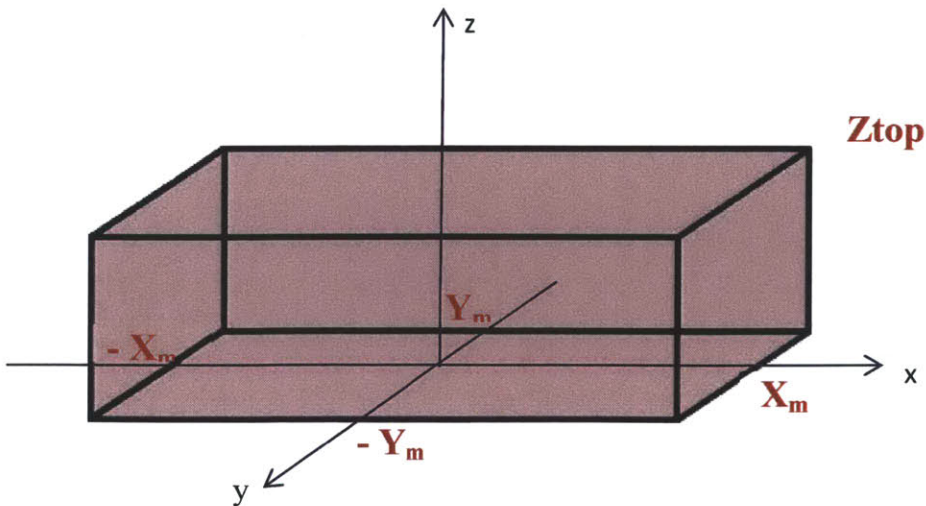


Figure 6.3 – Coordinates of the controlled volume according to GEOFRAC

At this stage of GEOFRAC coding, it is recommended not to use a large volume for the simulation in order to not encounter problems of insufficient computer memory. In fact instead of simulating the entire reservoir it is possible to concentrate the calculation in the zone defined by the injection and production wells and their depth.

The left and right sides are taken in GEOFRAC as the boundaries from and to which the flow is respectively generated and collected.

6.2.2 Stochastic inputs

GEOFRAC uses stochastic models to represent natural fracture systems based on available geological information. Figure 6.4 shows the stochastic parameters that need to be set in order to generate the fracture system. $P32$ and $E(A)$ define the intensity and the mean area of the fractures. The orientation distribution parameters define the orientation of the main planes generated in the controlled volume and the aperture model parameters define the type of model that one wants to use in order to set the aperture of the fractures. The stochastic parameters are explained in more detail below.

Once a distribution is chosen, one needs to insert just the parameter necessary to run that type of distribution.

Stochastic inputs

P32 E(A) Rot

Type of orientation distribution

Bivariate Fisher K1 K2

Uniform Max Phi Max Phi

Uniform

Uniform Fisher K

Aperture of the fractures

Deterministic Probabilistic

Alpha Mode h max

Beta h min

Figure 6.4 - Section of the stochastic inputs in the GUI.

P32, E(A) and Rot

The two most important stochastic parameters in GEOFRAC are:

- P₃₂: fracture intensity (cumulative fracture area per rock volume), that can be calculated from:

$$P_{32} = \frac{\sum_{i=1}^N A_{f,i}}{V}$$

- E(A): desired mean area of fractures
- Rot: rotation of the polygons in the tertiary process

P32, as described in Dershowitz and Herda (1992) and used in Ivanova (1995), is the most appropriate description of the fracture intensity in 3D space. P32 is not scale dependent because it incorporates the fracture size. P32 can be obtained from borehole spacing information or observations on outcrops using the approach by Dershowitz and Herda (1992). E(A) can be obtained from fracture trace lengths on outcrops with suitable bias corrections as developed by Zhang et al. (2002).

Rot is a parameter that allows the user to rotate the polygons (fractures) or not rotate them.

The values in Rot that can be inserted are:

Rot = 1: rotation of the polygons

Rot = 0: no rotation of the polygons

Types of orientation distribution

The orientation of the planes in the so-called primary process of GEOFRAC can be modeled choosing between 4 types of orientation distribution as explained in Ivanova, 1995, depending of the information available or the type of distributions that the user is most confident in. These four distributions and the parameters that need to be set are shown in Table 6.1.

Table 6.1- Types of orientation distributions in GEOFRAC and parameters that needs to be set

ORIENTATION DISTRIBUTION	PARAMETER
Bivariate Fisher	κ_1 , and κ_2
Uniform Max Phi	$\max \phi$
Uniform	κ
Uniform Fisher	κ

Fracture apertures models

As explained in Chapter 2, there are numerous models to simulate the apertures of fractures. In GEOFRAC the user can decide between two methods: deterministic and probabilistic method.

The deterministic method is the one proposed by Ivanova et al. (2012) and assumes that fracture aperture may be correlated with fracture length by a power-law function (Equation 6.1):

$$h = \alpha(2R_e)^\beta \quad \text{Equation 6.1}$$

where R_e is the equivalent radius of the fracture polygon, h is the aperture, and α and β are coefficients that depend on the site geology and that are measured on site or are possible to find in the literature.

The probabilistic model defined in Ivanova et al. (2012) assumes that the distribution of the fracture aperture can be represented by a truncated lognormal distribution (Equation 6.2):

$$f_{TR}(h) = \frac{f(h)}{\int_{h_{min}}^{h_{max}} f(h)d(h)}, h_{min} \leq h \leq h_{max} \quad \text{Equation 6.2}$$

where h_{min} and h_{max} are the minimum and the maximum aperture values.

The methods for the aperture model and the parameter are shown in Table 6.2.

Table 6.2 - Types of methods for the fracture aperture and parameters that needs to be set

FRACTURE APERTURE METHODS	PARAMETER
Deterministic	α and β
Probabilistic	mode, h_{min} and h_{max}

6.2.3 Simulation

In GEOFRAC, and as shown in Figure 6.5, it is possible to specify the number of simulations in order to improve the accuracy of the results. The program will produce the mean results of all the simulations.

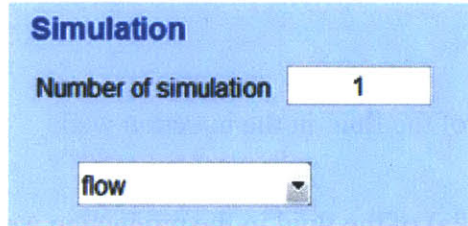


Figure 6.5 - Simulation parameter: number of simulations and type of simulation

The GUI allows one to run Geofrac obtaining the fracture pattern only (select NO FLOW), or it is possible to run Geofrac obtaining flow outputs (select FLOW). This choice allows the user to be able to use GEOFRAC just for the fracture system generation in order to use it for other applications, for example for slope stability analysis.

6.2.4 Flow inputs

If the flow simulation is chosen, some other parameters need to be fixed (Figure 6.6).

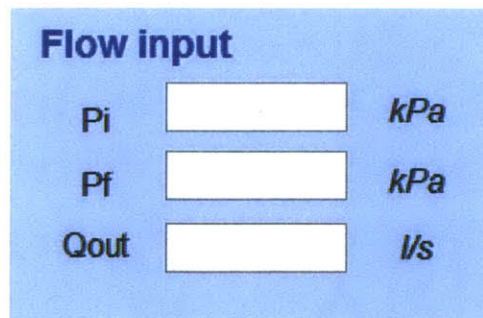


Figure 6.6 - Flow parameters in GEOFRAC

These parameters are:

P_i : injection pressure (kPa) of the fluid in the injection well

P_{out} : production pressure (kPa) of the fluid in the production well

Q_{out} : output flow rate (l/s)

The user can provide the injection pressure and/or the production pressure, or the user can set the flow rate target. Inserting the value -1 will set an unknown value, and the program will calculate it.

6.3 RESULTS

After pressing the RUN GEOFRAC button, GEOFRAC will display the fractures path as shown in Figure 6.7 and the pressure and the flow rate in the production well in the MATLAB workspace.

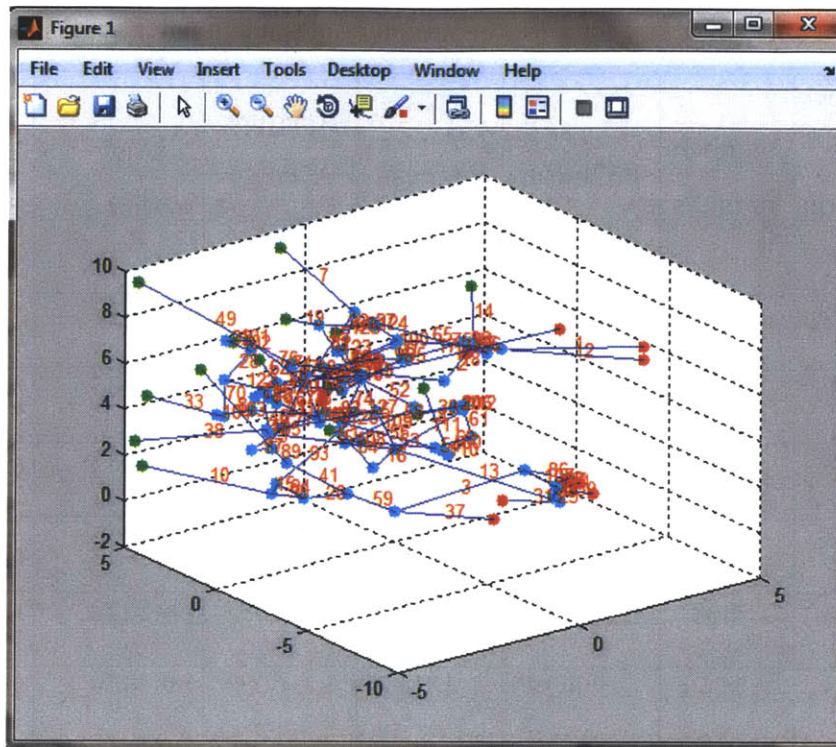


Figure 6.7 - Flow system in the controlled volume

GEOFRAC generates also an excel file called `Results` (Table 6.3) with the results of some of the parameters. The values in bold (last cells) are the mean values.

Table 6.3- Excel file with the summary of some important results

	Number s of Polygons	Numbers of paths	P32mean	EAmean	Aperture Average (m)	Qout (l/s)	Re mean (m)
1	2109	1295	3.11474	1.47688	0.005	7.4877	0.6259
2	2009	910	2.831532	1.409423	0.005	5.7135	0.6084
3	1278	400	1.828845	1.431021	0.005	5.6685	0.6137
4	1833	1216	2.633562	1.43675	0.005	5.6866	0.6180
5	2086	868	2.961028	1.419476	0.005	6.2722	0.6128
6	2093	1760	2.996395	1.431627	0.005	9.7479	0.6131
7	1659	608	2.430787	1.465212	0.005	4.6046	0.6224
8	2120	875	3.084176	1.4548	0.005	6.6706	0.6225
9	2229	1404	3.179268	1.42632	0.005	7.7514	0.6170
10	2189	1188	3.146575	1.437449	0.005	7.5592	0.6198
Av.	1784	821	2.566034	1.437124	0.005	6.2396	0.6172

The values of Qout can be plotted using excel as shown in Figure 6.8.

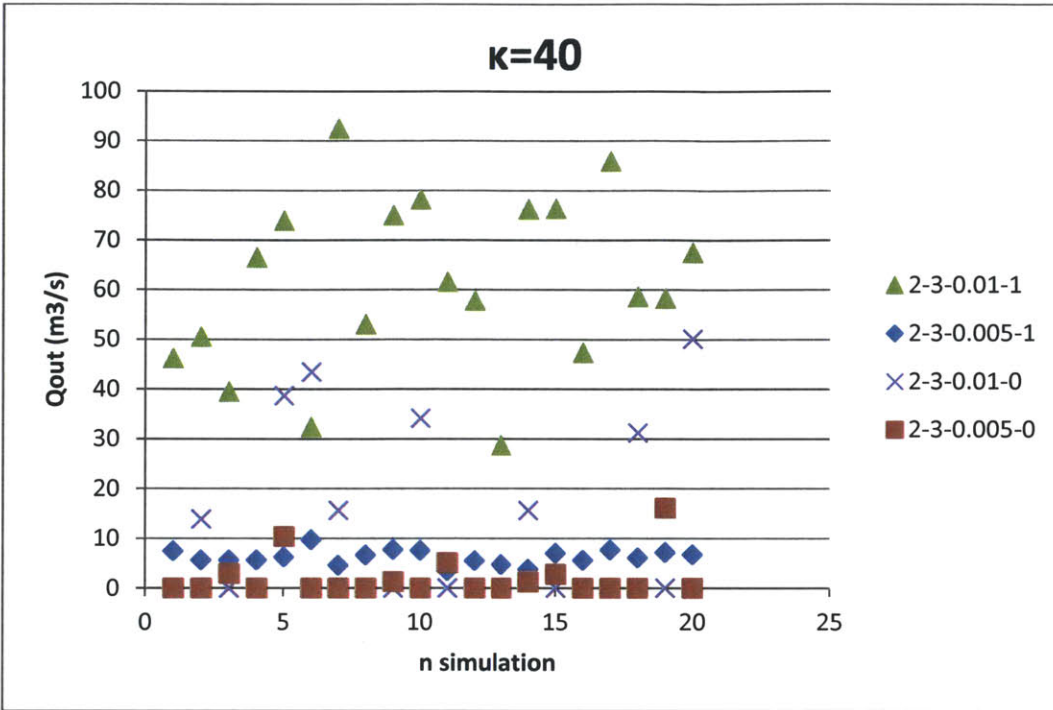


Figure 6.8- Example of representation of the Qout results

Appendix 1 – GEOFRAC: GUI – File list

RunGeofrac.m	Main file of the GUI. When running this file the GUI opens and it is possible to set the inputs before running Geofrac. It allows generating the fracture pattern and the flow pattern.
GeofracCallback.m	It contains the link formula between the GUI and Geofrac function.
GeofracGUI.m	File generated automatically by MATLAB when one creates a new GUI

Appendix 2 – GEOFRAC: GUI – Source code

```
-----  
  
function RunGeofrac  
  
% Creator: Alessandra Vecchiarelli.  
% RunGeofrac opens a GUI for hydro-mechanics simulation of a geothermal area.  
% Faculty: Prof. Herbert H. Einstein, Civil & Environmental Engineering.  
  
fig1 = openfig('GeofracGUI.fig', 'reuse');  
figure(fig1)  
  
set(fig1, 'DoubleBuffer',      'on')  
set(fig1, 'HandleVisibility',  'on')  
  
Xm_gui           = findobj( fig1, 'Tag', 'Xm_gui');  
Ym_gui           = findobj( fig1, 'Tag', 'Ym_gui');  
Ztop_gui         = findobj( fig1, 'Tag', 'Ztop_gui');  
P32_gui          = findobj( fig1, 'Tag', 'P32_gui');  
EA_gui           = findobj( fig1, 'Tag', 'EA_gui');  
Rot_gui          = findobj( fig1, 'Tag', 'Rot_gui');  
orientdistrib_radiobutton = findobj( fig1, 'Tag',  
'orientdistrib_radiobutton');  
bivFisher_radiobutton = findobj( fig1, 'Tag', 'bivFisher_radiobutton');  
UniMaxPhi_radiobutton = findobj( fig1, 'Tag', 'UniMaxPhi_radiobutton');  
Uniform_radiobutton = findobj( fig1, 'Tag', 'Uniform_radiobutton');  
UnifFisher_radiobutton = findobj( fig1, 'Tag',  
'UnifFisher_radiobutton');  
k1_gui           = findobj( fig1, 'Tag', 'k1_gui');  
k2_gui           = findobj( fig1, 'Tag', 'k2_gui');  
phimax_gui       = findobj( fig1, 'Tag', 'phimax_gui');  
k_gui            = findobj( fig1, 'Tag', 'k_gui');
```

```

nSimulation_gui          = findobj( fig1, 'Tag', 'nSimulation_gui');
flow_popup              = findobj( fig1, 'Tag', 'flow_popup');
Deterministic_radiobutton = findobj( fig1, 'Tag',
'Deterministic_radiobutton');
Stochastic_radiobutton  = findobj( fig1, 'Tag', 'Stochastic_radiobutton');
alpha_gui               = findobj( fig1, 'Tag', 'alpha_gui');
beta_gui                = findobj( fig1, 'Tag', 'beta_gui');
mod_gui                 = findobj( fig1, 'Tag', 'mod_gui');
hmax_gui                = findobj( fig1, 'Tag', 'hmax_gui');
hmin_gui                = findobj( fig1, 'Tag', 'hmin_gui');
pi_gui                  = findobj( fig1, 'Tag', 'pi_gui');
pf_gui                  = findobj( fig1, 'Tag', 'pf_gui');
Qout_gui                = findobj( fig1, 'Tag', 'Qout_gui');
RunGeofrac_button      = findobj( fig1, 'Tag', 'RunGeofrac_button');

set( flow_popup,        'String', {'flow', 'no flow'});
set( flow_popup,        'Value' , 1);
%set( orientdistrib_radiobutton 'String', {'bivFisher_radiobutton',
'UniMaxPhi_radiobutton', 'Uniform_radiobutton' , 'UnifFisher_radiobutton'});
%set( orientdistrib_radiobutton 'Tag');
set( nSimulation_gui,   'String', '1');
set( Xm_gui,            'String', '5');
set( Ym_gui,            'String', '5');
set( Ztop_gui,          'String', '10');
set( P32_gui,           'String', '2');
set( EA_gui,            'String', '3');
set( Rot_gui,           'String', '0');
set( k1_gui,            'String', '0.2');
set( k2_gui,            'String', '0.15');
set( phimax_gui,        'String', '3');
set( k_gui,             'String', '20');
set( alpha_gui,         'String', '0.01');
set( beta_gui,          'String', '1');
set( mod_gui,           'String', '0.1');
set( hmax_gui,          'String', '0.15');
set( hmin_gui,          'String', '0.002');
set( pi_gui,            'String', '4000');
set( pf_gui,            'String', '-1');

```

```
set( Qout_gui, 'String', '300');
```

```
set(RunGeofrac_button, 'Callback', 'GeofracCallback');
```

```
return
```

```
function GeofracCallback
```

```
fig1 = openfig('GeofracGUI.fig', 'reuse');
```

```
figure(fig1)
```

```
Xm_gui           = findobj( fig1, 'Tag', 'Xm_gui');  
Ym_gui           = findobj( fig1, 'Tag', 'Ym_gui');  
Ztop_gui         = findobj( fig1, 'Tag', 'Ztop_gui');  
P32_gui          = findobj( fig1, 'Tag', 'P32_gui');  
EA_gui           = findobj( fig1, 'Tag', 'EA_gui');  
Rot_gui          = findobj( fig1, 'Tag', 'Rot_gui');  
orientdistrib_radiobutton = findobj( fig1, 'Tag',  
'orientdistrib_radiobutton');  
bivFisher_radiobutton = findobj( fig1, 'Tag', 'bivFisher_radiobutton');  
UniMaxPhi_radiobutton = findobj( fig1, 'Tag', 'UniMaxPhi_radiobutton');  
Uniform_radiobutton = findobj( fig1, 'Tag', 'Uniform_radiobutton');  
UnifFisher_radiobutton = findobj( fig1, 'Tag',  
'UnifFisher_radiobutton');  
k1_gui           = findobj( fig1, 'Tag', 'k1_gui');  
k2_gui           = findobj( fig1, 'Tag', 'k2_gui');  
phimax_gui       = findobj( fig1, 'Tag', 'phimax_gui');  
k_gui            = findobj( fig1, 'Tag', 'k_gui');  
nSimulation_gui  = findobj( fig1, 'Tag', 'nSimulation_gui');  
flow_popup       = findobj( fig1, 'Tag', 'flow_popup');  
Deterministic_radiobutton = findobj( fig1, 'Tag',  
'Deterministic_radiobutton');  
Stochastic_radiobutton = findobj( fig1, 'Tag', 'Stochastic_radiobutton');  
alpha_gui        = findobj( fig1, 'Tag', 'alpha_gui');  
beta_gui         = findobj( fig1, 'Tag', 'beta_gui');  
mod_gui          = findobj( fig1, 'Tag', 'mod_gui');  
hmax_gui         = findobj( fig1, 'Tag', 'hmax_gui');  
hmin_gui         = findobj( fig1, 'Tag', 'hmin_gui');  
pi_gui           = findobj( fig1, 'Tag', 'pi_gui');  
pf_gui           = findobj( fig1, 'Tag', 'pf_gui');  
Qout_gui         = findobj( fig1, 'Tag', 'Qout_gui');  
RunGeofrac_button = findobj( fig1, 'Tag', 'RunGeofrac_button');
```



```

% get the values of the inputs
Xm_gui          = get( Xm_gui, 'String');
Ym_gui          = get( Ym_gui, 'String');
Ztop_gui        = get( Ztop_gui, 'String');
P32_gui         = get( P32_gui, 'String');
EA_gui          = get( EA_gui, 'String');
Rot_gui         = get( Rot_gui, 'String');
k1_gui          = get( k1_gui, 'String');
k2_gui          = get( k2_gui, 'String');
phimax_gui      = get( phimax_gui, 'String');
k_gui           = get( k_gui, 'String');
nSimulation_gui = get( nSimulation_gui, 'String');
alpha_gui       = get( alpha_gui, 'String');
beta_gui        = get( beta_gui, 'String');
mod_gui         = get( mod_gui, 'String');
hmax_gui        = get( hmax_gui, 'String');
hmin_gui        = get( hmin_gui, 'String');
pi_gui          = get( pi_gui, 'String');
pf_gui          = get( pf_gui, 'String');
Qout_gui        = get( Qout_gui, 'String');

% convert the values (string) into numbers
global Xm Ym Ztop mu P32 EA Rot k1 k2 phimax k nSimulation alpha beta mod
hmax hmin pi pf Qout
Xm          = str2num( Xm_gui);
Ym          = str2num( Ym_gui);
Ztop        = str2num( Ztop_gui);
P32         = str2num( P32_gui);
EA          = str2num( EA_gui);
Rot         = str2num( Rot_gui);
k1          = str2num( k1_gui);
k2          = str2num( k2_gui);
phimax      = str2num( phimax_gui);
k           = str2num( k_gui);
nSimulation = str2num( nSimulation_gui);
alpha       = str2num( alpha_gui);
beta        = str2num( beta_gui);

```

```

mod          = str2num( mod_gui);
hmax        = str2num( hmax_gui);
hmin        = str2num( hmin_gui);
pi          = str2num( pi_gui);
pf          = str2num( pf_gui);
Qout        = str2num( Qout_gui);

% radio button type of distribution
orientdistrib_radiobutton = get( orientdistrib_radiobutton, 'Tag');
bivFisher_radiobutton    = get( bivFisher_radiobutton, 'Tag');
UniMaxPhi_radiobutton    = get( UniMaxPhi_radiobutton, 'Tag');
Uniform_radiobutton      = get( Uniform_radiobutton, 'Tag');
UnifFisher_radiobutton   = get( UnifFisher_radiobutton, 'Tag');

global m;
if orientdistrib_radiobutton==bivFisher_radiobutton
    m=1;
elseif orientdistrib_radiobutton==UniMaxPhi_radiobutton
    m=2;
elseif orientdistrib_radiobutton==Uniform_radiobutton
    m=3;
else
    m=4;
end

% radio button aperture
Deterministic_radiobutton = get( Deterministic_radiobutton, 'Value');
Stochastic_radiobutton   = get( Stochastic_radiobutton, 'Value');

%other inputs

```

```

mu=P32;
r1=1;
for n=1:nSimulation
    [Qout]=geofrac(mu,EA,k,m,mod,hmin,hmax,pf,pi);
    % da aggiungere un file excel per lo storage dei risultati
    r2=horzcat(Qout);
    r1=vertcat(r1,r2);
end
header= 'Simulation';
colname={'Qout'};
xlswrite(r1,header, colname, 'Qout.xls');
return

% function orientdistrib_radiobutton_SelectionChangeFcn(hObject, eventdata,
handles)
% global m;
% if orientdistrib_radiobutton==handles.bivFisher_radiobutton
%     m=1;
% elseif orientdistrib_radiobutton==handles.UniMaxPhi_radiobutton
%     m=2;
% elseif orientdistrib_radiobutton==handles.Uniform_radiobutton
%     m=3;
% else
%     m=4;
% end

% % popup
% flow = get (flow_popup, 'Value');
% if flow==1
%     run geofrac;
% elseif flow ==2
%     run flowmodule;
% end

```

```

%% GEOFRAC:Primary, Secondary and Tertiary Processes
clear all;
tic
%function Qout=geofrac()

%user input data

EA=100; %expected fracture area
mu=1; %mu= intensity of the poisson plane network
m=4; % m=1: bivariate fisher , m=2: uniform maxphi;
      % m=3: uniform orientation, m=4: univariate fisher

% bivariate fisher k=[k1 k2]
% uniform maxphi k=maxphi
% uniform orientation k=[]
% univariate fisher k=k

k=1;

% create volume
% for test use cube

global Xm Ym Zinf Ztop
Xm=25;
Ym=25;
Zinf=0;
Ztop=50;

% rotation of the polygons
% rot=1 (for random); rot=0 (for no rotation)
global rot

rot=0;

% PRIMARY PROCESS: Generate planes and respective initial polygons

```

```

[fractureSet.plane, fractureSet.N]=...
    createFirstPolygon(mu,k,m);
% Calculate Equivalent Radius of the Planes

hpolygon=cell(1,length(fractureSet.plane));

    for j=1:length(fractureSet.plane)
        [hpolygon{j},T, d]=rotatePolygHoriz(fractureSet.plane{j},
fractureSet.N(j,:));
    end

        area= polygonArea(hpolygon);
        Re=getRe(hpolygon);
        Re=num2cell(Re);

lambda=1/EA; % intensity of the homogeneous Poisson point process that will
%induce the voronoi tessellation

% SECONDARY and TERTIARY PROCESSES: tessellation, random translation and
% random rotation

[fractureSet.polygon, fractureSet.Re, fractureSet.C, fractureSet.N1, ...
    fractureSet.area]= SplitFractureSetPPP (fractureSet.plane,lambda,...
    fractureSet.N, Re);

%clean cells that are empty
fractureSet.polygon(cellfun(@isempty,fractureSet.polygon))=[];
fractureSet.N1(cellfun(@isempty,fractureSet.N1))=[];
fractureSet.C(cellfun(@isempty,fractureSet.C))=[];
fractureSet.Re(cellfun(@isempty,fractureSet.Re))=[];
fractureSet.area(cellfun(@isempty,fractureSet.area))=[];
%fractureSet.drand(cellfun(@isempty,fractureSet.drand))=[];

load drand;

```

```
%% INTERSECTION MODULE
```

```
[frac, inter1, frac_int, inter_no, inter, fractureSet, count]=...  
    intersectionModule(fractureSet);
```

```
%
```

```
%calculates how many fractures, on average, each fracture intersects  
b=cellfun(@(x) size(x,2), frac_int, 'UniformOutput',false);  
b(cellfun(@(x) x==0, b))=[];
```

```
meanInt=mean(cell2mat(b))
```

```
%% CLEANING FRACTURES MODULE
```

```
g=frac_int;
```

```
%inter is COMPLETE
```

```
[g, inter_no, inter, polyB, polyE]=cleanModule(g, inter_no, inter,...  
    fractureSet.polygonall);
```

```
%% APERTURE GENERATION
```

```
%user input data
```

```
%Re=fractureSet.Reall;
```

```
    nPolygons=length(fractureSet.polygonall);
```

```
% alfa=0.01;
```

```
% beta=1;
```

```
% mod=0.05;
```

```
% hmax=0.1;
```

```
% hmin=0.002;
```

```
%varargin=[nPolygons, mod, hmax, hmin];
```

```
%varargin=[alfa; beta; Re];
```

```

%options: Deterministic, Probabilistic, Fixed value
fvalue=0.0005;
varargin=[nPolygons fvalue];
optionName='value';
%

fractureSet.h = generateAperture(optionName, varargin);

%% build list of edge connections:

clear temp;

% if no nodes intersect either the initial boundary or/and the final
% boundary, return Qout=0

if isempty(polyB) || isempty(polyE)
    disp('No fractures intersect at least one of the boundaries')
    return
end

[E,inter1, nodesB, nodesE]=buildNodesList(...
    fractureSet,inter_no,inter1,count,polyB,polyE);

% if no nodes intersect either the initial boundary or/and the final
% boundary, return Qout=0
% if isempty(nodesB) || isempty(nodesE)
%     disp('no fractures intersect at least one of the boundaries')
%     return
% end

%% FLOW MODULE (1.determining geometric paths)

%%A. Find paths%%

```

```

%A1. calculate mid point of intersections between fractures

midP=(inter1(:,1:3)+inter1(:,4:6))/2;

%A2. calculate score

[SC,W]=pathScore(E, inter1,frac,polyE,polyB,fractureSet,midP);

%A2. calculate the paths

E3=[E SC];

[costs,paths] = dijkstra(midP,E3,nodesB,nodesE);

%[costs1,paths1] = dijkstra(midP,E,nodesB,nodesE);

minPath=min(min(costs));

[m n]=find(costs==minPath);

minPath=paths{m,n};

if ~isnan(minPath)
minPath(1)=[];
minPath(end)=[];

minPath=frac(minPath',:);
minPath=minPath(:);
else
    disp('no paths')
    return %Qout=0;
end

```



```

%shortest distance
%
% minPath1=min(min(costs1));
%
% [m1 n1]=find(costs1==minPath1);
%
% minPath1=paths{m1,n1};
%
% if ~isempty(minPath1)
% minPath1(1)=[];
% minPath1(end)=[];
%
% minPath1=frac(minPath1',:);
% minPath1=minPath1(:);
% end
% toc

% %plotting path
% figure(2)
% for i=1:length(minPath)
%     hold on
%     fillPolygon3d(fractureSet.polygonall{minPath(i)},'r')
% end
% hold on;
% for i=1:length(minPath1)
%     hold on
%     fillPolygon3d(fractureSet.polygonall{minPath1(i)},'c')
% end

paths = cellfun(@(x) x(isnan(x)==0), paths, 'UniformOutput',false);
paths(cellfun(@isempty,paths))=[];
paths=paths(:)';

%
%% INTERSECTION OF BRANCHES: CREATION OF FLOW NETWORK

```

```

maxLength=max(cellfun(@(x) numel(x), paths));
p=cell2mat(cellfun(@(x) cat(2,x,NaN*ones(1,maxLength-
length(x))), paths, 'UniformOutput', false));
p=reshape(p,maxLength,length(paths));

for i=1:size(p,2)
    paths{i}=p(:,i);
    paths{i}(isnan(paths{i}))=[];
end

%determining adjacent nodes
tic
disp('FLOW MODULE...')
[nodes,branches, startNodes, endNodes]=FlowModule(paths);
toc
%sorting nodes
[S,I]=sort(nodes(:,3));
nodes=nodes(I,:);
%nodes=resolveBifurcation(nodes,branches);

[branches,nodes]=findEndNodes(nodes,branches);

%Find equivalent width, length and aperture of branches

[Weq, Leq,Heq]=findEq(branches,W,midP,fractureSet.h,frac, ...
    [nodesB polyB], [nodesE polyE]);

%creating network
[N,nodeEq]=createNetwork(nodes,startNodes(:,1),endNodes(:,1));

%% CALCULATING FLOW

% % branches geometrical characteristics (for testing only: randomly
assigned)
% % branEq(:,1) = thickness (h)

```

```

% % branEq(:,2) = width (w)
% % branEq(:,3) = lenght (l)

%
% %equivalent branches
branEq=[Heq' Weq' Leq'];

% %% Calculate Flow
%
% %INPUT:

global r f miu gamma
%
%roughness
r=0;
%friction factor
f=1+3.1.*(r./Heq).^1.5;
%water dynamic viscosity (T=20C): KN.s/m2
miu=0.4658*10^-6;
%water unit weight (T=20C): KN/m3
gamma=9.810;

pi=10e5; % injection pressure
pf=0; % out pressure; if =-1 it means not assigned (unknown)

disp('Calculating flow...')
[Qout,Q,P]=calculateFlow(startNodes,endNodes,branches,branEq,...
N,nodes,midP,nodeEq,pi,pf);
str=['Qtotal=',num2str(Qout),' m3/s'];
disp(str)
%% PLOTTING FLOW NETWORK

figure(1)

temp=nodes(:,1:2);

```

```

ee=midP(endNodes(:,1),:);
scatter3(ee(:,1),ee(:,2),ee(:,3),'filled','r');
% [~,loc]=ismember(nodeEq,endNodes(:,1));
% la=loc(loc~=0)';
% la=[find(loc~=0)' la];
% [~,I]=sort(la(:,2));
% la=la(I,:);
% la=la(:,1);
% %b=num2str(endNodes(:,1));
% la=num2str(la);
% la=strcat('(',la,')');
% dx=0.1; dy=0.1; dz=0.1;
% text(ee(:,1)+dx,ee(:,2)+dy,ee(:,3)+dz,la)

dx=0.1; dy=0.1; dz=0.1;

hold on;
bb=midP(startNodes(:,1),:);
scatter3(bb(:,1),bb(:,2),bb(:,3),'filled','g');
% [~,loc]=ismember(nodeEq,startNodes(:,1));
% la=loc(loc~=0)';
% la=[find(loc~=0)' la];
% [~,I]=sort(la(:,2));
% la=la(I,:);
% la=la(:,1);
% %b=num2str(startNodes(:,1));
% la=num2str(la);
% la=strcat('(',la,')');
% text(bb(:,1)+dx,bb(:,2)+dy,bb(:,3)+dz,la);

hold on;
for j=1:length(temp)
    d=midP(temp(j,:),:);
    dt=sum(d,1)/2;
    nbr=num2str(j);
    text(dt(1)+dx,dt(2)+dy,dt(3)+dz,nbr,'color','r')
    %hold on;

```

```

    plot3(d(:,1),d(:,2),d(:,3))

end

% intNodes=setdiff(temp(:),endNodes(:,1));
% intNodes=setdiff(intNodes,startNodes(:,1));
% iNodes=midP(intNodes,:);
% scatter3(iNodes(:,1),iNodes(:,2),iNodes(:,3),'filled','c');
% [~,loc]=ismember(nodeEq,intNodes);
% la=loc(loc~=0)';
% la=[find(loc~=0)' la];
% [~,I]=sort(la(:,2));
% la=la(I,:);
% la=la(:,1);
% %b=num2str(intNodes);
% la=num2str(la);
% la=strcat('(',la,')');
% text(iNodes(:,1)+dx,iNodes(:,2)+dy,iNodes(:,3)+dz,la);
%
% figure(2)
%
% for i=1:length(paths)
%     ddd=midP(paths{i},:);
%     hold on;
%     plot3(ddd(:,1),ddd(:,2),ddd(:,3),'--')
%     scatter3(ddd(:,1),ddd(:,2),ddd(:,3));
% end
%
% iNodes=midP(intNodes,:);
% scatter3(iNodes(:,1),iNodes(:,2),iNodes(:,3),'filled','c');

%end

% %%create an excel file to store data
n=[0 0 0 0 0 0 0 0 0];

```

```

P32mean= sum(fractureSet.Aall)/((Xm*2)*(Ym*2)*Ztop);
%EAn=pi*fractureSet.Reall.^2;
Reallmean= mean(fractureSet.Reall);
EAmean=mean(fractureSet.Aall);
ApertureAverage= mean (fractureSet.h);
%drandAverage= mean (fractureSet.drاند);
n1=horzcat (nPolygons, size(paths,2), P32mean, EAmean, ApertureAverage, Qout, meanIn
t, drاند, Reallmean);
n=vertcat (n, n1);

```

```

header{1}= 'Simulation Fisher';
colname={'nPolygons', 'paths', 'P32mean', 'EAmean', 'ApertureAverage', 'Qout', 'mea
nInt', 'drاند', 'Reallmean'};
xlswrite(n, header, colname, 'QoutTesting.xls');

```

toc

```

function varargout = GeofracGUI(varargin)
%GEOFRACTGUI M-file for GeofracGUI.fig
%
% GEOFRACTGUI, by itself, creates a new GEOFRACTGUI or raises the existing
% singleton*.
%
%
% H = GEOFRACTGUI returns the handle to a new GEOFRACTGUI or the handle to
% the existing singleton*.
%
%
% GEOFRACTGUI('Property','Value',...) creates a new GEOFRACTGUI using the
% given property value pairs. Unrecognized properties are passed via
% varargin to GeofracGUI_OpeningFcn. This calling syntax produces a
% warning when there is an existing singleton*.
%
%
% GEOFRACTGUI('CALLBACK') and GEOFRACTGUI('CALLBACK',hObject,...) call the
% local function named CALLBACK in GEOFRACTGUI.M with the given input
% arguments.
%
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".

```

```

%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GeofracGUI

% Last Modified by GUIDE v2.5 17-May-2013 12:19:20

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @GeofracGUI_OpeningFcn, ...
                  'gui_OutputFcn',  @GeofracGUI_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before GeofracGUI is made visible.
function GeofracGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for GeofracGUI

```

```

handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GeofracGUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = GeofracGUI_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function Xm_gui_Callback(hObject, eventdata, handles)
% hObject handle to Xm_gui (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'String') returns contents of Xm_gui as text
% str2double(get(hObject, 'String')) returns contents of Xm_gui as a
double

% --- Executes during object creation, after setting all properties.
function Xm_gui_CreateFcn(hObject, eventdata, handles)
% hObject handle to Xm_gui (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

```



```
% Hint: edit controls usually have a white background on Windows.
```

```
% See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
```

```
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
```

```
end
```

```
function Ztop_gui_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to Ztop_gui (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of Ztop_gui as text
```

```
% str2double(get(hObject,'String')) returns contents of Ztop_gui as a  
double
```

```
% --- Executes during object creation, after setting all properties.
```

```
function Ztop_gui_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to Ztop_gui (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
```

```
% See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
```

```
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
```

```
end
```

```
function Ym_gui_Callback(hObject, eventdata, handles)
```

```

% hObject    handle to Ym_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Ym_gui as text
%         str2double(get(hObject,'String')) returns contents of Ym_gui as a
double

% --- Executes during object creation, after setting all properties.
function Ym_gui_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Ym_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function P32_gui_Callback(hObject, eventdata, handles)
% hObject    handle to P32_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of P32_gui as text
%         str2double(get(hObject,'String')) returns contents of P32_gui as a
double

% --- Executes during object creation, after setting all properties.
function P32_gui_CreateFcn(hObject, eventdata, handles)

```

```
% hObject    handle to P32_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
```

```
%         See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function EA_gui_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to EA_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of EA_gui as text
```

```
%         str2double(get(hObject,'String')) returns contents of EA_gui as a
double
```

```
% --- Executes during object creation, after setting all properties.
```

```
function EA_gui_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to EA_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
```

```
%         See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```

function nSimulation_gui_Callback(hObject, eventdata, handles)
% hObject    handle to nSimulation_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of nSimulation_gui as text
%        str2double(get(hObject,'String')) returns contents of
nSimulation_gui as a double

% --- Executes during object creation, after setting all properties.
function nSimulation_gui_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nSimulation_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function hmin_gui_Callback(hObject, eventdata, handles)
% hObject    handle to hmin_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of hmin_gui as text
%        str2double(get(hObject,'String')) returns contents of hmin_gui as a
double

```

```

% --- Executes during object creation, after setting all properties.
function hmin_gui_CreateFcn(hObject, eventdata, handles)
% hObject    handle to hmin_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function beta_gui_Callback(hObject, eventdata, handles)
% hObject    handle to beta_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of beta_gui as text
%         str2double(get(hObject,'String')) returns contents of beta_gui as a
double

% --- Executes during object creation, after setting all properties.
function beta_gui_CreateFcn(hObject, eventdata, handles)
% hObject    handle to beta_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))

```

```

    set(hObject, 'BackgroundColor', 'white');
end

function hmax_gui_Callback(hObject, eventdata, handles)
% hObject    handle to hmax_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'String') returns contents of hmax_gui as text
%        str2double(get(hObject, 'String')) returns contents of hmax_gui as a
double

% --- Executes during object creation, after setting all properties.
function hmax_gui_CreateFcn(hObject, eventdata, handles)
% hObject    handle to hmax_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUiControlBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

function mod_gui_Callback(hObject, eventdata, handles)
% hObject    handle to mod_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'String') returns contents of mod_gui as text

```

```
%      str2double(get(hObject,'String')) returns contents of mod_gui as a
double
```

```
% --- Executes during object creation, after setting all properties.
```

```
function mod_gui_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to mod_gui (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
```

```
%      See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
```

```
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
```

```
end
```

```
function alpha_gui_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to alpha_gui (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of alpha_gui as text
```

```
%      str2double(get(hObject,'String')) returns contents of alpha_gui as a
double
```

```
% --- Executes during object creation, after setting all properties.
```

```
function alpha_gui_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to alpha_gui (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
```

```

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in bivFisher_radiobutton.
function bivFisher_radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to bivFisher_radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of bivFisher_radiobutton

% --- Executes on button press in UnifFisher_radiobutton.
function UnifFisher_radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to UnifFisher_radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of UnifFisher_radiobutton

% --- Executes on button press in Uniform_radiobutton.
function Uniform_radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to Uniform_radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Uniform_radiobutton

% --- Executes on button press in UnifMaxPhi_radiobutton.
function UnifMaxPhi_radiobutton_Callback(hObject, eventdata, handles)

```



```

% hObject    handle to UnifMaxPhi_radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of UnifMaxPhi_radiobutton

```

```

function k_gui_Callback(hObject, eventdata, handles)
% hObject    handle to k_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of k_gui as text
%        str2double(get(hObject,'String')) returns contents of k_gui as a
double

```

```

% --- Executes during object creation, after setting all properties.
function k_gui_CreateFcn(hObject, eventdata, handles)
% hObject    handle to k_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```

```

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function k2_gui_Callback(hObject, eventdata, handles)
% hObject    handle to k2_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

```

```

% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of k2_gui as text
%          str2double(get(hObject,'String')) returns contents of k2_gui as a
double

% --- Executes during object creation, after setting all properties.
function k2_gui_CreateFcn(hObject, eventdata, handles)
% hObject      handle to k2_gui (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%          See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function k1_gui_Callback(hObject, eventdata, handles)
% hObject      handle to k1_gui (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of k1_gui as text
%          str2double(get(hObject,'String')) returns contents of k1_gui as a
double

% --- Executes during object creation, after setting all properties.
function k1_gui_CreateFcn(hObject, eventdata, handles)
% hObject      handle to k1_gui (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB

```

```

% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function phimax_gui_Callback(hObject, eventdata, handles)
% hObject    handle to phimax_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of phimax_gui as text
%         str2double(get(hObject,'String')) returns contents of phimax_gui as
a double

% --- Executes during object creation, after setting all properties.
function phimax_gui_CreateFcn(hObject, eventdata, handles)
% hObject    handle to phimax_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in Stochastic_radiobutton.

```

```

function Stochastic_radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to Stochastic_radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Stochastic_radiobutton

% --- Executes on button press in Deterministic_radiobutton.
function Deterministic_radiobutton_Callback(hObject, eventdata, handles)
% hObject    handle to Deterministic_radiobutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
Deterministic_radiobutton

% --- Executes on button press in RunGeofrac_button.
function RunGeofrac_button_Callback(hObject, eventdata, handles)
% hObject    handle to RunGeofrac_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on selection change in flow_popup.
function flow_popup_Callback(hObject, eventdata, handles)
% hObject    handle to flow_popup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns flow_popup
contents as cell array
%         contents{get(hObject,'Value')} returns selected item from flow_popup

```

```

% --- Executes during object creation, after setting all properties.
function flow_popup_CreateFcn(hObject, eventdata, handles)
% hObject    handle to flow_popup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function pi_gui_Callback(hObject, eventdata, handles)
% hObject    handle to pi_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pi_gui as text
%         str2double(get(hObject,'String')) returns contents of pi_gui as a
double

% --- Executes during object creation, after setting all properties.
function pi_gui_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pi_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

end

```
function Qout_gui_Callback(hObject, eventdata, handles)
% hObject    handle to Qout_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Qout_gui as text
%        str2double(get(hObject,'String')) returns contents of Qout_gui as a
double
```

```
% --- Executes during object creation, after setting all properties.
function Qout_gui_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Qout_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function pf_gui_Callback(hObject, eventdata, handles)
% hObject    handle to pf_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pf_gui as text
```

```

%         str2double(get(hObject,'String')) returns contents of pf_gui as a
double

% --- Executes during object creation, after setting all properties.
function pf_gui_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pf_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function orientation_distribution_CreateFcn(hObject, eventdata, handles)
% hObject    handle to orientation_distribution (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% --- Executes when selected object is changed in orientdistrib_radiobutton.
function orientdistrib_radiobutton_SelectionChangeFcn(hObject, eventdata,
handles)
% hObject    handle to the selected object in orientdistrib_radiobutton
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none was
selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)

```

```

function Rot_gui_Callback(hObject, eventdata, handles)
% hObject    handle to Rot_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Rot_gui as text
%        str2double(get(hObject,'String')) returns contents of Rot_gui as a
double

% --- Executes during object creation, after setting all properties.
function Rot_gui_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Rot_gui (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```


CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Summary and conclusions

GEOFRAC is a discrete fracture pattern model combined with a fracture flow model that can represent the fracture-flow system in a geothermal reservoir. Recent developments have made GEOFRAC more efficient by basing it on Matlab, and it has been expanded by including an intersection algorithm and a flow model.

In Chapter 2 models and the software tools used to model fracture flow systems are presented. Each of them has some properties that make it difficult to model and to optimize a geothermal basin.

Chapter 3 describes the algorithm and the basic concept of GEOFRAC.

In Chapter 4 the results obtained from a parametric study with the fracture flow model GEOFRAC are analyzed and discussed. Specifically the influence of fracture aperture and orientation on flow was investigated. The parametric study demonstrates how aperture, the Fisher parameter for fracture orientation and rotation of the individual fractures influence flow rate. As to be expected greater apertures produce greater flow.

The effects of orientation are more complex as the effect of fracture plane orientation (Fisher parameter) and of rotation of individual fractures interact. For planes randomly generated the case with no rotation of the fractures and an aperture of 0.01 m generates greater flow than with rotation, while for parallel planes greater flow occurs in the case of rotation of the fracture and aperture equal to 0.01 m. This study of a simple synthetic case shows that the model produces consistent results but also that there are some unexpected complexities affected by fracture orientation.

Chapter 5 presents the extension of GEOFRAC with an algorithm that allows the modeling of wells intersecting the fractured region.

In Chapter 6 a GUI that was created in order to simplify the work to the user is presented.

In conclusion, GEOFRAC seems to accurately model the fracture flow system for a geothermal reservoir. The values of flow rates obtained in the parametric study are reasonably comparable to the ones found in the literature. The well intersection algorithm adds the possibility to properly include injection and production wells. (This feature can eventually be used to improve the location of the wells in order to obtain higher flow rate and pressure from the wells). The GUI presented in this thesis will allow any user to easily use GEOFRAC.

7.2 Future research

Further improvements of GEOFRAC are possible:

- Implementation of an algorithm to obtain the optimum position of the injection and production wells. The borehole intersection algorithm presented in Chapter 5 can be used to evaluate the fractures that intersect the wells for a given position. One of the most important improvements in the code would be to implement an algorithm to select the optimum position of the borehole after the fracture flow system is generated; the optimization should be done considering the most fractured area, and the fractures zone with the greatest aperture. This will increase the flow rate in the production well and may reduce the cost associated to exploration and construction of the wells.
- Test GEOFRAC with data obtained from a power plant in operation. The Rock Mechanics Group recently obtained data from a geothermal power plant in Iceland. These data will be used as input in GEOFRAC to further validate the code.
- Improve the GUI; add the thermal inputs of the algorithm implemented at this moment.
- Analysis of the change in porosity and permeability due to the cold fluid injection. The initial injection of water at 20° C may create a state of stress in the rock that induces more fractures in the system.

- Fluid physical conditions (change of density, change of phase liquid to vapor).
The assumption made up to now is a single phase fluid in its liquid state.
- Analysis of the temperature of the basin over a long period. Some studies show a reduction of the temperature over the long term. It would be useful to analyze this phenomenon and, if necessary, create an algorithm that will simulate the flow rate and temperature over time, and simulate this decrease in temperature.
- Geochemistry and reactive transport model.
- Matrix domain model (Double porous model). Fluid flow through a geothermal system is dominated by flow through the fractures while heat transfer is controlled by the interaction between matrix and fracture. The amount of flow from the matrix to the fracture affects heat transfer.

REFERENCE

- Apuani T., Corazzato C., Cancelli A., Tibaldi A. (2005).** Stability of a collapsing volcano (Stromboli, Italy): Limit equilibrium analysis and numerical modeling, *Journal of Volcanology and Geothermal Research* 144 (2005) 191– 210.
- Clearwater J., Burnell J., Azwar L. (2012).** Modelling the Ngatamariki geothermal system. *Proceeding, Thirty-Seventh Workshop on Geothermal Reservoir Engineering, Stanford University, Stanford, California, January 30 – February 1, 2012.*
- Coates R. T. and Schoenberg M. (1995).** Finite-difference modeling of faults and fractures, *Geophysics, VOL. 60, NO. 5 (September-October 1995); P. 1514-1526, 11 FIGS*
- Deng S., Podgorney R., Huang H. (2011).** Discrete Element Modeling of Rock Deformation, Fracture Network Development and Permeability Evolution Under Hydraulic Stimulation, *PROCEEDINGS, Thirty-Sixth Workshop on Geothermal Reservoir Engineering Stanford University, Stanford, California, January 31 - February 2, 2011*
- Dershowitz and Herda (1992).** Interpretation of fracture spacing and intensity, *Rock Mechanics*, Tillerson & Wawersik (eds), 1992 Balkema, Rotterdam.
- Dershowitz W. S. and Einstein H.H (1988).** Characterizing rock joint geometry with joint system models, *Rock Mechanics and Rock Engineering* 21, 21-51 (1988).
- Dershowitz W. S. and Herda (1992).** Interpretation of fracture spacing and intensity, *Rock Mechanics*, Tillerson & Wawersik (eds), 1992 Balkema, Rotterdam.
- Dershowitz, W.S. (2006).** Hybrid Discrete Fracture Network and Equivalent Continuum Model for Shaft Sinking, *ARMA/USRMS 06-1029*

- Diodato D.M. (1994).** A Compendium of Fracture Flow Models *Work sponsored by U.S. Department of Defense, United States Army, Europe, Combat Maneuver Training Center, Hohenfels, Germany 1994.*
- Einstein H.H. and LoCSin J.Z. (2012).** Modeling Rock Fracture Intersections and application to the Boston area, *Journal of geotechnical and geoenvironmental engineering, ASCE, November 2012.*
- Hatton C. G., Main I. G., Meredith P. G. (1994).** Non-universal scaling of fracture length and opening displacement, *Nature, Vol. 367, 13 January 1994.*
- Hicks T. W., Pine R. J., Willis-Richards J., Xu S., Jupe A. J., Rodrigues N. E. V. (1996).** A Hydro-thermo-mechanical Numerical Model for HDR Geothermal Reservoir Evaluation, *Int. J. Rock Mech. Min. Sci. & Geomech. Abstr. Vol. 33, No. 5, pp. 499-511, 1996*
- Hong Shen, Herbert Klapperich, Syed Muntazir Abbas, Abdelazim Ibrahim (2012).** Slope stability analysis based on the integration of GIS and numerical simulation, *Automation in Construction 26 (2012) 46–53.*
- Illman W. A., Hughson D. L. (2005).** Stochastic simulations of steady state unsaturated flow in a three-layer, heterogeneous, dual continuum model of fractured rock, *Journal of Hydrology 307 (2005) 17–37*
- Ivanova V. (1995).** Three-dimensional stochastic modeling of rock fracture systems, *Master thesis.*
- Ivanova V., Sousa R., Murrihy B., Einstein H.H. (2012).** Mathematical algorithm development and parametric studies with the GEOFRAC three-dimensional stochastic model of natural rock fracture systems.
- Jing L. (2003).** A review of techniques, advances and outstanding issues in numerical modelling for rock mechanics and rock engineering, *International Journal of Rock Mechanics & Mining Sciences 40 (2003) 283–353.*

- Johnston J. D. and McCaffrey K. L. W. (1995).** Fractal geometry of vein systems and the variation of scaling relationship with mechanism, *Journal of Structural Geology*, Vol. 18, Nos 2/3, pp. 349 to 358, 1996.
- Mancktelow, N.S., (1999).** Finite-element modelling of single-layer folding in elasto-viscous materials: the effect of initial perturbation geometry. *Journal of Structural Geology* 21, 161±177.
- Newsom J., Mannington W., Sepulveda1 F., Lane R., Pascoe R., Clearwater E., O’Sullivan M. J. (2012).** Application of 3d modelling and visualization software to reservoir simulation: Leapfrog geothermal and Tough2. *Proceeding, Thirty-Seventh Workshop on Geothermal Reservoir Engineering*, Stanford University, Stanford, California, January 30 – February 1, 2012.
- Podgorney R., Huang H., Gaston D. (2010).** Massively parallel fully coupled implicit modeling of coupled thermal-hydrological-mechanical processes for enhanced geothermal system reservoirs. *Proceeding, Thirty-Fifth Workshop on Geothermal Reservoir Engineering*, Stanford University, Stanford, California, February 1-3, 2010.
- Podgorney R., Lu C., Huang H. (2012).** Thermo-hydro-mechanical modeling of working fluid injection and thermal energy extraction in EGS fractures and rock matrix. *Proceeding, Thirty-Seventh Workshop on Geothermal Reservoir Engineering*, Stanford University, Stanford, California, January 30 – February 1, 2012.
- Reichenberger V., Jakobs H., Bastian P., Helmig R. (2005).** A mixed-dimensional finite volume method for two-phase flow in fractured porous media, *Advances in Water Resources* 29 (2006) 1020–1036.
- Sousa, R. (2013).** Three-Dimensional discrete fracture flow model. *Research report*, Massachusetts Institute of Technology, Cambridge, MA (draft).

- Staub I. (2002).** Models for the geometry of fractures, *Golden Associates AB report, May 2002.*
- Stone D. (1984).** Sub-surface fracture maps predicted from the Eye-Dashwa Pluton, Atikokan, Canada, *Int. J. Rock Mech. Min. Sci. & Geomech. Abstr. Vol. 21, No. 4, pp. 183-194, 1984.*
- Tezuka K. and Watanabe K. (2000).** Fracture network modeling of Hijiori hot dry rock reservoir by deterministic and stochastic crack network simulator (D/SC), *Proceedings World Geothermal Congress 2000 Kyushu - Tohoku, Japan, May 28 - June 10, 2000.*
- Vecchiarelli A., Li W. (2013).** Parametric study with GEOFRAC, MIT, *research report draft.*
- Veneziano D. (1978).** Probabilistic model of joints in rock, *Research Report, Dept. of Civil Engineering, June 1978.*
- Vermilye J. M. and C. H. Scholz (1995).** Relation between vein length and aperture, *Journal of Structural Geology, Vol. 17, No. 3, pp. 423-434, 1995.*
- Watanabe K. and Takahashi H. (1995).** Fractal geometry characterization of geothermal reservoir fracture networks, *Journal of Geophysical research, Vol. 100, B1, pages 521-528, January 10, 1995.*
- Willis-Richards J. and Wallroth T. (1995).** Approaches to the modelling of hdr reservoirs: a review, *Geothermics Vol. 24, No. 3, pp. 307-332, 1995.*
- Zhang L. and Einstein H.H. (2000).** Estimating the intensity of rock discontinuities, *International Journal of Rock Mechanics and Mining Sciences 37 (2000) 819-837.*
- Zhang, Y., Hobbs, B.E., Ord, A., Muhlhaus, H.B., (1996).** Computer simulation of single-layer buckling, *Journal of Structural Geology 18, 643±655.*

Zhang Y., Mancktelow N.S., Hobbs B.E., Ord A., Muhlhaus H.B. (2000). Numerical modeling of single-layer folding: clarification of an issue regarding the effect of computer codes and the influence of initial irregularities, *Journal of Structural Geology*, v. 22, p. 1511–1522.