IMPERIAL COLLEGE LONDON

DEPARTMENT OF EARTH SCIENCE AND ENGINEERING

APPLIED COMPUTATIONAL SCIENCE AND ENGINEERING MSc

INDEPENDENT RESEARCH PROJECT

A DISCRETE FRACTURE NETWORK GENERATION AND ANALYSIS LIBRARY FOR USE IN CAD SOFTWARE ENVIRONMENTS

BY

FALOLA, YUSUF

Fyf17@ic.ac.uk

https://github.com/Falfat

AUGUST 2019

SUPERVISORS: DR ADRIANA PALUSZNY AND DR ROBIN THOMAS

ABSTRACT

Discrete Fracture Network (DFN) modelling and simulation is an active area of research in earth science due to the inability to observe detailed 3D structure of the subsurface fracture network. There are few software packages available for DFN modelling and simulation. However, they are mostly complex to use, commercial, and closed sourced. Thereby, precluding any form of adaptability by researchers to functionalities not included in these packages. This work introduces an easy to use, open source library, Y-Frac, for DFN modelling and analysis. Y-Frac is built upon the python APIs available on Rhinoceros 6. Hence, Y-Frac is fit for use on Rhinoceros software package. Y-Frac can model fracture networks containing circular, elliptical and regular polygonal fractures. This library is computationally cheap for DFN modelling and analysis. Some of the functionalities of this library for DFN analysis include fracture intersection analysis, cut-plane analysis, and percolation analysis. Algorithms for constructing an intersection matrix and determining percolation state of a fracture network are also included in this work. The output text file from this library containing modelled fracture networks' parameters can serve as input for appropriate software packages to simulate flow and perform mechanical analysis in fracture networks. The practical applicability of Y-Frac was demonstrated by performing percolation threshold analysis of 3D fracture networks and comparing the results to published data.

Keywords: Discrete Fracture Networks, Python, Rhinoceros, DFN modelling, Percolation threshold, Y-Frac.

Code metadata:

Current code version	v 1.0.0
Link to repository used for this code version	https://github.com/msc-acse/acse-9-independent-research-project-Falfat
Legal Code License	MIT
Code versioning system used	git
Programming languages, tools and services used	Python, Spyder IDE
Technical platform for software implementation	Rhinoceros 6 for windows https://www.rhino3d.com/download Rhinoceros user guide http://docs.mcneel.com/rhino/6/usersguide/en-us/index.htm
Compilation/hardware requirements, operating environments and dependencies	Operating system supported: Windows 10, 8.1 or 7 SPI Hardware: 8GB RAM or higher, 600MB disk space, 63 CPU cores or less, a multiple-button mouse with scroll wheel.
Link to software documentation	https://github.com/msc-acse/acse-9-independent-research-project-Falfat/tree/master/Documentation
Link to Rhinoceros API library	https://developer.rhino3d.com/api/RhinoScriptSyntax/
Support email for questions	Fyf17@ic.ac.uk

1.0 Introduction

Generally, fractures represent a space between planes [1] and are ubiquitous in many natural, engineered and biological materials. This discontinuity may be because of mechanical failure, from tectonic events [2] or human factors such as hydraulic fracturing, tunnel evacuation [3] or chemical processes, e.g. weathering [4]. The term therefore includes faults, joints, fissures, cleavages and even discontinuities between mineral particles [5]. A fracture can be on a scale of a few microns (e.g. microcracks) to several kilometres (e.g. faults). Fractures are essential in science and engineering as they play a significant role in material strength, rock block stability, and in creating pathways for fluid flow [6,7]. Fractures are of great research interest in various fields of studies, not limited to, geotechnical applications, reservoir engineering, waste disposal, mining engineering, and earthquake studies [8].

A fracture network is a system of fractures developed within the same rock volume. A network may involve several distinct fracture sets, which may or may not intersect [9]. These sets generally evolve and vary in their spatial distribution [10]. Fracture network modelling and simulation is an active area of research which has received much attention in the last decade, primarily due to the challenge of directly observing the detailed 3D structure of fracture networks deep in the crust [11]. Fractures are complex objects in terms of their geometry and topology, and they occur at all scales, which make their modelling and simulation an exciting area of research. Increased computing software and hardware capabilities have contributed to the rapid growth in the modelling of fractured rock [12]. Direct observations of fracture networks are relatively scarce and are limited to surface outcrops (2D), tunnel wall (2D), and core drilling (1D) [13,14]. Although seismological surveys may be able to locate 3D large-scale structures, current technology can hardly detect widespread medium and small fractures due to resolution limits [11].

Lei et al. [11] defined a Discrete Fracture Network (DFN) as "a computational model that explicitly represents the geometrical properties of each fracture (e.g. size, orientation, position, shape and aperture), and the topological relationships between individual fractures and fracture sets", as opposed to continuum modelling which models the entire system as one domain [14,15]. DFN models provide an effective method for simulating and studying features of fractured rock [16]. There are few commercial and non-commercial software capable of DFN modelling. However, they are often; expensive (e.g. MVE [17], NAPSAC [18], FracMan [19], MoFrac [20]), designed for specific tasks, have limited or no functionality for extensive research in the academia by being closed source, and complex to use [21,22].

This work presents a python-based open-source library, **Y-Frac**, library for simulation and analysis of three-dimensional stochastic discrete fracture networks, utilising the Rhinoceros 3D commercial CAD (Computer-Aided Design) application. Rhinoceros CAD software is an industry-standard tool, which offers a user-friendly and cost-effective platform for handling and manipulating 3D geometries. Y-Frac adds some functionalities, pertinent to geoscience and engineering, to Rhinoceros software. Y-Frac expands on the python API provided by Rhinoceros to put forward a state-of-the-art python library for DFN generation and analysis. Additionally, with the availability of various output file formats on Rhinoceros software, results from this library can be exported for use on a variety of software applications. Y-Frac consists of methods for DFN generation, regeneration, analysis/characterisation and postprocessing. Y-Frac provides an efficient tool for studying the science of fracture networks.

Based on fracture data retrieved from rock samples by geoscientists and engineers, this free and non-sophisticated tool can be used to generate a synthetic rock mass (SRM) model. This fracture model can thus be analysed for fracture intensity, cut-plane features, network connectivity, and percolation state using Y-Frac.

The main contributions of this work include provision of: a computationally cheap object-oriented Rhinoceros' python script library for Discrete Fracture Networks generation, a platform to generate fracture networks of different shapes (disks, ellipses and polygons) and sizes, fracture network analysis via Rhinoceros and postprocessing on python IDEs for visualisations, a method for re-generating fracture networks from a text file containing fracture properties of pre-generated fracture networks, an innovative technique for fracture network intersection analysis and determination of percolation state.

This work is divided into sections. Section 2 gives a brief review of DFN's history and some important recent work on the subject matter. It further summarises some existing DFN simulation tools, their functionalities and shortcomings. A short discussion on fracture network analysis and background of Rhinoceros 6 are also contained in this section. Section 3 introduces the architecture and discusses the structure of Y-Frac. The section also contains an explanation of the software development strategy employed in this project. It further describes some of the essential functionalities (with example codes) of Y-Frac and provides algorithms of some Y-Frac's essential methods. In section 4, the usage of Y-Frac was demonstrated. Furthermore, analysis of its computational cost and percolation threshold was presented. Lastly, section 5 presents the conclusion of this study and future work to improve the applications of Y-Frac.

2.0 Review on Discrete Fracture Network

Various modelling approaches have been employed to represent fractured rock. Xing & Sanderson [23] categorised these approaches into discrete and continuous models. The primary assumption of continuum-based methods is that the computational domain is treated as a single body [24], as opposed to discrete methods in which each fracture is characterised within a structural domain [25]. However, hybrid models exist which involve the combination of these methods [16].

Stochastic DFN studies began in the 1980s to investigate two essential topics: percolation theory and hydrogeology [16]. Long et al. [26] developed methods to determine if between fractured systems should be treated as an equivalent porous (continuum) medium or as a collection of the discrete fracture flow path. They extended their work to 3-D when they modelled the steady flow of fluid in disc-shaped random network fractures [27]. Endo et al. [28] and Endo [29] performed tracer experiments and suggested that not all fractures behave like equivalent porous media. They observed that fracture systems with continuous fractures have directionally dependent hydraulic effective porosity which negates the idea of an equivalent porous medium. Balberg et al. [30] used stochastic networks to study the dependence of the percolation threshold of the 3-D sticks systems on aspect ratio and macroscopic anisotropy. They employed the Monte Carlo technique to determine percolation thresholds for randomly placed sticks in a domain. Andersson & Dverstorp [31] investigated how to predict flow through a network of discrete fractures in 3-D space. This technique has continuously developed afterwards with many applications in civil, environmental and reservoir engineering and other geoscience fields.

A recent review on DFN by Lei et al. [11], grouped DFNs into three categories:

Geological-mapped DFNs, which are generated using datasets observed from outcrops such as analogue mapping, borehole imaging, aerial photographs, and seismic surveys. This method ensures the preservation of geological realism and fair characterisation of complex topologies (e.g. intersection, spacing, clustering, truncation and hierarchy). However, it can be quite challenging to collect geological data, mainly due to limited access to the subsurface [32]. Besides, sampled geological data cannot be said to be fully representative, since observations are usually made from a locality of the rock mass [33]. Geological measurements are mostly in 2-D. Therefore, accuracy is lost when upscaling to 3-D. Although seismic measurements are in 3-D, they are limited in terms of resolution.

Stochastic DFNs use statistical principles to create fracture datasets. Fracture properties used as variables include fracture lengths, orientations, locations and shapes. Advantages of this method include ease of generation as compared to geological mapping, the possibility of automation, efficient fracture generation, viability for both 2-D and 3-D, and applicability for various scales. However, the shortcomings of this technique involve oversimplification of fracture geometries and topologies, uncertainties in statistical parameters, and negligence of physical processes.

Geo-mechanical DFNs incorporate fracture physics and fracture network evolution in simulating fracture datasets. This technique involves the use of rock and fractures mechanical properties and paleo-stress conditions as inputs. It links geometry with physical mechanisms and ensures the correlation between different fracture attributes. However, they require large amounts of computational resources, have difficulties in including hydrological, thermal and chemical processes into simulations and uncertainties in input properties. Nonetheless, modern DFN studies have continuously made use of fracture data collected from geological mapping to refine the input parameters required for stochastic DFNs [25], both as a standalone tool [14,34,35] or integrated within more complex geomechanical simulations [16,36,37], leveraging the strengths of the three techniques to create more realistic fracture datasets.

2.1 Existing DFN tools.

There are many DFN simulation tools, both as standalone commercial software programmes and closed source in-house computer codes. Notable standalone DFN commercial software include Midland Valley Exploration, NAPSAC, FracMan, and MoFrac;

Midland Valley Exploration (MVE) fracture modelling tool uses geometrical properties such as stress and strain values and statistical properties such as curvature as proxies for intensity and orientation for DFN generation. It allows for multiple direct inputs of filed data to constrain the production of DFNs. This tool is applicable to fracture networks characterisation and provides direct output on a geocellular model, which shows the current state of the subsurface in 3-D by representing a complex geological reservoir with millions of cells [17].

NAPSAC generates fracture networks stochastically, ensuring fracture properties have the same statistical attributes as those of the geologically mapped field data. It employs the finite element method to simulate fluid flow in fracture networks. Users can validate model predictions against data from hydro-geological experiments (e.g. well tests). Its applications include effective permeability determination, porosity prediction, geometric and percolation analysis, transport and flow modelling. [38].

GOLDER's FracMan software suite allows users to generate both randomly generated DFNs and DFNs guided by information from mapped structures. DFNs upscaling to an equivalent porous medium (continuum) model is possible, without losing previous details of DFNs. It is a versatile software suite used in various sectors including mining engineering (e.g. fragmentation assessment and hydrogeological evaluation), nuclear (e.g. groundwater and solute transport), oil and gas (e.g. fracture modelling and geomechanics simulation), and civil engineering (e.g. groundwater evaluation) [19].

MoFrac computer software generates DFNs from user-defined fracture properties, including fracture intensity, orientation, truncation rules, size, shape and undulation. The generated models simulate realistic fracture representation by constraining the fracture lengths and intensities using cumulative length distribution (CLD) and cumulative area distribution (CAD) respectively. It ensures the input data is honoured during the stochastic process of fracture generation. Its applications are mainly in geomechanical and hydraulic flow modelling such as mine design, hydraulic fracturing, rock characterisation, blast optimisation, fracture mechanics, reservoir modelling and waste transport [20].

Summarily, these software packages are stand alone and not to be integrated. By generating DFNs using CAD software, we can produce various output file formats which can be used in other software packages. This project also aims to make its DFN tool easy to use in terms of network generation and analysis. Users only need to define input parameters using a text file. Furthermore, being an open-source code, there is room for its continuous development by the public and adaptation to suit different needs.

Other hybrid, complex, industry and academic DFN simulation tools include SIDNUR, Flow123d, FracSim3D, ADFNE, DFNWorks, and ICGT.

SIDNUR [21,39] generates different realisations of DFNs following the Monte Carlo process and solves for steady-state flow in each of these networks whose statistical properties are constrained by in-situ experiments. They adopted a non-conforming discretisation of the DFN to minimise the number of unknowns and ease mesh refinements. Fracture edges and domains are discretised in a staircase-like manner [39]. Additionally, the mortar method is used to handle fracture-fracture intersections. This method involves specifying one of the fractures as a master and the other a slave [40]. The resulting linear system is a semi-positive definite matrix containing millions of unknowns after a Dirichlet boundary condition is enforced. The Balanced Domain Decomposition [41] algorithm is applied to solve the linear system. This highly parallelised tool is distinguished by its method of discretisation of the fracture networks and solving of the linear system. However, it only simulated steady-state flow in fracture networks.

Flow123d [42–47] combines continuum and DFN models to simulate water, solute and heat transport in fractured porous media. This software is distinct in that it supports numerical computations on complex meshes, which consists of elements of various dimensions. This enables Flow123d to combine both continuum and discrete fracture models. Additionally, unlike SIDNUR, it solves both steady and unsteady Darcy flow in fracture networks. It applies different solvers for specific problems. Heat transfer and solute transport problems in fracture networks are solved with finite volume method, discontinuous Galerkin method solves convective transport model, and mixed-hybrid formation of finite element method is applied to solve steady and unsteady Darcy flow in fracture networks.

FracSim3D [48–52] uses marked point process, a technique which considers fracture properties as marks associated with stochastic points, to simulate 2D and 3D DFNs. Geometries and properties of fractures are modelled using probability distribution. Fracture locations are simulated using the Cox process model, homogeneous Poisson point process, non-homogeneous Poisson point process, and cluster point process. Fracture orientation and size are determined by Monte-Carle simulation of their probability distribution. The probability distribution for fracture orientation includes uniform, wrapped normal, Von-Mises and Fisher's, while those for fracture size are uniform, exponential and lognormal. Additionally, functionalities for cut-plane, rectangular window, scanline, and core samplings are available on this tool. It provides an output CSV file containing information about fracture networks for subsequent use in fracture network flow modelling.

ADFNE [12] is a DFN tool capable of 2D and 3D simulation. It is written in Matlab and contains various functions for fracture generation and characterisations. Fraction locations are simulated using uniform or Poisson's distribution, fracture lengths are modelled by a negative exponential distribution, and orientations are determined by Fisher's distribution. This tool can be used for fracture characterisations (e.g., clustering, intersection and connectivity analyses), fluid flow applications (using Finite Difference Method) and geometric modelling (e.g., complex polygonal fracture faces). However, this tool does not provide functionality for determining percolation state of fracture networks and fracture intensity.

DfnWorks [22] is a parallelised 3D DFN tool kit which uses the Finite Volume Method to solve flow equations for transport simulation. It requires several additional packages like PETSC libraries, Mathematica, PFLOTRAN and LaGrit. DfnWorks consists of three main packages; DFNGEN, DFNFLOW and DFNTRANS. DFNGEN creates the fracture network from observed data and does the Delaunay triangulation of the medium using LaGrit. PFLOTRAN calculates steady-state pressure in the fractured medium after the mesh output from DFNFLOW has been converted to an appropriate input in DFNFLOW. Lastly, DFNTRANS reconstructs the Darcy flow rate from DFNFLOW to track particles in the fractured medium, while the Lagrangian approach introduced by Painter et al. [53] to determines pathlines through the network and simulates transport.

ICGT [54–61] is a 3D DFN simulator based on the Finite Element Method. It grows discrete fractures from initial flaws and takes the geomechanical properties of fractures into consideration to ensure fracture growth and interactions are realistic. It is integrated with CSMP++, an object-oriented finite element-based library for automatic meshing. Fracture geometries are represented by NURBS surfaces, which ensures geometries are kept distinct from meshes. The workflow of this tool summarily involves defining and updating fracture geometry by resolving thermo-hydro-mechanical and flow models until fracture growths are no longer perceived. Fracture growth is benchmarked by input failure and propagation. During this process, conditions in the fractured medium such as thermo-poroelastic deformation, contact stresses, fracture tip growth and stress intensity factor, local aperture, and permeability, are calculated continuously and updated.

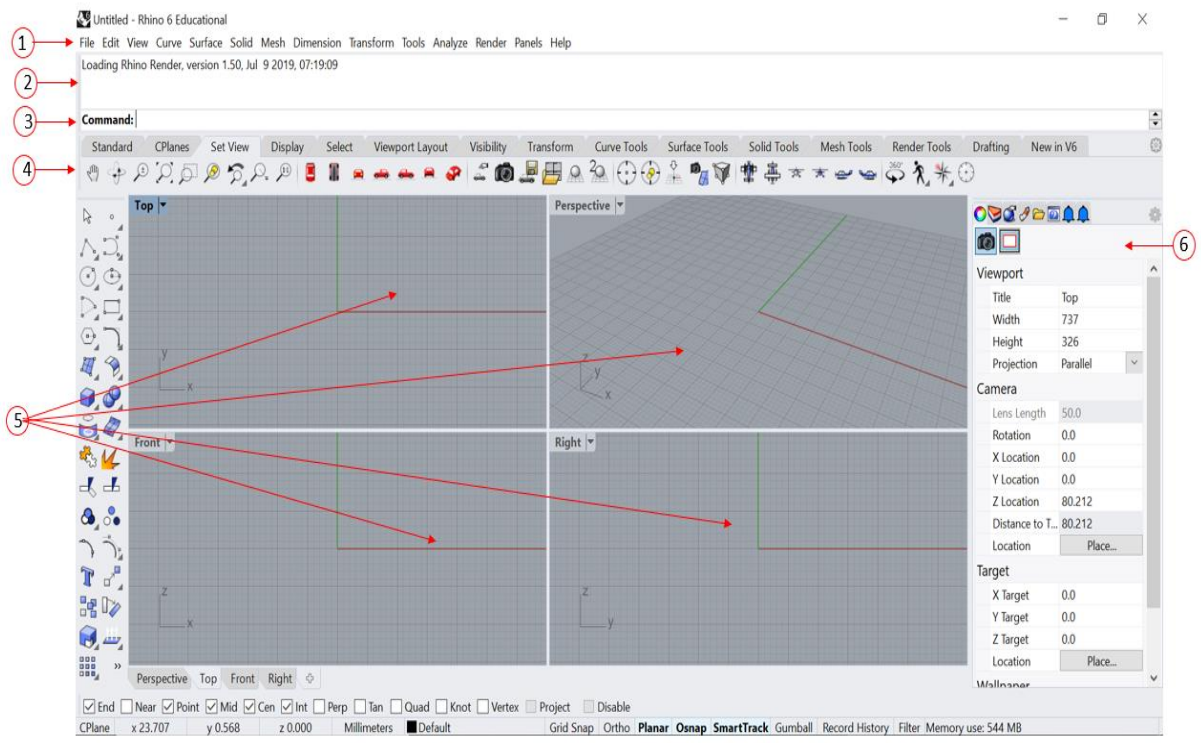
Except for ADFNE, which employs a simple finite difference method to solve for fluid flow, all other tools depend on involved mathematical formulations to simulate fluid flow in fractured networks, which makes them complex to use and understand their backends. These codes are closed source, except DfnWorks and ADFNE, preventing any form of development and adaptation by other researchers.

2.2 Rhinoceros and CAD Softwares

Groover and Zimmers [62] defined Computer-Aided Design (CAD) as a form of technology which creates, modifies, analyses, and optimises a design on a computer system. CAD software packages have been in use since the early 1970s [63]. Examples of modern CAD software packages include;

AutoCAD [64], SolidWorks, CATIA [65], Inventor [66], Pro/ENGINEER [67], Rhinoceros [68] and 3ds Max [69]. CAD software packages are the backbone of modern engineering designs, due to benefits such as automation of repeated tasks, tracking of previous design realisations, making changes without reproducing an entire drawing, and high-quality rendering [70].

Rhinoceros is a 3D computer graphics and CAD software package developed by Robert McNeel and Associates [71]. It is primarily a surface modelling tool, based on NURBS (Non-Uniform Rational B-splines) [68]. NURBS are mathematical representations for modelling geometry ranging from simple lines to a complex 3D surface or solid [72]. The primary geometry objects in Rhinoceros used in this are points, curves, surfaces, and poly-surfaces [73]. Professionals in aerospace, marine, automobile, sport and medical industries make use of Rhinoceros for standard designs [73]. In this work, we chose Rhinoceros because it allows the user to write python scripts and includes a Python API to facilitate this. Rhinoceros is well documented and actively developed, fast, relatively cheap to procure, and available for student's download on a free trial [74,75].



Untitled - Rhino 6 Educational
1
File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Panels Help
2
3
Command:
4
Standard CPlanes Set View Display Select Viewport Layout Visibility Transform Curve Tools Surface Tools Solid Tools Mesh Tools Render Tools Drafting New in V6
5
Top
Perspective
Front
Right
6
Perspective Top Front Right

```
Loading Rhino Render, version 1.50, Jul  9 2019, 07:19:09
```

Viewport	
Title	Top
Width	737
Height	326
Projection	Parallel
Camera	
Lens Length	50.0
Rotation	0.0
X Location	0.0
Y Location	0.0
Z Location	80.212
Distance to T...	80.212
Location	Place...
Target	
X Target	0.0
Y Target	0.0
Z Target	0.0
Location	Place...

☑ End ☐ Near ☑ Point ☑ Mid ☑ Cen ☑ Int ☐ Perp ☐ Tan ☐ Quad ☐ Knot ☐ Vertex ☐ Project ☐ Disable

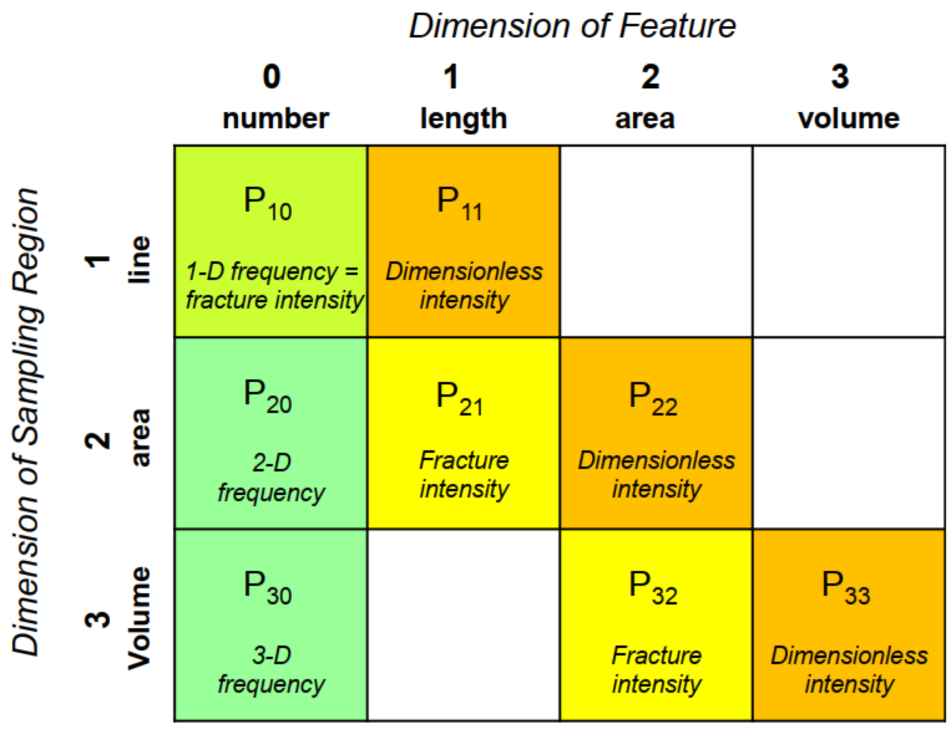
CPlane x 23.707 y 0.568 z 0.000 Millimeters ■Default Grid Snap Ortho **Planar** **Osnap** **SmartTrack** Gumball Record History Filter Memory use: 544 MB

Figure 1. Rhinoceros 6 user interface

Figure 1 depicts the Rhinoceros 6 user interface. Label (1) indicates the menu which groups Rhinoceros' commands by function. Label (2) specifies the command history window which displays previous commands and prompts, and it also shows the print statements of Rhinoceros python script. Label (3) designates the command prompt, which displays prompts for current command actions. Label (4) points at the toolbars containing graphical icon buttons for starting commands. Label (5) indicates the viewports displaying the Rhino working environment. The panels containing layers, properties and settings are indicated by label (6) [68]. Fracture networks created on Y-Frac are displayed on the Rhinoceros interface. The perspective view mode displays the networks in 3-D. Rhinoceros python script is run by clicking the Tools tab, then PythonScript tab, and lastly the Edit tab, where Y-Frac's functionalities can be used.

2.3 Fracture Network Analyses

There are various measures employed to characterise and analyse fracture networks. One of such methods used in DFN studies to describe fracture abundance, which can be quantified in several ways, including fracture frequency [76,77], fracture density [78] and fracture intensity [77]. Fracture frequency refers to the number of fractures normalised by the line length [1D, L^{-1}], sample area [2D, L^{-2}] or domain volume [3D, L^{-3}]. Fracture intensity originally referred to the total trace length per unit area with dimensions [L^{-1}] but has been extended to 1- and 3-dimensions with same units [L^{-1}] [9]. Fracture intensity includes fracture size in its calculation, and it is more common in DFN studies [16]. However, these three terms have been used interchangeably in the literature [79,80]. Dershowitz and Herda [76] formally introduced the P_{xy} system to describe these features, where x refers to the feature dimension, and y depicts the sampling region dimension. Figure 2 summarises these features.



Dimension of Feature

Dimension of Sampling Region	0 number	1 length	2 area	3 volume
1 line	P_{10} 1-D frequency = fracture intensity	P_{11} Dimensionless intensity		
2 area	P_{20} 2-D frequency	P_{21} Fracture intensity	P_{22} Dimensionless intensity	
3 Volume	P_{30} 3-D frequency		P_{32} Fracture intensity	P_{33} Dimensionless intensity

Figure 2: P_{xy} system for fracture intensity, frequency and density definitions (Adapted from Sanderson and Nixon [9]).

Fracture connectivity is essential in studying the flow behaviour and stability of a rock mass [81]. Fracture connectivity can be described as a measure of the extent to which the fractures in a network are interconnected, or, in terms of the percolation concept- a threshold below which the system is not connected and above which it is connected [9]. The percolation threshold of a system correlates with its permeability and thus defines the transport properties of the system [82]. Domain connectivity is of great importance in DFN applications such as oil and gas recovery and waste disposal. Various studies [30,82–84] have been carried out to discuss percolation theory and percolation threshold in 2D and 3D systems. Dimensionless parameters are involved in defining connectivity because connectivity is scale-independent. Therefore, measures such as fracture intensity [L^{-1}] are unsuitable for determining fracture connectivity [9]. Balberg et al. [30] introduced the concept of “excluded volume” for determining the percolation threshold of fracture networks. They defined the excluded volume of a fracture as “the volume surrounding it, in which the centre of another object must be found for the two objects to intersect". This concept underscores the scale independence of the percolation threshold. The dimensionless density is

$$\rho' = \rho V_{ex} \qquad [1.0]$$

Where ρ is the fracture density and V_{ex} is the excluded volume [85]. For two dimensional convex objects F_1 and F_2 with Areas A_1 and A_2 and perimeters P_1 and P_2 which are randomly oriented, their excluded volume is

$$V_{ex} = \frac{1}{4}(A_1P_1 + A_2P_2) \qquad [2.0]$$

If they are identical, this formula reduces to

$$V_{ex} = \frac{1}{2}AP \qquad [3.0]$$

Fracture intersections create pathways in a network. Increased number of intersections in a network inmproves the chances for the network to percolate [86]. Various measures reported in literature such as intersection per unit length, lengths of intersections per unit of surface area [9], connectivity index [49], and intersection density [87] all rely on fracture intersection analysis.

Cut-planes provides the opportunity to visualise the internal geometry of a 3D fracture network in 2D. Inference about the hydraulic properties of fracture networks and intersection observations are possible through cut-plane analysis [16]. Studying the cut-planes is also a useful way to decipher if the fracture percolates or not. See appendix 3 for examples of fracture network cut planes.

3.0 Software Development Strategy and Features

This library consists of various independent methods. The library was developed from scratch leveraging on Rhinoceros 6 python API and is not an extension of pre-existing code. Rhinoceros APIs give access to python functions stored in Rhinoceros. Hence, this work currently serves as the state-of-the-art in the realm of DFN generation and analysis on CAD software environments.

In building this library, git [88] was used as a version control tool for efficient code merging, tracking and integration, while this library is being released and monitored via GitHub. Git and GitHub were adopted for this project because they are open-sourced, easy to use and manage, ability to push and pull from the GitHub repository without connecting to the internet, strong community support due to their popularity, and the availability of git's built-in support in every major IDE.

The development of this library follows the V-Model[89] method of software development, a variant of Waterfall methodology [90]. This method is also known as the verification and validation model. This method is a sequential development approach in which the following phases are followed sequentially: software requirement analysis and specification, software architecture design, module design, coding, deployment and maintenance. This methodology was preferred for various reasons. It involves testing after each development phase, it makes the project more natural to understand and manage, the project plan and schedule was made easy to follow. For example, the methods to generate fractures needed to be tested and confirmed functional before writing modules for fracture analysis such as P_{21}, P_{32}, intersection analysis. Modifications were made to previous phases when there was a need, but testing was always done to confirm the functionality of each method. A module, as used here, refers to a python file containing classes, methods and functions.

The library built for DFN generation and analysis through this research was based on modular programming. This coding style was employed to make this library as compact as possible, maintainable and simple to use. Each module contains functions, classes or both. The basic modules could further be categorised based on their roles, namely; DFN generation, DFN analysis, DFN regeneration, DFN postprocessing, and DFN input. Tables 1, 2 and 3 show the architectural design of this library, containing the modules and the function and classes they contained.

Modules and Methods for Discrete Fracture Generation		
DFN_Gen	**Domain**	**Frac**
+ GeneratePoint()	+_init_()	Frac.Fracture
+ FractureSize()	+Show()	+Intersect()
+PolyOrientation()	+NumberOfFractures()	+OldFracturesGuid()
+InclinePlane()	+CreateBoundary()	+NewFracturesGuid()
+FixedFractureGen()	+GetSurfaceFromFractureLayer()	
+RandomFractureGen()	+ConvertPolySurfaceToSurface()	
+SeparatedFractureGen()	+CreateSetOfExtensionBounsaries()	
	+RemoveSurfacesIfAnyIntersectBoundary()	
	+RemoveSurfacesOutsideOfBox()	

Table 1: Architecture for fracture network generation

Modules and Methods for Fracture Network Analysis		
Domain	**DFN_Analysis**	**Matrix**
Domain.Domain	DFN_Analysis.IntersectionAnalysis	Matrix.Matrix
+_init_()	+LengthOfIntersection()	+_init_()
+IntersectionMatrix()	+FractureSurfaceArea()	+PrintMatrix()
+Percolate()	+FractureIntensity_P32()	+MatrixToFile()
	+IntersectionsPerUnitArea()	+ConvertObjectToIndex()
	DFN_Analysis.CutPlane	
	+_init_()	
	+DrawPlane()	
	+TotalLengthOfFractures()	
	+NumberOfIntersectingFractures()	
	+FractureIntensity_P21()	
	+PlaneLines()	
	+IntersectionMatrix()	
	+Percolate()	
	+IntersectionsPerFracture()	

Table 2: Architecture for fracture network analysis

Modules for Fracture Network Regeneration and Post-processing	
DFN_ReGen	**DFN_PostProcessing**
+RedrawNetwork()	+IntersectionMatrixColorMap()
	+IntersectionHistogram()
	+IntersectionsPerFracturePlot()
Modules for DFN Input	
Input	**StatInput**
+ReadFile()	+ReadFile

Table 3: Architecture for DFN regeneration, postprocessing and input

3.1 Y-Frac Fracture Network Representation and Generation

Fracture Networks created by Y-Frac are stored as NURBS surfaces on Rhinoceros 6 interface. NURBS representation of fractures has been used in various works on fracture studies, suggesting its acceptance in geoscience [91,92]. NURBs surfaces are precise, quick to generate, smooth without sharp edges, and can be moved between various modelling, rendering, and animation programs [93,94]. They also require less parameters to generate them and their memory consumption is independent of resolution [92,95]. Individual fractures are stored as Globally Unique Identifiers (GUIDs) in python. GUIDs are identifiers of an object on the Rhinoceros interface. A GUID is a 128-bit value, represented by a string of 32 alphanumeric characters separated by hyphens in the form 8-4-4-4-12 [96]. GUIDs are beneficial in representing Rhinoceros objects because they are guaranteed to be unique every time they are generated. All fractures and in the network have separate layers and names displayed under the layers tab on the panels, labelled "6" in figure 1. Rhinoceros layers enable a user to make changes to all objects on a layer at once and keep track of the objects. For example, the display colour of all objects on a layer can be changed at once. Layers in Rhinoceros are also used by python to store a set of surfaces together. The setup of the DFNs ensures single fracture or fracture sets can be studied distinctly through identification by their layers and GUIDs.

Fracture networks are generated on Y-Frac by creating a domain and inserting a specified or random number of fractures. Domains are made by creating an instance of the Domain class. The class method *Show()* is then used to display the domain on Rhinoceros interface. Fractures are inserted into the created medium using any of these three functions in DFN_Gen module: *FixedFracureGen(), RandomFractureGen(),* and *SeparatedFractureGen().* Fracture parameters such as shape and size are specified in a text file and read using the input modules to provide arguments for these functions. Statistical distributions are also specified in a text file and serve as global variables for the functions. Fracture shapes available on Y-Frac include circle, ellipse and regular polygons. The basic DFN modelling approach involves using statistical distribution to describe cardinal variables such as spatial location, intensity, orientation and size [25]. A uniform distribution is used for fracture orientation, location, intensity, and fracture sizes.

Undoubtedly, some of the fractures in the network will extrude beyond the fractured medium. The domain class' method *RemoveSurfacesOutsideOfBox()* trims the out of bounds fractures. This method utilises *CreateBoundary()*, *GetSurfaceFromFractureLayer(),* *ConvertPloySurfaceToSurface()*, *CreateSetOfExtensionBoundaries(),* and *RemoveSurfacesIfAnyIntersectBoundaries()*.

The specific procedure taken by *RemoveSurfacesOutsideOfBox()* to trim extruding fractures is described in Algorithm 1.

Algorithm 1. RemoveSurfacesOutsideOfBox(): Trimming out of bound fractures

```
1:  Create a fracture domain boundary and store its identifier.
2:  Get and store all fracture identifiers into list k.
3:  Initialise a list p to store all fractures' identifiers post trimming.
4:  for all fractures in the list k:
        check if the fracture intersects the boundary and split it.
        append the identifier of the non-extruding part of the fracture in the list p.
        If the fracture does not intersect the boundary:
            append it to list p as well.
        end if
    end for
5:  Delete all old fractures using identifiers in list k.
```

```
6:  Display all new fractures using the identifiers in list p.
```

3.2 Methods for Fracture Network Analysis

The DFN_Analysis module holds most methods for fracture analysis and characterisation in the library developed. Its methods perform fracture analysis in both 3D on a DFN, and 2D on a cut-plane. The Class IntersectionAnalysis contains four methods for a 3D DFN analysis: *LengthOfIntersection(), FracturesSurfaceArea(), FractureIntensity_P32(),* and *IntersectionsPerUnitArea().* The CutPlane class draws the cut-plane using the *DrawPlane()* method to produce a 2D plane in a 3D domain. *TotalLengthOfFractures(), FractureIntensity_P21(). IntersectionMatrix(),* and *Percolate()* are other methods contained in the CutPlane class.

The Domain module also does fracture analysis with the functions. Its *IntersectionMatrix()* and *Percolate()* functions perform same roles as those of cut-plane, however in 3D for this case. The Matrix module is an auxiliary module containing the Matrix class which stores the matrix created by the intersection matrix function and its properties. It has three methods: *PrintMatrix()*, *MatrixToFile()* and *ConvertObjectToIndex()*.

IntersectionMatrix() creates a square matrix of fracture-fracture and fracture-boundary intersections for a fracture network and *Percolate()* uses the matrix created to determine if the domain percolates or not. These two methods are pivotal features of Y-Frac and contribute the most to the computational cost of fracture analysis using this software. The description of *Percolate()* and *IntersectionMatrix()* are contained in Algorithm 2 and 3 respectively.

Algorithm 2. Percolate(): Determine if two opposite boundary percolates

```
1:  Create an object list containing all fractures and boundaries identifiers.
2:  Create a list of all other boundaries other than the two we want to check percolation for.
3:  Get the index of the 1st boundary in the boundary list and its row number in the matrix
4:  Get the index of the 2nd boundary in the boundary list and its row number in the matrix
5:  Check if all the elements of both the 1st and 2nd boundaries rows are zero
      If this check is True:
        return False (No percolation)
      end if
6:  Initialise an index list with the 1st boundary's index, to store matrix indices as we check for
    percolation.
7:  Initialise a list, say track_list, with the 1st boundary's identifier, to store fracture identifiers
    and possibly the 2nd boundary's identifier as we check for percolation.
8:  Set a variable, say k, to zero. It moves through the columns of the matrix.
9:  Set a variable, say old_length, with the length of track_list. It keeps track of the length of the
    list in 7, after each phase.
10: Set a variable, say l, to zero. It is updated after each phase.
11: While True:
      for every column of the intersection matrix:
        if any column of the boundary is greater than 0, and the corresponding boundary's or
        fracture's identifier is not already in the lists created in 7 and 2 above.
          append the fracture's identifier in the list created in 7 and its index in the index list
        end if
      end for
    if l is equal to k (that is, all rows of "old_length" number of fractures have been visited)
        check the number of fractures added
        if no fracture or boundary 2 was not added:
            return False (No percolation)
        end if
        increment l and old_length by the number of fractures added
    end if
    if the second boundary's identifier is in track_list created in 7 above:
        return True (There is percolation)
    end if
    increment k by 1
```

3.3 Discrete Fracture Network Library functionalities

The core functionalities of this library are demonstrated in this section. The following examples illustrate some of the usages of this library for fracture generation, analysis and characterisation. The lines of code demonstrating the functionalities of Y-Frac in figures 3 to 8 are continuous but broken into snippets. We start by generating a fixed number of disc-shaped fractures in a domain.

```
# import necessary modules
import rhinoscriptsyntax as rs
import DFN_Gen
import Domain
import Input
import DFN_Gen
import Frac

#Avoid drawing whilst computing
rs.EnableRedraw(False)

#read fracture network data
data = "DataFile.txt"
radius, boxlength, n, fracture_shape = Input.ReadFile(data)

#create an instance of domain
dom = Domain.Domain(boxlength)

#draw domain
dom.Show()

#insert n fractures in the domain
frac_list = DFN_Gen.FixedFractureGen(n)
```

Figure 3: code snippet providing an example script which creates a fracture network

The code snippet starts by importing the required modules. We disabled redraw to prevent the drawing of objects on Rhinoceros interface while Rhinoceros is computing because drawing while computing adds to the computational cost of the python script. The fracture network data required are read in using the Input module, accessed through *ReadFile()*. An instance of the Domain class was created and named "dom". The domain was drawn on the Rhinoceros interface by *Show().* Finally, we inserted a specified number of disc-shaped fractures into the domain through the FixedFractureGen() function. The other arguments of the function are not needed for adding disc fractures. The fracture shape was specified as "circle" in the text file "DataFile". Fracture size, domain length, the number of fractures and fracture shape are specified in DataFile.txt. It is noteworthy that here the fractures were not trimmed, and it is possible of fractures may extrude the boundaries of the domain.

```
#trims fractures outside the domain
dom.RemoveSurfacesOutsideOfBox(dom.length)

#get the guids of new fractures in the domain after trimming
dom_frac = dom.my_fractures

# delete the old fractures
# change the guid of Fracture object to the new ones
new_frac_guids = Frac.NewFracturesGuids(dom_frac,frac_list)

#print number of fractures in the domain
Total_Fractures = dom.NumberOfFractures()
print(Total_Fractures)

#check if the first Fracture_1 intersects Fracture_8
frac_list[0].Intersect(frac_list[7])
```

Figure 4: Code snippet showing Y-Frac's functionality to trim out of bound fractures

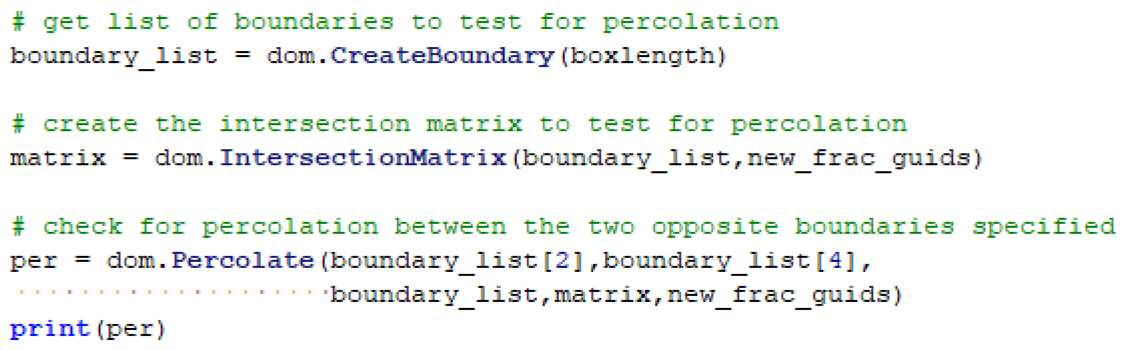
A further illustration of the library's functionality for fracture generation is shown in Figure 4. The next step after generating the fractures is to trim the extruding ones, performed by *RemoveSurfaceOutsideBox()*. New fracture objects, of which none goes outside the domain boundaries, are generated during trimming at the location of old fractures. Therefore, there is a need to delete the previous fractures and replace the GUIDs of our fracture objects in the Fracture class with the new GUIDs. This process is carried out by the *Frac.NewFracuturesGuids()* method. Figure 4 also shows how the total number of fractures generated can be printed to the console, which is very useful in the case of random fractures generation. It is also possible to check if a fracture intersects another using the *Intersect()* method.

3D intersection analysis can be done after generating the fracture network. Figure 5 shows a set of possible 3D analysis for a fracture network using this library.

```
import DFN_Analysis as da

# create an instance of IntersectionAnalysis class
ia = da.IntersectionAnalysis()

# find total length of intersection
l = ia.LengthOfIntersection(new_frac_guids)
print(l)

# determine fracture intensity P32
domain_width, domain_length, domain_height = boxlength, boxlength, boxlength
P_32 = ia.FractureIntensity_P32(new_frac_guids,domain_length,
                                domain_width, domain_height)
print(P_32)

# find total lengths of intersection per unit area
inter_per_unit_area = ia.IntersectionsPerUnitArea(new_frac_guids, domain_length,
                                                   domain_width, domain_height)
print(inter_per_unit_area)
```

Figure 5: Code snippet showing 3D analysis of fracture network using Y-Frac.

The process starts by importing the DFN_Analysis module and then creating an instance of the IntersectionAnalysis class. The sum of intersection lengths in the network was determined with the *LengthOfIntersection()* method. The P_{32} fracture intensity is also found using the *FractureIntensity_P32()* method, and *IntersectionsPerUnitArea()* gives the total lengths of intersection per unit area of the domain. One of the current assumptions of this library is that the domain is a cube, which is the reason why its width, length and height have the same value. With our fracture network adequately set up, it can be determined if the system percolates or not, as shown in Figure 6.



```
# get list of boundaries to test for percolation
boundary_list = dom.CreateBoundary(boxlength)

# create the intersection matrix to test for percolation
matrix = dom.IntersectionMatrix(boundary_list,new_frac_guids)

# check for percolation between the two opposite boundaries specified
per = dom.Percolate(boundary_list[2],boundary_list[4],
                    boundary_list,matrix,new_frac_guids)
print(per)
```

Figure 6: Code snippet demonstrating the percolation test using Y-Frac.

The check for percolation of a fracture network is of great importance in earth sciences. It is an essential part of studies for unconventional oil recovery, radioactive and waste disposal, underground water hydrology and underground CO_2 storage. A percolation test is done by first creating a list of boundaries GUIDs, the six sides of the domain. The Domain class method *IntersectionMatrix()* creates a square intersection matrix, as explained previously. Of course, the domain sides do not intersect one another. Consequently, the intersection values will be zero*. The Percolate()* method returns a bool indicating if the domain percolates or not. The Intersection Matrix is an innovative tool for performing percolation analysis. It reduces the computational cost drastically if we are check percolation between two other opposite sides of the domain other than the ones used previously. The cost reduction is evident since we only must check through the matrix, rather than performing the intersection tests between all fracture domain's sides once again.

Even though the network is a 3D fracture model, Analysis can also be performed on a 2D cut-plane. An example of this is shown in Figure 7.

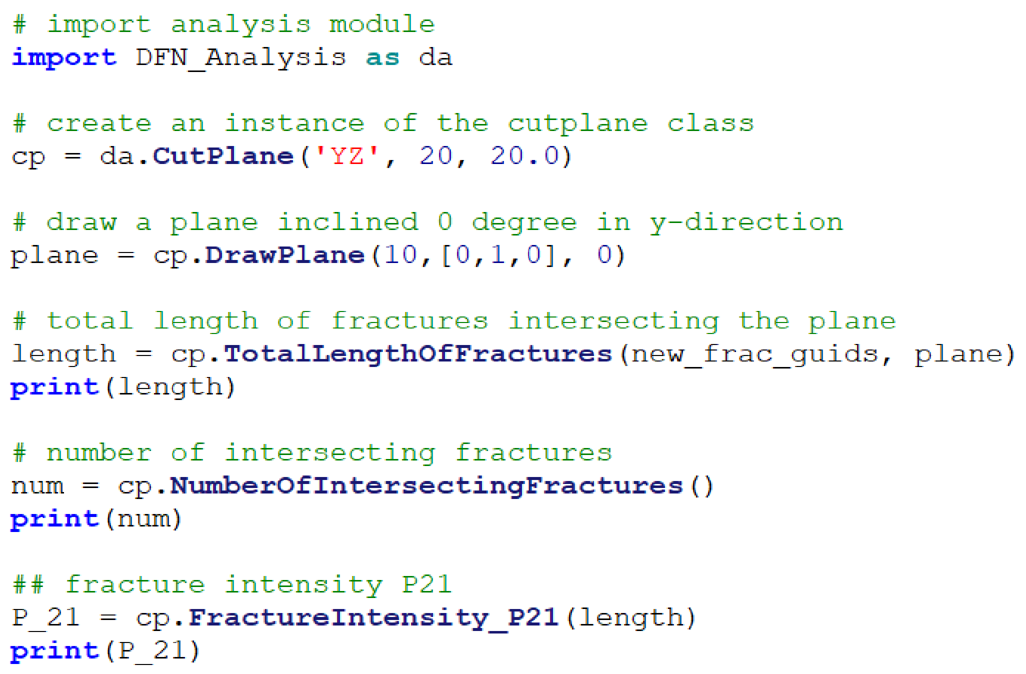
An instance of the CutPlane class is created, specifying the direction of the plane, in this case, 'YZ', and the width and height of the plane. DrawPlane() draws the plane, inclined at the fixed angle. The plane is stored as a surface on Rhinoceros, and its GUID is stored in python. The cut-plane is set up at this stage. Fracture analysis can then be carried out, with TotalLengthOfFractures() providing the total lengths of fracture intersecting the plane, NumberOfIntersectingFractures() returning the number of fractures that crossed the plane and FractureIntensity_P21() determines the P_{21} fracture intensity. Percolation analysis can be performed on the 2D cut-plane.

Steps like the 3D case are carried out to determine the percolation state of the cut plane in Figure 8. The GUIDs of the fractures intersecting the plane must be found, which the CutPlane class object "intersecting_fractures" detects. *PlaneLines()* method returns a list of the plane boundary GUIDs. The intersection matrix is then created, and percolation state is determined by the method *Percolate()*.

Another important ability of this library is to regenerate fracture networks. When a fracture is being generated, the library writes the necessary features required for fracture regeneration in a text file, which is used for fracture regeneration. This functionality is shown in Figure 9.

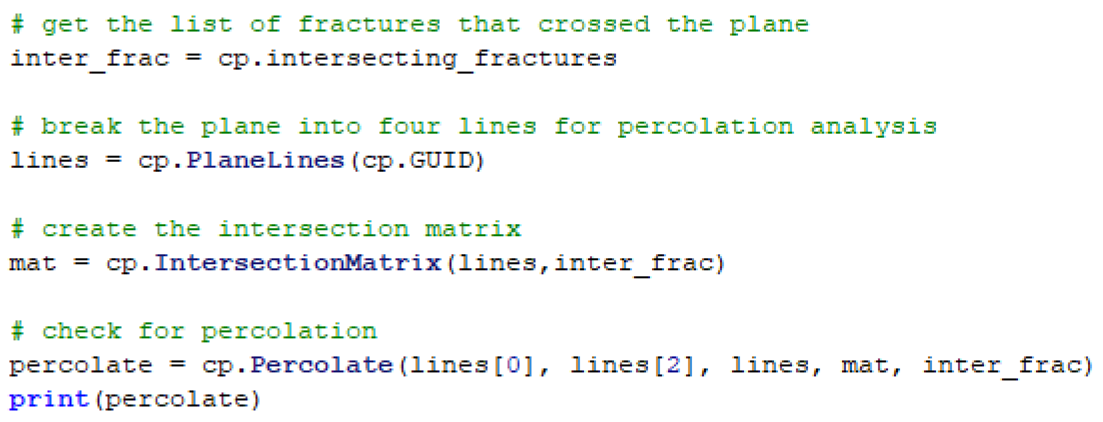
The steps involve importing the appropriate module, DFN_ReGen, specifying the path where the text file is saved and regenerating the network using the *RedrawNetwork()* function. Information exported into fracture_data.txt by Y-Frac includes fracture orientation, size, domain size, plane coordinates, number of sides of fracture polygons, and coordinates of fracture polygons. Details in this text file can serve as input variables for appropriate software packages to simulate flow and perform mechanical analysis in fracture networks.

Postprocessing is done outside the Rhinoceros environment because it does not support the matplotlib python library for visualisation. Therefore, the part of the script in Figure 10 after the dashed line must be run in a python IDE. The arguments are created by copying the results from the Rhinoceros console to the python IDE.



```
# import analysis module
import DFN_Analysis as da

# create an instance of the cutplane class
cp = da.CutPlane('YZ', 20, 20.0)

# draw a plane inclined 0 degree in y-direction
plane = cp.DrawPlane(10,[0,1,0], 0)

# total length of fractures intersecting the plane
length = cp.TotalLengthOfFractures(new_frac_guids, plane)
print(length)

# number of intersecting fractures
num = cp.NumberOfIntersectingFractures()
print(num)

## fracture intensity P21
P_21 = cp.FractureIntensity_P21(length)
print(P_21)
```

Figure 7: Code snippet demonstrating cut-plane analysis with Y-Frac



```
# get the list of fractures that crossed the plane
inter_frac = cp.intersecting_fractures

# break the plane into four lines for percolation analysis
lines = cp.PlaneLines(cp.GUID)

# create the intersection matrix
mat = cp.IntersectionMatrix(lines,inter_frac)

# check for percolation
percolate = cp.Percolate(lines[0], lines[2], lines, mat, inter_frac)
print(percolate)
```

Figure 8: Code snippet demonstrating further cut-plane analysis with Y-Frac

```
# import required module
import DFN_ReGen as dr

# specify path where the text file is saved
path = "~/Rhinoceros/6.0/scripts/text_files/fracture_data.txt"

# regenerate fracture network
dr.RedrawNetwork(path)
```

Figure 9: Code snippet demonstrating fracture regeneration with Y-Frac

```
# import postprocessing module
import DFN_PostProcessing as pp

# save the lengths of intersecting lines
l = ia.LengthOfIntersection(new_fracture_guids)

# get all lengths of intersection
lengths = ia.lengths_of_intersection_lines
print(lengths)

# get the number of intersections per fracture
inter_list = da.IntersectionsPerFfracture(mat)
print(inter_list)

------------------------------------------------------------------

# show intersection matrix colour map
# the result from the intersection matrix is used as the argument
pp.IntersectionMatrixColorMap(mat)

# plot the graph of intersections per fracture
pp.IntersectionsPerFracturePlot(inter_list)

# plot the histogram of lengths of intersection
pp.IntersectionHistogram(lengths)
```

Figure 10: Code snippet demonstrating fracture network postprocessing with Y-Frac

It is necessary to import the postprocessing module for these tasks. Instances of modules imported earlier were used. The three functions, *IntersectionHistogram(), IntersectionMatrixColorMap(), and IntersectionsPerFracturePlot()* use their respective arguments to display their plots.

4.0 DFN Library Case Study

This section contains the results and analyses of case studies evaluated in this work. A 20m x 20m x 20m domain was created, and 50 fractures of different shapes – disc, ellipse, polygon (square) – of size 4m were inserted. Figures 11-16 show the fracture network for these cases pre and post trimming of extruding fractures.

Fracture System	Total length of intersections (m)	Total length of intersections per unit area (m^{-1})	Fracture Intensity (P_{32}) (m^{-1})	Percolation state
Circular	370.888	0.0464	0.265	True
Elliptical	126.397	0.0158	0.133	False
Square	178.927	0.0224	0.173	False

Table 4: 3D analysis results for fracture networks.

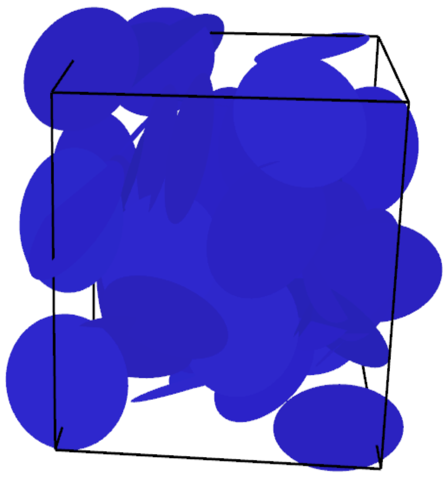


Figure 11: Fracture Network containing 50 circular fractures before trimming.

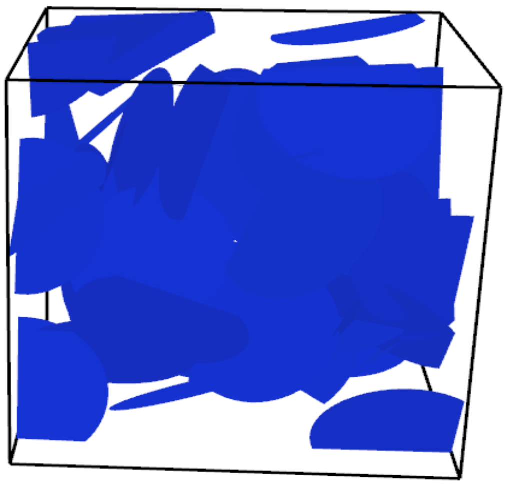


Figure 12: Fracture Network containing 50 circular fractures after trimming.

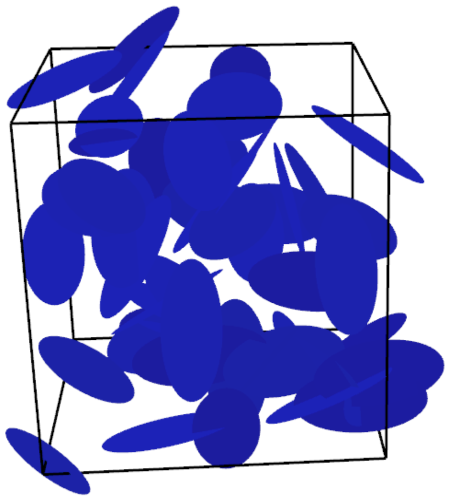


Figure 13: Fracture Network containing 50 elliptical fractures before trimming.

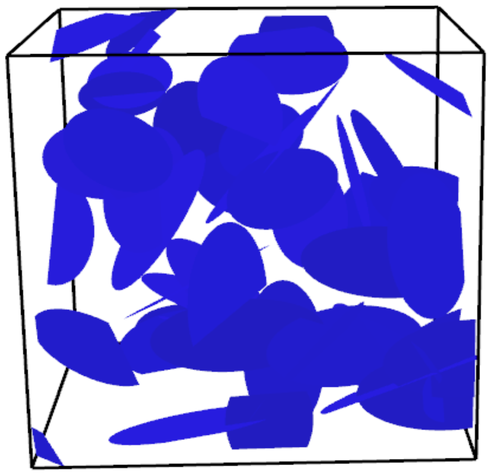


Figure 14: Fracture Network containing 50 elliptical fractures after trimming.

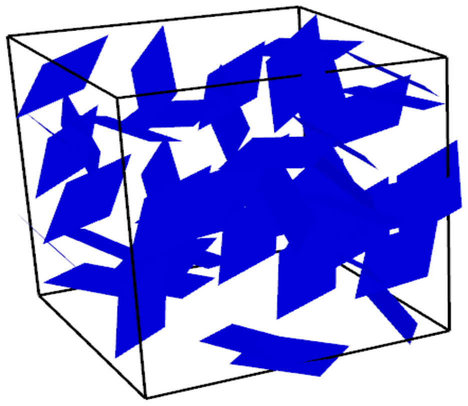


Figure 15: Fracture Network containing 50 square fractures before trimming.

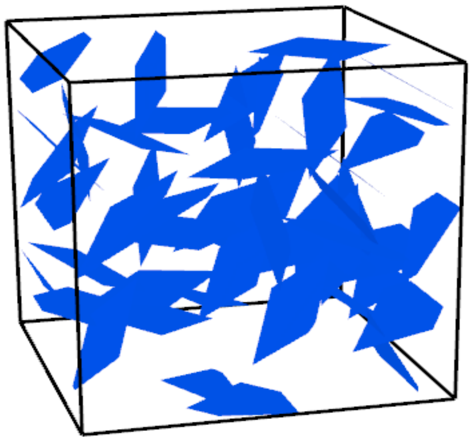


Figure 16: Fracture Network containing 50 square fractures after trimming.

The fracture networks were generated with the *DFN_Gen()* module without trimming. They were then regenerated and trimmed using the *DFN_ReGen()* module. The circular fractures expectedly occupy most space in the domain, due to having the largest surface area, followed by the square fractures then the elliptical fractures being the least. Further analyses were carried out on these three fracture systems, using the fracture regeneration module along with relevant modules and methods. The fractures were regenerated to show the possibility of analysis on pre-generated fracture networks. The code snippets for these could be found in the implementation folder on this library's GitHub page. The table below contains the 3D analyses conducted on these systems.

Circular fracture system has the highest numerical values for all the parameters considered as contained in table 4, while the elliptical system has the least. This trend further confirms the role the fractures' surface areas play on the lengths of intersection and fracture intensity. It is necessary to point out that only the circular fracture system percolates, specifically between top and bottom of the domain.

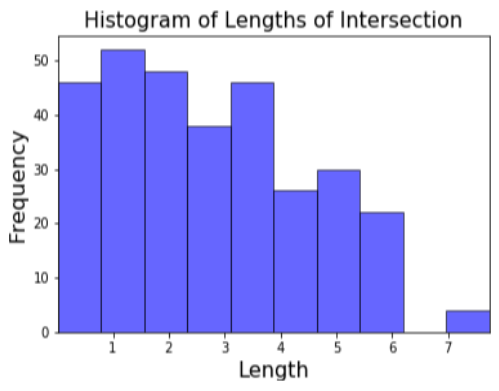
The results from the cut-plane analysis done on these systems are summarised in Table 5.

Fracture System	**Direction**	**Total lengths of intersections between fractures**	**Number of intersecting fractures**	**Fracture Intensity (P_{21}) (m^{-1})**	**Percolation state**
Circular	YZ	139.16	23	0.348	False
	XY	84.84	18	0.212	False
	ZX	97.53	16	0.244	False
Elliptical	YZ	63.43	18	0.159	False
	XY	38.75	10	0.097	False
	ZX	24.10	8	0.060	False
Square	YZ	46.19	12	0.115	False
	XY	63.41	16	0.159	False
	ZX	83.57	19	0.209	False

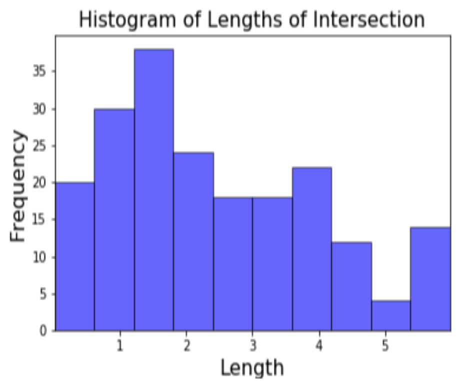
Table 5: Cut-plane 2D analysis results for fracture networks.

All three directions were considered in the 2D analysis. A 20m x 20m cut-plane was introduced at the centre of the domain in the three directions. Although this library permits for the inclination of planes in these directions, the cut planes for these analyses were not rotated. A critical look at the table above shows the trend in the 3D analysis is being followed in the cut-plane assessment of the systems. The circular network has the highest values for all the variables considered, and the elliptical system has the least. However, none of the cut-plane systems percolates. Lang et al. [97] studied the extent to which cut-planes percolate. They obtained a set of 75 cut-planes, each obtained in a random manner from 11 3D fracture network. They concluded that the dimensionless density ρ' at which the 3D networks percolate is linearly related to that of cut-planes by a multiple of 4. That is, $\rho'_{3D} = 4\rho'_{2D}$. Therefore, at low fracture densities, percolation may occur in 3D networks but not in any of the 2D cut-planes. Similar analysis is possible with Y-Frac, as a scenario has been demonstrated with this example.

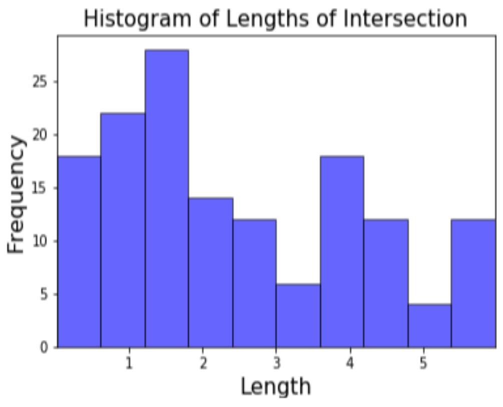
Postprocessing of these fracture network characteristics was done. Postprocessing aids in visualisation of the characteristics of fracture networks and gives some insights on this network, which could be tedious with numerical values only. Figures 17(a), (b) and (c) present the postprocessing visualisations for these systems.



(a)



(b)



(c)

Figures 17(a), (b) & (c): Histograms of intersection lengths for circular, elliptical and square fracture systems respectively.

The lengths of intersections for all the fracture networks ranges mostly between 0 – 3.5m. The disc radius used to generate the fractures in this system is 4m. The circular fracture system has the highest length of fracture intersection, 7.5m, which is expected since its fractures have the largest surface area. It can also be deduced that the probability of fracture overlapping is negligible.

The colour map plots in (see appendix 1) of the fracture networks' intersection matrix also confirm higher lengths of fracture-fracture intersection occur in the circular network. The boundaries have larger surface areas than the fractures, which leads to more significant surface-surface interaction; consequently, all networks have lengths of intersection >7m for fracture-boundary intersections.

Plots of the number of intersections per fracture (see appendix 2) made known isolated fractures - that is, fractures which do not intersect any other fracture - in all networks.

4.1 Computational Cost

The computational cost to generate different fracture shapes was investigated. Ten realisations of fracture systems containing 100 to 500 fractures were performed, and the timings averaged to determine the computational cost of each system on the Rhinoceros 6 interface. These tests were performed on a laptop computer with 4 cores, 1.7GHz clock rate, and 8GB RAM. This computer specification represents a typical environment for using Y-Frac rather than a high-performance computing platform. Figure 18 shows the result of this study.

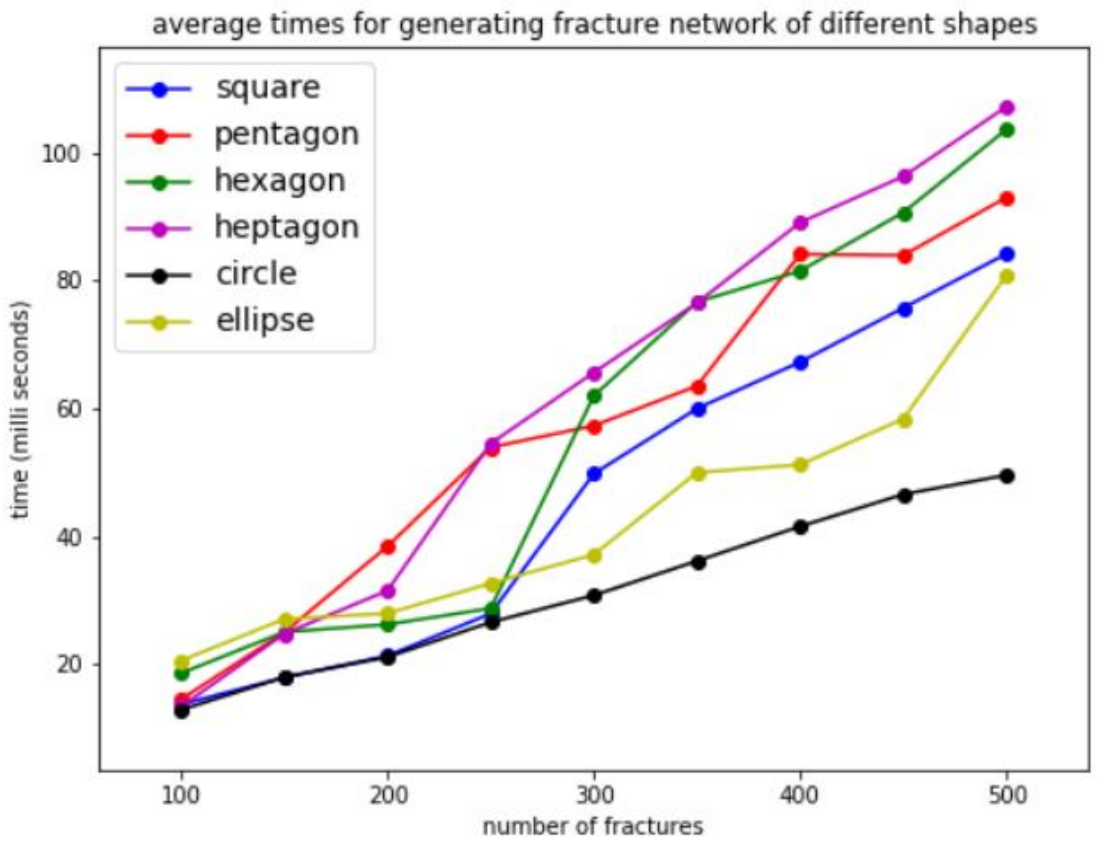


Figure 18: computational cost to generate fracture networks of different shapes.

The plot above indicates that the circular fractures cost least to generate, followed by the elliptical fractures. Additionally, it can be deduced that the higher the number of sides of the polygon, the greater the computational cost. The higher cost is due to the additional cost involved in generating and connecting more points as the number of sides increases. The most populated fracture system studied contains 500 fractures. The highest computational cost is less than 1 second. Overall, this analysis has shown that it is very cheap to generate fracture networks using Y-Frac. Hence, users do not need to worry about computational cost when using this library.

Another essential operation is the intersection analysis between fractures. The cost of preparing the intersection matrix for postprocessing and percolation analysis was also studied as shown in Figure 19. This operation is expected to be costlier than fracture generation due to the number of intersection operations. Here, ten realisations were carried out for each number of fractures, and the timings were averaged.

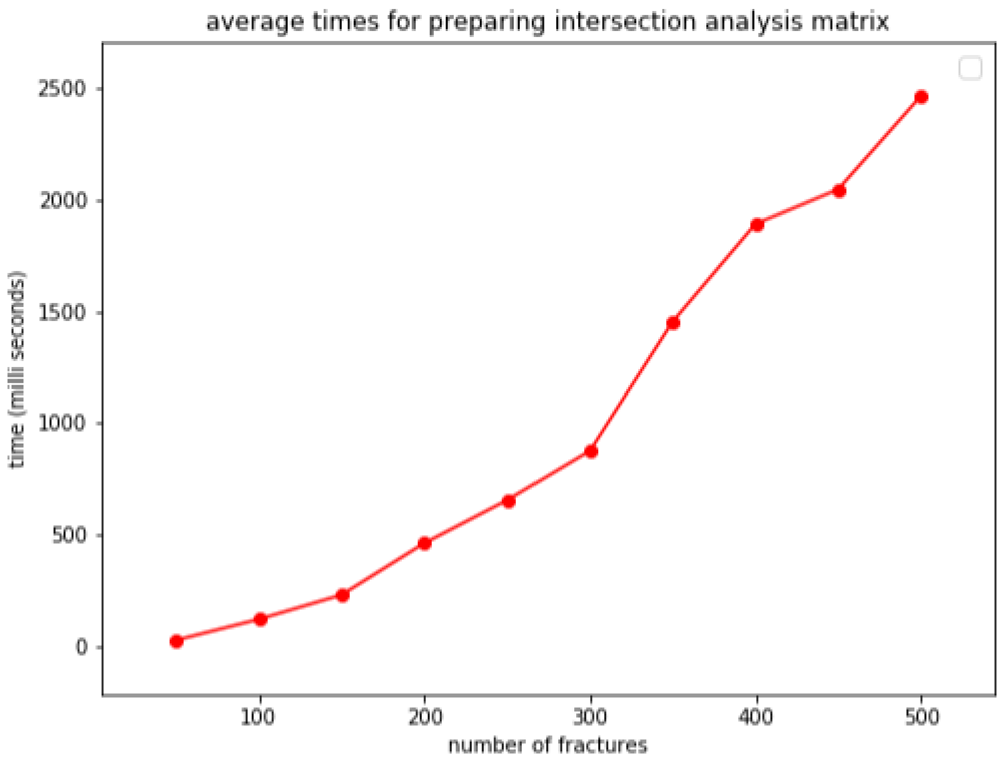


Figure 19: computational cost for preparing intersection matrix

The highest computational cost, for a network of 500 fractures, is about 2.5 seconds. This cost is relatively cheap for an operation involving 125,000 intersection operations. As previously mentioned, the intersection matrix is an innovative and very cheap way of checking for percolation between more than one opposite boundary for a fracture system. It is cheaper than the traditional method of determining percolation by performing intersection tests when considering two opposite boundaries of a domain.

4.2 Percolation Threshold Analysis

Huseby et al. [83] performed studies to determine percolation analysis for regular polygons in a 3D domain. Convex, identical fractures were considered. They are also assumed to be isotropically oriented and uniformly distributed. The fracture medium was considered periodic as described by Alder [98]. The finite polygons were embedded in a disk whose centres were uniformly distributed.

They performed their simulation, with 500 realisations, on seven types of equally sized polygons including triangles, squares, hexagons, octagons, 20-gons, rectangles with aspect ratio of 2, and uniform sized distributed rectangles of aspect ratio of 2. They obtained a range for the percolation threshold, the fracture dimensionless density above which the fracture percolates, as $\rho_c' = 2.26 \pm 0.04$. Basically, the percolation threshold is the value of the dimensionless density, ρ', at which the percolation probability is 0.5. Percolation probability is the ratio of the number of fracture network realisations that percolate to the total number of realisations.

Furthermore, Adler et al. [99] did a percolation study on simulated fracture networks in excavated damage zones and concluded that ρ_c' is in the range of 2.3 ± 0.1. Sisavath et al. [100] also conducted a study on percolation properties of fracture networks from line data and found ρ_c' to be approximately 2.254.

This library was used to perform simulation for determining the percolation threshold. Various sets of 250 realisations were done using a 20m x 20m x 20m fracture domain with a fracture disc size of 4m. The fractures were identical, uniformly distributed and oriented. Triangle, Square, hexagonal and octagonal shaped fractures were considered. The percolation threshold of range, $\rho_c' = 2.205 \pm 0.55$ was obtained. This result clearly is within the range of published result. This further demonstrates the applicability of this library for practical purposes. The summary of the results is contained in table 6. Figure 20 shows the plot of percolation probability against dimensionless density.

Shape	Triangle	Square	Hexagon	Octagon
Percolation threshold	2.32	2.26	2.37	2.37

Table 6: summary of percolation analysis results

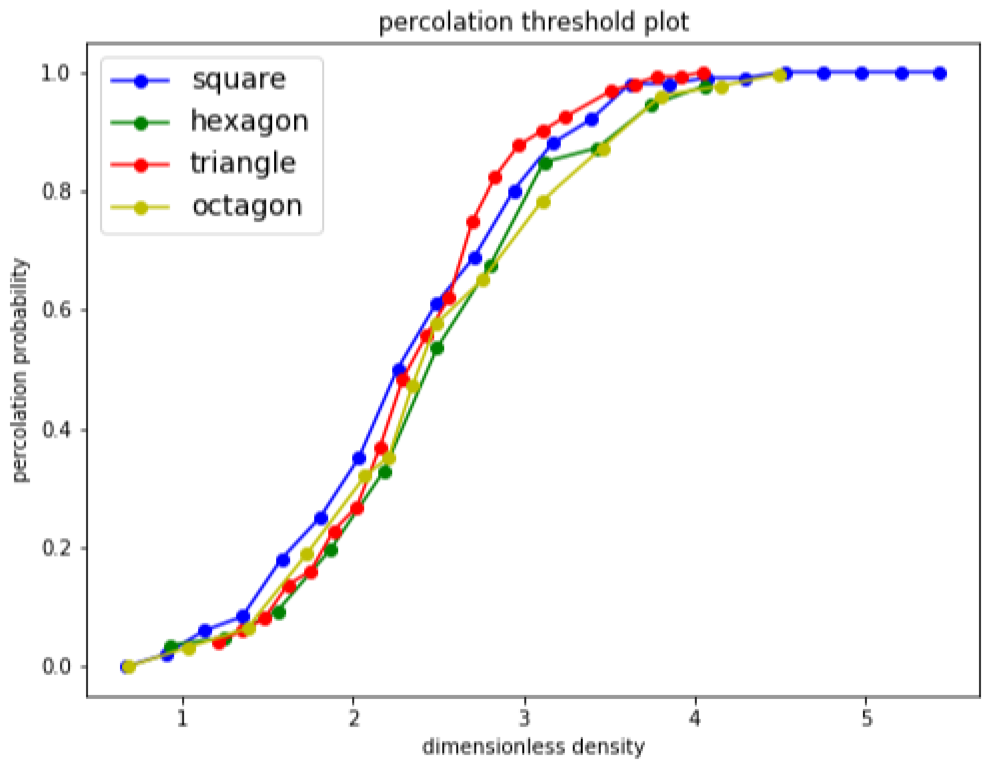


Figure 20: Plot for the percolation threshold analysis.

4.3 Testing

It was not feasible to do a continuous integration test, for instance, with Travis CI [101]. Travis CI is a continuous integration test service that runs tests on codes a user commits on GitHub. Various factors preclude this option. This work was based on Rhinoceros 6 python API. Hence, almost all the functions involve the use of Rhinoscryptsythax library, which is unknown to Travis and could not be imported. Also, being a library for geometrical modelling on Rhinoceros, most methods and functions return Rhinoceros GUIDs, which are unusual data type and are not recognised for pytest either. Additionally, the Rhinoceros python script environment does not allow for unit testing of functions. Even though results from various functions of this library are visibly seen on the Rhinoceros Interface, a python script that can be run on Rhinoceros, containing various code blocks which can be uncommented for testing essential functions and methods in this library has been provided on the GitHub page.

Algorithm 3. IntersectionMatrix(boundary_list, domain_fractures): Creating intersection matrix

```
1:  Initialise the matrix as a list.
2:  Get the number of fractures in the domain from the domain fractures' list.
3:  Calculate the number of rows and columns of the matrix.
4:  for each row of the matrix:
        append an empty list to the previous matrix list. It then becomes a list of lists
            for each column of the matrix:
                append zero as the matrix element
            end for
    end for
5:  for every fracture:
        for every fracture:
            if the fractures are not the same:
                check for intersection
                    if there is an intersection:
                        set the element of symmetric matrix indices as the intersection length (for 3D)
                        set the element of symmetric matrix indices as one (for 2D)
                    end if
            end if
        end for
    end for
6:  for every fracture:
        for every boundary:
            check for intersection
            if there is an intersection:
                set the element of the symmetric matrix indices as the intersection length (for 3D)
                set the element of symmetric matrix indices as one (for 2D)
            end if
        end for
    end for
```

5.0 Conclusion and Recommendation

This work has put forward an open-source, easy to use library for Discrete Fracture Network generation, regeneration and analysis for use in Rhinoceros 6 CAD environment. The requirements for this library - both hardware and software – have been stated in the code metadata. Instructions on how to install and use this library are contained in ReadMe file on the library's GitHub page. Rhinoceros was chosen for this work due to the opportunity to write python scripts exploiting Rhinoceros' python API. It is also easy to learn and work with, and relatively cost-effective both computationally and economically.

This library contains various modules (python file) for specific purposes. Modules were written for fracture input, fracture generation, fracture regeneration, fracture analysis and postprocessing. This library can generate fracture networks containing fractures of circular, elliptical and regular polygonal shapes. Fractures in this library are objects, with each knowing its name, GUID and location. The GUID are identifiers of objects which are stored in python and object's surfaces are stored in Rhinoceros. Fixed, random and separated (by a threshold) fractures are possible to model with this library. Being a fracture simulation tool, statistical distributions were employed to generate basic fracture parameters. Fracture location, size, intensity and orientation all follow a uniform distribution. Both 3D and cut-plane (2D) analysis in all directions can be done on fracture networks using this library. 3D fracture analysis and characterisation functionalities in this library include; total lengths of fracture intersection, fracture intensity (P_{32}), total intersection length per unit area and percolation state. Also, the number of intersecting fractures, total lengths of intersecting fractures, fracture intensity (P_{21}) and percolation state can be calculated in 2D on cut-planes. A fast technique of determining percolation state was introduced in this work using the intersection matrix. Fracture Networks can be saved and loaded using a text file, which is automatically populated when either fixed or random fractures are generated. Information contained in the output text file can serve as input for appropriate software packages to simulate flow and perform mechanical analysis in fracture networks. The third section of this work contains several code snippets showing how to use some of these functionalities.

The computation cost for fracture and intersection matrix generation was evaluated. The circular fractures cost the least to generate, and the polygonal fractures cost most. Furthermore, the higher the number of sides of a polygon, the higher its cost. Overall, it costs less than 200 milliseconds to generate 500 fractures of the shapes considered. The cost of intersection matrix generation studies shows that the most significant operation consisting of over 125,000 intersection operations costs about 2.5 seconds, which is quite fast. Therefore, this library is suitable for important and computationally intensive geoscience study such as percolation analysis.

Y-Frac was demonstrated for practical usage to confirm the percolation threshold of regular polygons in an isotropic network. Lastly, it is expected that students and researchers in geoscience will use this tool for practical research, exploiting the current functionalities of Y-Frac for DFN generation and analysis.

Suggested future work on this library includes determination of percolation in space as against boundaries, the inclusion of irregular polygons as fracture shapes, including more statistical distributions for fracture parameters, determination of percolation clusters, flexibility in domain shapes, permeability determination, integration of geomechanical properties for DFN simulation and fluid and heat flow in fracture networks.

References

1. Hernqvist L. Characterization of the Fracture System in Hard Rock for Tunnel Grouting. Chalmers University of Technology; 2009.

2. Gillespie PA, Howard CB, Walsh JJ, Watt J. Measurement and characterisation of spatial distributions of fractures. 1993;226:113–41.

3. Baek SH, Kim SS, Kwon JS, Um ES. Ground penetrating radar for fracture mapping in underground hazardous waste disposal sites: A case study from an underground research tunnel, South Korea. Journal of Applied Geophysics. 2017;141:24–33.

4. Lachassagne P, Wyns R, Dewandel B. The fracture permeability of Hard Rock Aquifers is due neither to tectonics, nor to unloading, but to weathering processes. Terra Nova. 2011;23(3):145–61.

5. Lacazette. Natural Fracture Types [Internet]. 2001 [cited 2019 Jun 13]. Available from: https://www.naturalfractures.com/1.1.1.htm

6. National Research Council. Rock Fractures and Fluid Flow: Contemporary Understanding and Applications. Washington, DC: National Academy Press; 1996. 568 p.

7. Dverstorp B, högskolan KT. Analyzing flow and transport in fractured rock using the discrete fracture network concept. Stockholm : Hydraulic Engineering, Royal Institute of Technology; 1991.

8. Brian B. Characterizing flow and transport in fractured geological media: A review. Advances in Water Resources [Internet]. 2002;25(2002):861–84. Available from: http://www.sciencedirect.com/science/article/pii/S0309170802000428

9. Sanderson DJ, Nixon CW. The use of topology in fracture network characterization. Journal of Structural Geology [Internet]. 2015;72:55–66. Available from: http://dx.doi.org/10.1016/j.jsg.2015.01.005

10. Bear J, Verruijt A. Theory and applications of transport in porous media. Modeling of groundwater flow and pollution, Dordrecht: Reidel. 1987.

11. Lei Q, Latham JP, Tsang CF. The use of discrete fracture networks for modelling coupled geomechanical and hydrological behaviour of fractured rocks. Computers and Geotechnics [Internet]. 2017;85:151–76. Available from: http://dx.doi.org/10.1016/j.compgeo.2016.12.024

12. Fadakar Alghalandis Y. ADFNE: Open source software for discrete fracture network engineering, two and three dimensional applications. Computers and Geosciences [Internet]. 2017;102(September 2016):1–11. Available from: http://dx.doi.org/10.1016/j.cageo.2017.02.002

13. Maillot J, Davy P, Goc R Le, Darcel C, de dreuzy LR. Connectivity, permeability, and channeling in randomly distributed and kinematically defined discrete fracture network models. Water Resources Research: 2016;52(11).

14. Alghalandis YF. Stochastic Modelling of Fractures in Rock Masses. PhD Thesis. University of Adelaide. 2014;(March).

15. Neuman SP. Trends, prospects and challenges in quantifying flow and transport through fractured rocks. Hydrogeology Journal. 2005;13(1):124–47.

16. Thomas RN. Permeability of fracture networks generated through geomechanical fracture-

growth simulations. PhD Thesis. Imperial College London. 2019;(March).

17. Midland Valley ltd. 2018 Brochure FractureModelling [Internet]. 2018 [cited 2019 Jun 24]. Available from: https://www.mve.com/media/2018_Brochure_FractureModelling.pdf

18. Wheeler AF. NAPSAC Technical Summary. 2016;(July). Available from: https://www.amecfw.com/documents/downloads/specialist-services/connectflow/dfn-technical-summary.pdf

19. Golder. FracMan Software [Internet]. 2019 [cited 2019 Jun 24]. Available from: https://www.golder.com/fracman/

20. MoFrac. Discrete Fracture Network Modelling Software [Internet]. 2019 [cited 2019 Jun 24]. Available from: http://www.mofrac.com/

21. Erhel J, Gander MJ, Halpern L, Pichot G, Griebel M. Methods in Science and Engineering XXI Editorial Board. 2014. 970 p.

22. Hyman JD, Karra S, Makedonska N, Gable CW, Painter SL, Viswanathan HS. DfnWorks: A discrete fracture network framework for modeling subsurface flow and transport. Computers and Geosciences [Internet]. 2015;84:10–9. Available from: http://dx.doi.org/10.1016/j.cageo.2015.08.001

23. Xing Zhang and Sanderson DJ. Numerical Modelling and Analysis of Fluid Flow and Deformation of Fractured Rock Masses. London, UK: Elsevier; 2002. 300 p.

24. Lisjak A, Grasselli G. A review of discrete modeling techniques for fracturing processes in discontinuous rock masses. Journal of Rock Mechanics and Geotechnical Engineering [Internet]. 2014;6(4):301–14. Available from: http://dx.doi.org/10.1016/j.jrmge.2013.12.007

25. Rogers S, Elmo D, Stead D, Guidelines for the quantitative description of discontinuities for use in DFN modeling. 2015;(May):1–11.

26. Long JCS, Remer JS, Wilson CR, Witherspoon PA. Porous media equivalents for networks of discontinuous fractures. Water Resources Research. 1982;18(3):645–58.

27. Long JCS, Gilmour P, Witherspoon PA. A Model for Steady Fluid Flow in Random Three-Dimensional Networks of Disc-Shaped Fractures. Water Resources Research. 1985;21(8):1105–15.

28. Endo HK, Long JCS, Wilson CR, Witherspoon PA. A Model for Investigating Mechanical Transport. Water Resources Research. 1984;20(10):1390–400.

29. Endo HK. Mechanical transport in two-dimensional networks of fractures. PhD Thesis. University of California, Berkeley; 1984.

30. Balberg I, Binenbaum N, Wagner N. Percolation thresholds in the three-dimensional sticks system. Physical Review Letters. 1984;52(17):1465–8.

31. Andersson J, Dverstorp B. Conditional simulations of fluid flow in three-dimensional networks of discrete fractures. Water Resources Research. 1987;23(10):1876–86.

32. Palleske, C., Lato MJ, Hutchinson DJ, Elmo D, Diederichs MS. Impacts of limited spacing and persistence data on DFN modelling of rockmasses. Proceedings of GeoMontreal. Montreal; 2013.

33. Vazaios I. Factors affecting realism of DFNs for Mechanical Stability Analysis in Tunneling Environment. International Discrete Fracture Network Engineering Conference. 2014;(July

2016).

34. Mauldon M. Estimating mean fracture trace length and density from observations in convex windows. Rock Mechanics and Rock Engineering. 1998;31(4):201–16.

35. Min KB, Jing L, Stephansson O. Determining the equivalent permeability tensor for fractured rock masses using a stochastic REV approach: Method and application to the field data from Sellafield, UK. Hydrogeology Journal. 2004;12(5):497–510.

36. Rogers S, Elmo D, Webb G, Catalan A. Volumetric Fracture Intensity Measurement for Improved Rock Mass Characterisation and Fragmentation Assessment in Block Caving Operations. Rock Mechanics and Rock Engineering. 2014;48(2):633–49.

37. Elmo D, Rogers S, Stead D, Eberhardt E. Discrete Fracture Network approach to characterise rock mass fragmentation and implications for geomechanical upscaling. Mining Technology. 2014;123(3):149–61.

38. Amec Foster Wheeler. Modelling of Discrete Fracture Networks (DFN) [internet]. 2019 [Cited 2019 July 16] Availabe from: https://www.amecfw.com/services/specialist-services/connectflow/connectflow-dfn.

39. Pichot G, Poirriez B, Erhel J, De Dreuzy J-R. A mortar BDD method for solving flow in stochastic discrete fracture networks. In: Proceedings of the 21st international conference in Domain decomposition methods in science and engineering XXI. Inria Rennes Center, France; 2012.

40. Pencheva G, Yotov I. Balancing domain decomposition for mortar mixed finite element methods. Numerical Linear Algebra with Applications. 2003;10(1–2):159–80.

41. Dryja M, Proskurowski W. On preconditioners for mortar discretization of elliptic problems. Numerical Linear Algebra with Applications. In: Numerical Libear Algebra with Applications 2003;10(1–2):65–82.

42. Hokr JBM. Mixed-Hybrid Formulation of Multidimensional Fracture Flow. In: Numerical Methods and Applications, Lecture Notes in Computer Science. 2011. p. 125–32.

43. Březina J, Exner P. Fast algorithms for intersection of non-matching grids using Plücker coordinates. Computers and Mathematics with Applications. 2017;74(1):174–87.

44. Březina J, Stebel J. Analysis of model error for a continuum-fracture model of porous media flow. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2016;9611:152–60.

45. Exner P, Březina J. Partition of unity methods for approximation of point water sources in porous media. Applied Mathematics and Computation. 2016;273:21–32.

46. Brezina J. Mortar-like mixed-hybrid methods for elliptic problems on complex geometries. Proceedings of ALGORITMY. 2012;200–8.

47. Šístek J, Březina J, Sousedík B. BDDC for mixed-hybrid formulation of flow in porous media with combined mesh dimensions. Numerical Linear Algebra with Applications. 2015;22(6):903–29.

48. Xu C, Dowd PA, Mardia KV, Fowell RJ. Parametric pointintensity estimation for stochastic fracture modeling. LUMA(LeedsUniversityMining Association) Journal. 2003;16:85–93.

49. Xu C, Dowd PA, Mardia KV, Fowell RJ. A connectivity index for discrete fracture networks. Mathematical Geology. 2006;38(5):611–34.

50. Dowd PA, Martin JA, Xu C, Fowell RJ, Mardia K V. A three-dimensional fracture network data set for a block of granite. International Journal of Rock Mechanics and Mining Sciences. 2009;46(5):811–8.

51. Mardia KV, Nyirongo VB, Walder AN, Xu C, Dowd PA, Fowell RJ, Kent J. Markov Chain Monte Carlo implementation of rock fracture modeling. Mathematical Geology. 2007;39:355–81.

52. Xu C, Dowd PA. A new computer code for discrete fracture network modelling. Computers and Geosciences. Computers and Geosciences [Internet]. 2010;36(3):292–301. Available from: http://dx.doi.org/10.1016/j.cageo.2009.05.012

53. Painter SL, Gable CW, Kelkar S. Pathline tracing on fully unstructured control-volume grids. Computational Geosciences. 2012;16(4):1125–34.

54. Nejati M, Paluszny A, Zimmerman RW. On the use of quarter-point tetrahedral finite elements in linear elastic fracture mechanics. Engineering Fracture Mechanics [Internet]. 2015;144:194–221. Available from: http://dx.doi.org/10.1016/j.engfracmech.2015.06.055

55. Nejati M, Paluszny A, Zimmerman RW. A disk-shaped domain integral method for the computation of stress intensity factors using tetrahedral meshes. International Journal of Solids and Structures [Internet]. 2015;69–70:230–51. Available from: http://dx.doi.org/10.1016/j.ijsolstr.2015.05.026

56. Nejati M, Paluszny A, Zimmerman RW. A finite element framework for modeling internal frictional contact in three-dimensional fractured media using unstructured tetrahedral meshes. Computer Methods in Applied Mechanics and Engineering [Internet]. 2016;306:123–50. Available from: http://dx.doi.org/10.1016/j.cma.2016.03.028

57. Paluszny A, Matthai SK. Impact of fracture development on the effective permeability of porous rocks as determined by 2-D discrete fracture growth modeling. Journal of Geophysical Research. 2010;115(B2).

58. Paluszny A, Matthäi SK. Numerical modeling of discrete multi-crack growth applied to pattern formation in geological brittle media. International Journal of Solids and Structures [Internet]. 2009;46(18–19):3383–97. Available from: http://dx.doi.org/10.1016/j.ijsolstr.2009.05.007

59. Salimzadeh S, Usui T, Paluszny A, Zimmerman RW. Finite element simulations of interactions between multiple hydraulic fractures in a poroelastic rock. International Journal of Rock Mechanics and Mining Sciences. 2017;99(March):9–20.

60. Salimzadeh S, Paluszny A, Zimmerman RW. Three-dimensional poroelastic effects during hydraulic fracturing in permeable rocks. International Journal of Solids and Structures. 2017;108:153–63.

61. Thomas RN, Paluszny A, Zimmerman RW. Quantification of Fracture Interaction Using Stress Intensity Factor Variation Maps. Journal of Geophysical Research: Solid Earth. 2017;122(10):7698–717.

62. Mikell P. Groover EWZ. CAD/CAM: Computer-Aided Design and Manufacturing. Prentice-Hall; 1984. 489 p.

63. Tian F, Tian X, Geng J, Li Z, Zhang Z. A hybrid interactive feature recognition method based on lightweight model. 2010 International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2010. 2010;1:113–7.

64. Autodesk. AutoCad [Internet]. 2019 [cited 2019 Aug 7]. Available from: http://usa.autodesk.com/autocad/

65. Dasault Systems. Solid Works. [Internet]. 2019 [cited 2019 Aug 7]. Available from: https://www.solidworks.com/

66. Autodesk. Inventor Overview. [Internet]. 2019 [cited 2019 Aug 7]. Available from: https://www.autodesk.com/products/inventor/overview

67. PTC. Creo Parametric 3D Modeling Software [Internet]. 2019 [cited 2019 Aug 7]. Available from: https://www.ptc.com/en/products/cad/creo/parametric

68. Mcneel R. Rhinoceros 5 - Training Manual Level 1. 2013.

69. Autodesk. 3ds-max Overview [Internet]. 2019 [cited 2019 Aug 7]. Available from: https://www.autodesk.co.uk/products/3ds-max/overview

70. De Weck O. Lecture 4: Computer Aided Design (CAD). 16810: Engineering Design and Rapid Prototyping. 2005.

71. Associates MN and. Welcome to Rhino [Internet]. [cited 2019 Aug 7]. Available from: http://docs.mcneel.com/rhino/6/help/en-us/index.htm

72. Bingol OR, Krishnamurthy A. NURBS-Python: An open-source object-oriented NURBS modeling framework in Python. SoftwareX [Internet]. 2019;9:85–94. Available from: https://doi.org/10.1016/j.softx.2018.12.005

73. Mcneel R. Rhinoceros 5 User ’ s Guide Table of Contents Section I : Introduction. 2014.

74. Gorjanc S, Jurkin E. Introducing 3D Modeling into Geometry Education at Technical Colleges. The Visual Language of Technique. 2015;57–67.

75. Nassery F. 3d Models of Regular Polyhedrons in: Rhinoceros 3d, Autocad, 3ds Max – Possible Applications in the Teaching of Engineering Graphics. The Journal of Polish Society for Geometry and Engineering Graphic. 2015;27:37–44.

76. Dershowitz WS, Associates G, Herda HH. Interpretation of fracture spacing and intensity. Rock Mechanics. 1992;757–66.

77. Dershowitz, W. S ; Einstein HH. Characterizing rock joint geometry with joint system models. Rock Mechanics and Rock Engineering 21. 1988;(1):21–51.

78. Mauldon M. Intersection probabilities of impersistent joints. International Journal of Rock Mechanics and Mining Sciences and. 1994;31(2):107–15.

79. Ebigbo A, Lang PS, Paluszny A, Zimmerman RW. Inclusion-Based Effective Medium Models for the Permeability of a 3D Fractured Rock Mass. Transport in Porous Media. 2016;113(1):137–58.

80. Nixon CW, Sanderson DJ, Bull JM. Analysis of a strike-slip fault network using high resolution multibeam bathymetry, offshore NW Devon U.K. Tectonophysics [Internet]. 2012;541–543:69–80. Available from: http://dx.doi.org/10.1016/j.tecto.2012.03.021

81. Einstein HH, Locsin J-LZ. Modeling Rock Fracture Intersections and Application to the Boston Area. Journal of Geotechnical and Geoenvironmental Engineering. 2012;138(11):1415–21.

82. Adler TM, Thovert J-F, Mourzenko VV. Percolation of three-dimensional fracture networks with power-law size distribution. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics. 2005;72(3):1–14.

83. Huseby O, Thovert J-F, Adler PM. Geometry and topology of fracture systems. Journal of Physics A: Mathematical and General. 1997;3–11.

84. Dietrich S, Amnon A. Introduction To Percolation Theory. Revised Se. CRC press; 1994. 192 p.

85. Pierre M, Adler J-FT, Mourzenko VV. Frcatured Porous Media. Oxford: Oxford University Press; 2013. 175 p.

86. Thovert JF, Mourzenko V V., Adler PM. Percolation in three-dimensional fracture networks for arbitrary size and shape distributions. Physical Review E. 2017;95(4):1–14.

87. Alghalandis YF, Xu C, Dowd PA. A general framework for fracture intersection analysis: algorithms and practical applications. Australian Geothermal Energy Conference 2011. 2011;

88. Straub SC and Ben. Git [Internet]. 2019 [cited 2019 Aug 28]. Available from: https://git-scm.com/

89. Tutorialspoint. V-Model (software development) [Internet]. 2019 [cited 2019 Aug 26]. Available from: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm

90. Oxagile. Waterfall Software Development Model [Internet]. 2014 [cited 2018 Aug 26]. Available from: https://www.oxagile.com/article/the-waterfall-model/

91. Luiz F. Martha, Paul A. Wawrzynek ARI. Arbitrary crack representation using solid modeling. Engineering with Computers. 1993;9(2):63–82.

92. Paluszny A, Zimmerman RW. Numerical fracture growth modeling using smooth surface geometric deformation. Engineering Fracture Mechanics [Internet]. 2013;108:19–36. Available from: http://dx.doi.org/10.1016/j.engfracmech.2013.04.012

93. Digicams S. Understanding NURBS in Computer Graphics [Internet]. 2019 [cited 2019 Aug 28]. Available from: http://www.steves-digicams.com/knowledge-center/how-tos/video-software/understanding-nurbs-in-computer-graphics.html#b

94. Vision R. Advantages of using NURBS in organic modeling and reverse engineering [Internet]. 2019 [cited 2019 Aug 28]. Available from: https://rangevision.com/en/application/examples/revers-inzhiniring-i-kontrol-geometrii/advantages-of-using-nurbs-in-organic-modeling-and-reverse-engineering/

95. McNeel Wiki. NURBS surfaces [Internet]. 2019 [cited 2019 Aug 28]. Available from: https://wiki.mcneel.com/rhino/nurbs

96. Microsoft. GUID [Internet]. 2006 [cited 2019 Aug 28]. Available from: https://docs.microsoft.com/en-us/previous-versions/aa373931(v=vs.80)

97. Lang PS, Paluszny A, Zimmerman RW. Permeability tensor of three-dimensional fractured porous rock and a comparison to trace map predictions. Journal of Geophysical Research: Solid Earth. 2014;6288–307.

98. Adler PM. Porous media geometry and transports. Boston, USA: Butterworth-Heinemann; 1992. p. 551.

99. Mourzenko V V., Thovert JF, Adler PM. Percolation and permeability of fracture networks in excavated damaged zones. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics. 2012;86(2):1–13.

100. Sisavath S, Mourzenko V, Genthon P, Thovert J-F, Adler PM. Geometry, percolation and transport properties of fracture networks derived from line data. Geophysical Journal. 1965;157(2):917–34.

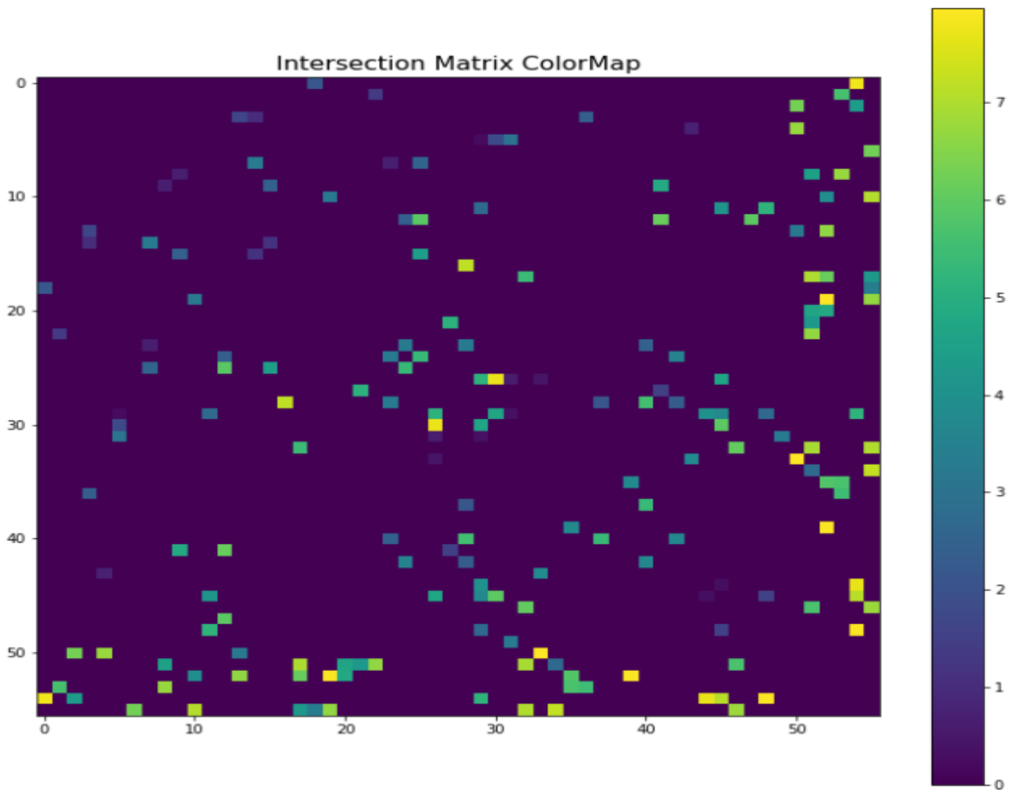
101. Travis CI. Core Concepts for Beginners [Internet]. 2019. p. August 27 2019. Available from:

https://docs.travis-ci.com/user/for-beginners/

Appendices

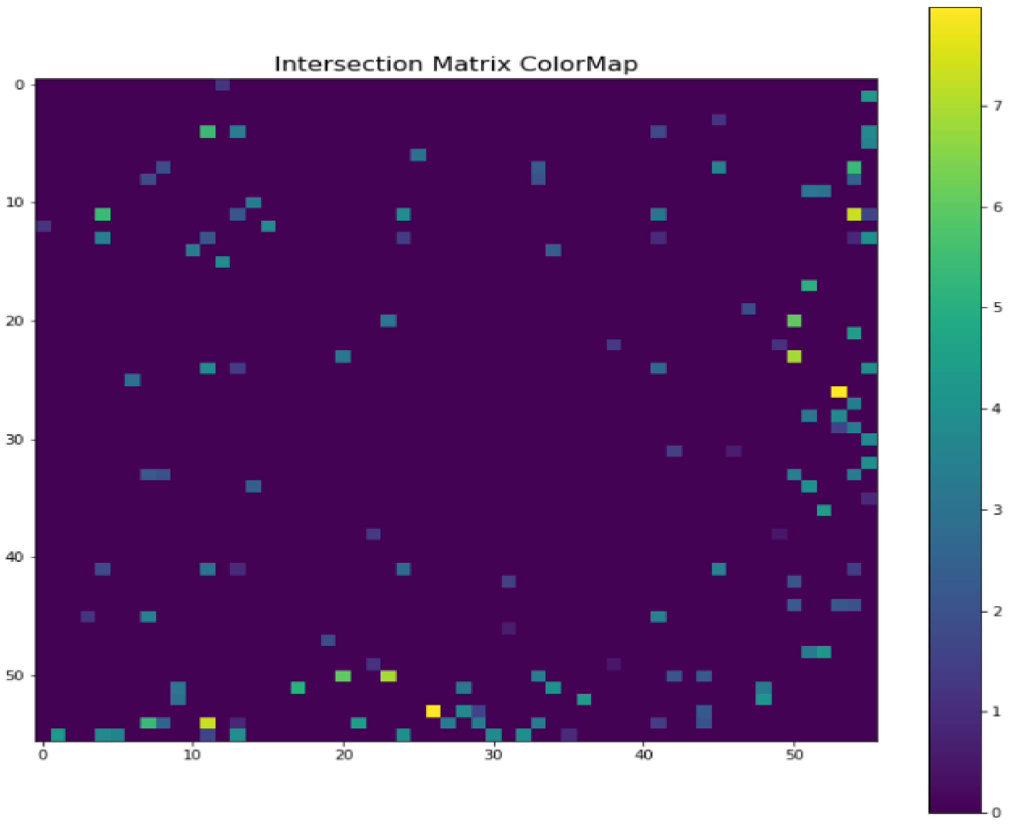
Appendix 1

Colour map visualisation of intersection matrix for circular, elliptical and square fracture networks respectively.

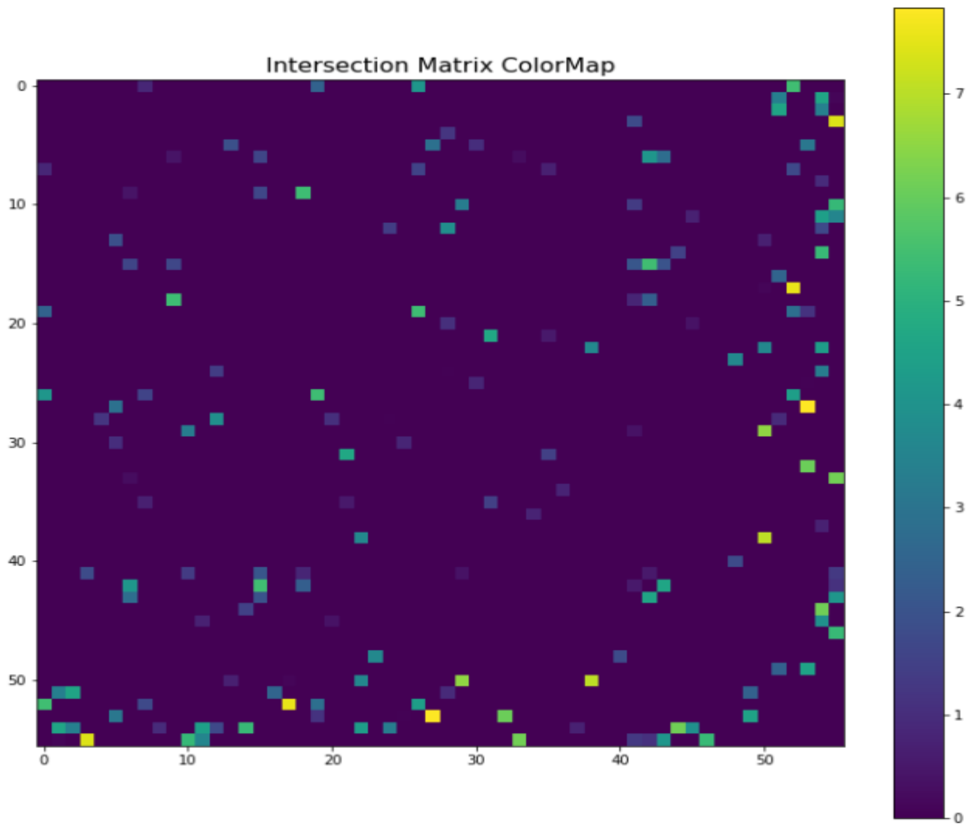


(b)

(a)


Intersection Matrix ColorMap

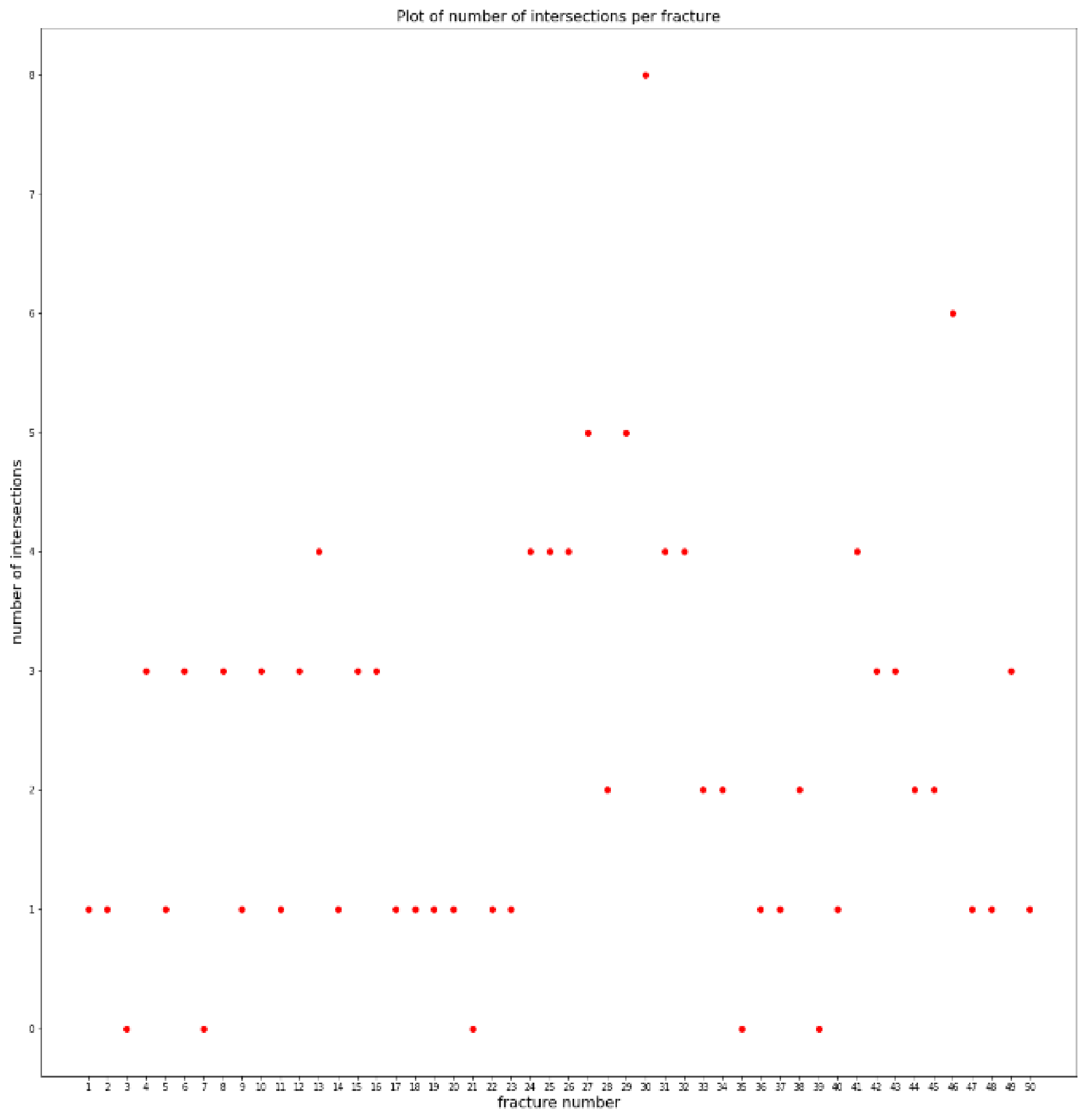
(b)


Intersection Matrix ColorMap

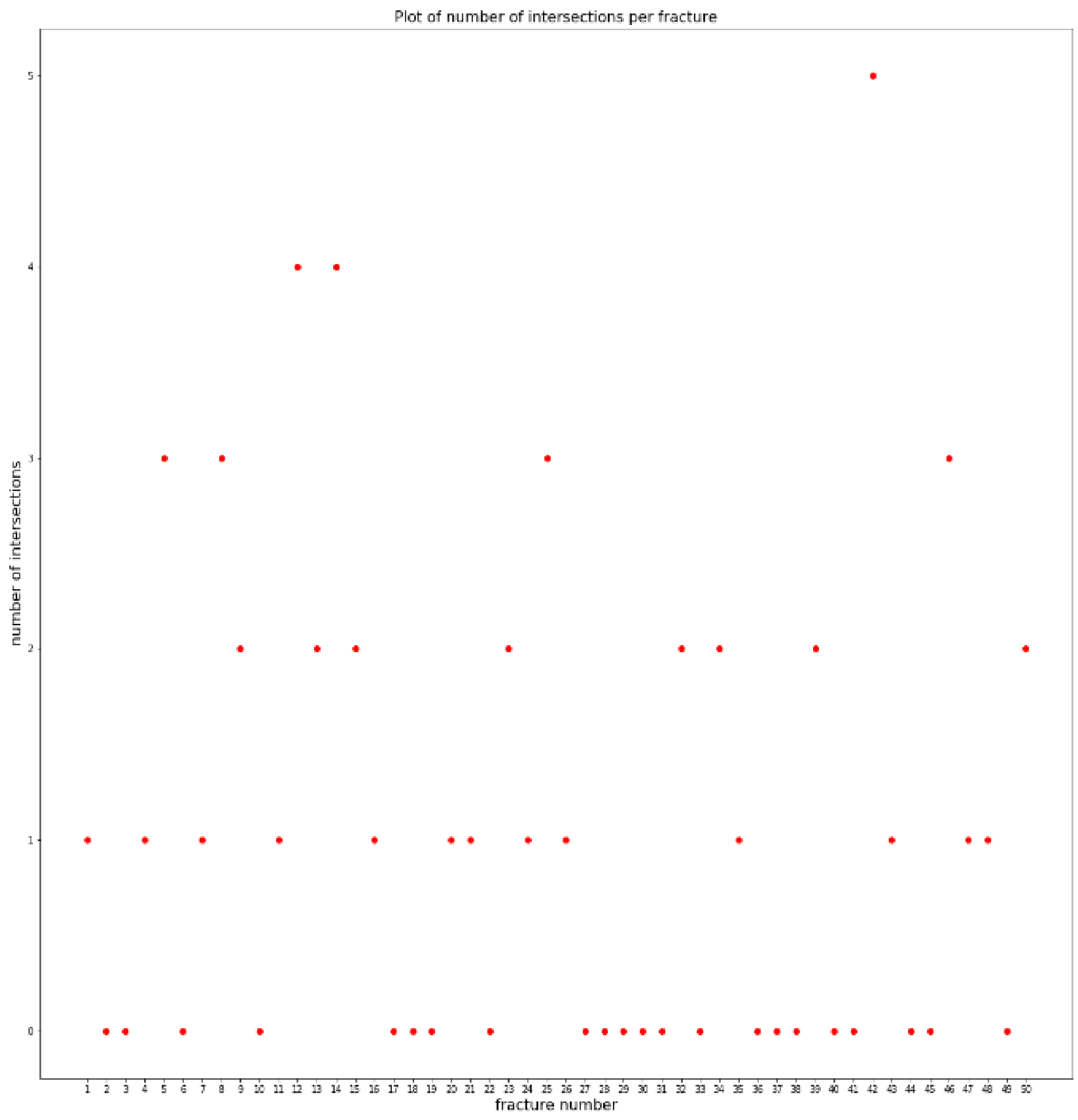
(c)

Appendix 2

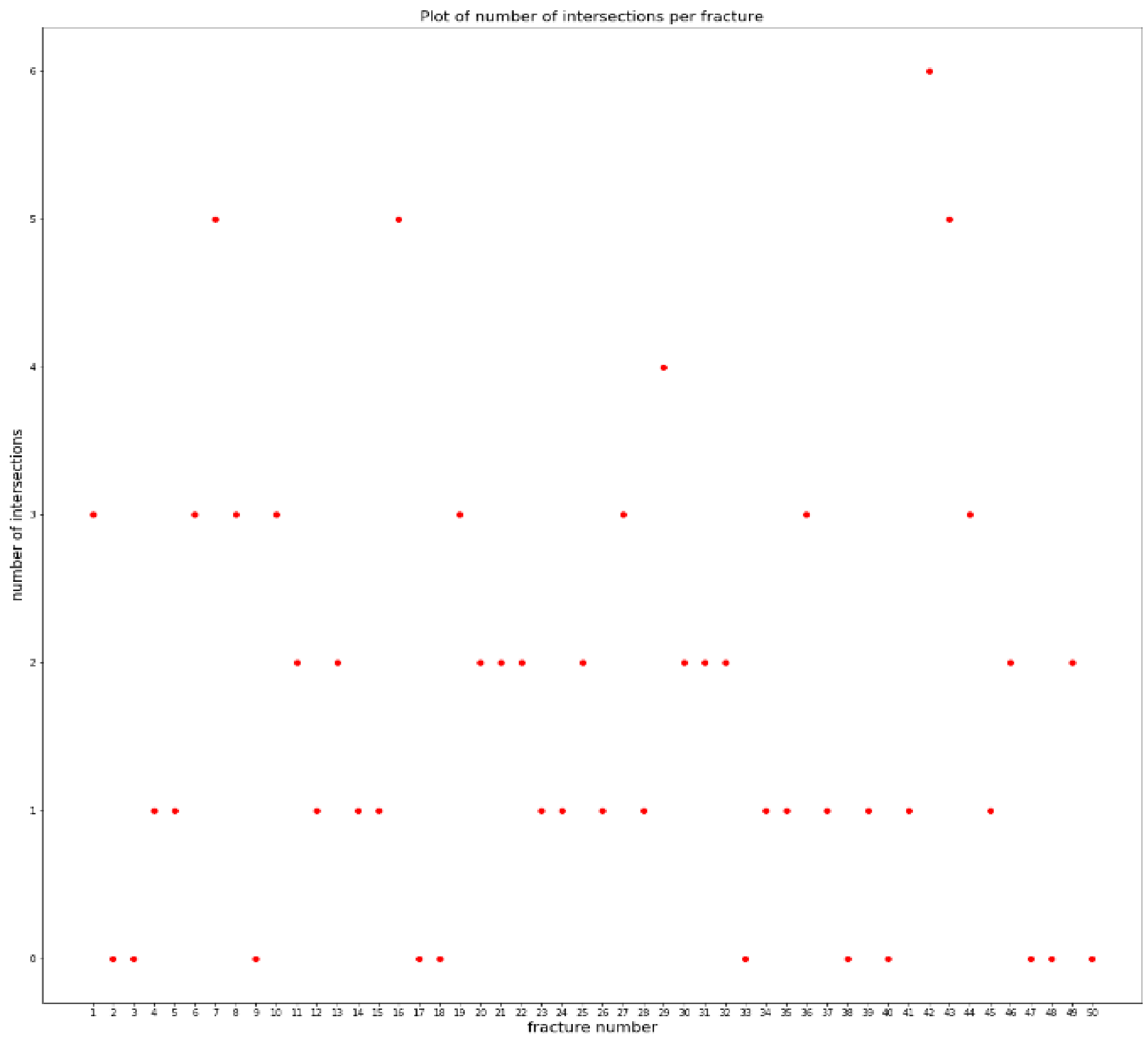
Plots of the number of intersections per fracture for circular, elliptical and square fracture networks respectively.



(a)


Plot of number of intersections per fracture
number of intersections
fracture number

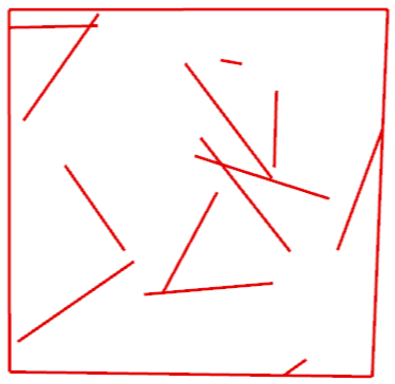
(b)



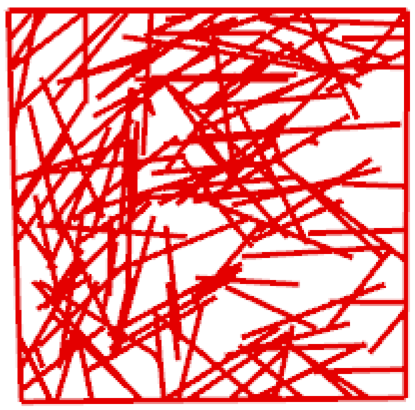
(c)

Appendix 3

Examples of 2D cut-planes in YZ direction for 50 and 500 fractures respectively.



50 fractures



500 fractures