UnBlocks-gen: A Python library for 3D rock mass generation and analysis

Article · July 2020

DOI: 10.1016/j.softx.2020.100577

Leandro Lima Rasmussen
University of Brasília
14 PUBLICATIONS **20** CITATIONS

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

UnBlocks^{gen}: A Python library for 3D rock mass generation and analysis

Leandro Lima Rasmussen

University of Brasilia, Civil and Environmental Engineering Department, Brasilia-DF, Brazil

ARTICLE INFO

Article history:
Received 1 June 2020
Received in revised form 12 July 2020
Accepted 30 July 2020

Keywords:
Rock block
Shape
Size
Generation
DFN

ABSTRACT

In this paper, a Python library named *UnBlocks^{gen}* is presented for the generation and analysis of 3D rock block systems. The library provides the tools for the construction of Discrete Fracture Networks; the generation of rock blocks based on the constructed DFN; and the analysis of blocks' geometrical characteristics, such as shape and size. An illustrative example is presented based on the Brazilian Monte Seco tunnel. It shows the library capabilities in dealing with complex Discrete Fracture Networks as well as performing excavation through a selected region of the block system. The library shall be valuable to research that attempts to understand the relations between rock masses' geometrical characteristics and behavior during engineering works.

Code metadata

Current code version	v 1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2020_237
Code Ocean compute capsule	none
Legal Code License	GPL v3
Code versioning system used	git
Software code languages, tools, and services used	C++ and Python 3.
Compilation requirements, operating environments & dependencies	Tested in Ubuntu 18.04 Bionic Beaver and Ubuntu 20.04 Focal Fossa. Required packages from Ubuntu package archive: python3, python3-numpy, python3-matplotlib, libboost-python-dev, coinor-clp and coinor-libclp-dev. Additional packages in Ubuntu 20.04: libboost1.67-dev and libboost-python1.67-dev.
If available Link to developer documentation/manual	https://github.com/LeandroLimaRa/UnBlocks-gen/blob/master/doc/Manual.pdf
Support email for questions	leandro.lima@unb.br

1. Introduction

Rock engineering deals with human interventions in discontinuous media, consisting of rock material separated by fractures. For robust engineering design, the strength, deformability and permeability behavior of such complex material have to be considered. Due to the stochastic nature of the discontinuity system of a rock mass, rock blocks usually present a variety of different shapes and sizes, determined by the number of fracture sets and their orientation, persistence and spacing values. It is the combined geometrical properties of rock blocks and the inter-block shear strength that defines a rock mass' mechanical behavior under a certain in-situ stress condition [1]. Therefore, rock blocks' shape and size characterization is a fundamental step in an engineering analysis of a fractured rock mass.

Blocks' shape and size have, for example, been shown to be directly related to the stability conditions and optimum reinforcement design of both surface and underground excavations [2,3]. The geometrical aspects of rock blocks are also important for the mining industry, when determining the cavability of an ore body or defining the required dimensions for mining facilities (e.g. ore passes) [4]. It should also be pointed that block size is an important parameter for rock mass classification systems, being either explicitly or implicitly represented [5].

Although simple estimates of rock blocks' shape and size can be made directly from field data, obtained by means of outcrop mapping and core logging, the computational representation of block systems is yet necessary for comprehensive analyses of rock engineering works. For this reason, the development of computer

E-mail address: leandro.lima.ra@gmail.com.

https://doi.org/10.1016/j.softx.2020.100577

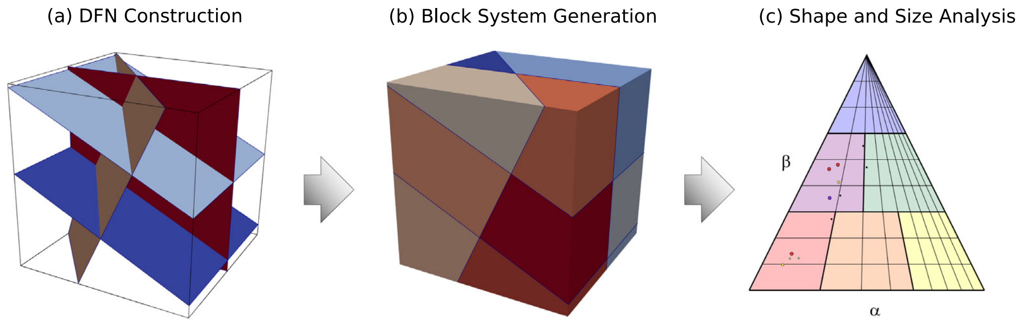

Fig. 1. The steps in the application of the *UnBlocks*^{gen} library.

codes for the generation and analysis of 3D rock mass models is justifiable.

Computer programs have been made to provide for 3D rock mass models. Fracman [6], Siromodel [7] and 3DEC [8] are examples of commercial software packages that can construct DFN, generate rock block systems and perform block stability analysis. In addition to commercial codes, research-oriented programs capable of generating 3D block systems from DFN data are also mentioned in the rock mechanics literature: BLOCKS [9], Block Generation Language [10], MSB Code [11], DC Code [12], RESOBLOK [13], GeoSMA-3D [14], GeneralBlock [15], 3D Block Generation Program [16], Rock Block Cutting program [17] and General Block [18] are examples of programs developed by the academia. Nonetheless, all these codes have two disadvantages: they do not provide for a block shape and size analysis based on Kalenchuk et al. [19] sound approach and they are either commercial (proprietary) or restricted to certain research groups. Of note, an open-source code for constructing DFNs, generating rock block systems and performing blocks' shape and size analysis has not been made available up to this moment.

In this work, the open-source *UnBlocks*^{gen} Python library is presented, which was created for performing computational modeling and analysis of 3D rock masses. It has been written in C++ and compiled with wrappers that provide Python users with direct access to its main classes. The library usage is based on the following simple three step process:

1. Discrete Fracture Network (DFN) construction.
2. Rock block system generation.
3. Blocks' shape and size analysis.

Fig. 1 provides an illustration regarding the three steps above. A DFN is a computational model which represents the geometrical properties of the discontinuities of a rock mass [20], usually based on a calibrated stochastic process capable of representing the inherent variability present in fractures' geometrical characteristics such as position, orientation, persistence and intensity. Once ready, a DFN consists in a set of planar structures positioned in space. In order to perform analysis however, it is necessary to convert these elements into a 3D block system, which possesses the geometrical information regarding each rock block, such as vertices, edges and facets data. When generated, the block system can then be used for different purposes, including shape and size analysis.

2. Software description

UnBlocks^{gen} library works based on a three step process: DFN construction; rock block system generation; and blocks' shape and size analysis. Each step is now thoroughly detailed.

Table 1
Poisson disk model assumptions.

Geometrical Attribute	Geometry, Model or Probabilistic Distribution
Shape	Circular
Location	Poisson process (i.e. random in space)
Persistence (fracture radius)	Exponential or Log-Normal Distribution
Orientation	Von Mises–Fisher Distribution
Intensity (P-System)	One of the following values for each fracture set: P10: Number of fractures per unit length P20: Number of fractures per unit area P21: Length of fractures per unit area P30: Number of fractures per unit volume P32: Area of fractures per unit volume

2.1. Discrete fracture network construction

The DFN construction is either deterministic, with the user defining all geometrical characteristics of a fracture system, or stochastic, with geometrical characteristics following predefined probabilistic distributions.

UnBlocks^{gen} library constructs DFNs based on the assumptions of the Poisson disk model, also known as Baecher disk model [21]. The assumptions regarding fracture shape, location, persistence, orientation and intensity value are shown in Table 1.

In order to calculate fracture intensity following the Dershowitz and Herda's P-System [22], either a line, surface or volume mapping region is defined. Priest [23] asserts that realistic representations of fracture networks can be achieved based on the assumptions presented in Table 1 while providing computational simplicity.

2.2. Rock block system generation

Block system generation is performed based on the sequential rock slicing procedure proposed by Boon et al. [24]. In this approach, each convex rock block is defined by a set of planes and the region they occupy in space is represented by a set of linear inequalities of the form:

$$\mathbf{a}_i^T\mathbf{x} \leq d_i, \quad i = 1, \ldots, N \tag{1}$$

where $\mathbf{a}_i$ is the unit normal vector of the plane number i; d_i is the distance of the plane number i to the origin of the coordinate system; $\mathbf{x}$ is the position vector; and N is the total number of planes.

Following a similar approach, each convex discontinuity is defined by a plane equation and a set of linear inequalities representing its boundaries:

$$\begin{aligned} &\mathbf{a}_{disc}^T\mathbf{x} = d_{disc} \\ &\mathbf{a}_{j,bound}^T\mathbf{x} \leq d_{j,bound}, \quad j = 1, \ldots, M \end{aligned} \tag{2}$$

where $\mathbf{a}_{disc}$ is the unit normal vector of the discontinuity plane; d_{disc} is the distance of the discontinuity plane to the origin of the coordinate system; $\mathbf{a}_{j,bound}$ is the unit normal vector of the plane number j defining a border of the discontinuity; $d_{j,bound}$ is the distance of the plane number j defining a border of the discontinuity to the origin of the coordinate system; $\mathbf{x}$ is the position vector; and M is the total number of planes defining the borders of the discontinuity. *UnBlocks^{gen}* uses 26 planes to approximately represent circular discontinuities as default, however this value can be changed by the user.

For the fact that blocks and discontinuities are both represented by sets of linear equalities and inequalities, it is then possible to make use o linear programming to verify whether a discontinuity intersects a block. The optimization problem is defined as:

$$\begin{aligned}
&\textit{minimize } s \\
&\mathbf{a}_i^T\mathbf{x} - d_i \leq s, \quad i = 1, \dots, N \\
&\mathbf{a}_{disc}^T\mathbf{x} - d_{disc} = 0 \\
&\mathbf{a}_{j,bound}^T\mathbf{x} - d_{j,bound} \leq s, \quad j = 1, \dots, M
\end{aligned} \tag{3}$$

A discontinuity intersects a block if $s < 0$. In case an intersection occurs, the intersected block is split by generating a copy of itself and adding the discontinuity plane equation to the set of plane equations representing both the new and the original block, however with opposite unit normal vectors. Of note, a block is fully sliced whenever a fracture intersects it.

Based on the approach aforementioned, the rock block system is generated sequentially based on a DFN data, starting with the first fracture of the first fracture set and concluding with the last fracture of the last fracture set. In the beginning of the process, it is assumed that the entire model region is composed of a single rock block. Once the slicing process is concluded, the blocks' geometrical information (i.e. vertices, edges and facets) are generated from the plane equations.

For further details on the rock block generation process, the interested reader is referred to the work of Boon et al. [24].

2.3. Blocks' shape and size analysis

Once the rock block system has been generated, *UnBlocks^{gen}* provides for the shape and size analysis of the formed blocks. For each rock block, the following values are calculated: α and β parameters, volume, inscribed sphere radius, aspect ratio and order. These are further explained in whats follows.

The α and β parameters were proposed by Kalenchuk et al. [19] to characterize the flatness and elongation of a rock block respectively. Both values range between 1 and 10 and the shape of a block can be inferred by the values' plotting position on the block shape diagram proposed by these same authors. Fig. 2 shows the diagram and the different block shapes represented: "C" for Cubic; "E" for Elongated; "P" for Platy; and "CE", "EP" and "PC" for the intermediate shapes between.

The α and β parameters are calculated based on the following equations:

$$\alpha = \frac{A_s l_{avg}}{7.7V} \tag{4}$$

where V is the block volume; A_s the surface area; and l_{avg} the average chord length.

$$\beta = 10\left[\frac{\sum(\mathbf{a}\cdot\mathbf{b})^2}{\sum\|\mathbf{a}\|^2\|\mathbf{b}\|^2}\right]^2 \tag{5}$$

where $\mathbf{a}$ and $\mathbf{b}$ are the different chords of the rock block with length longer than the chords' median length value.

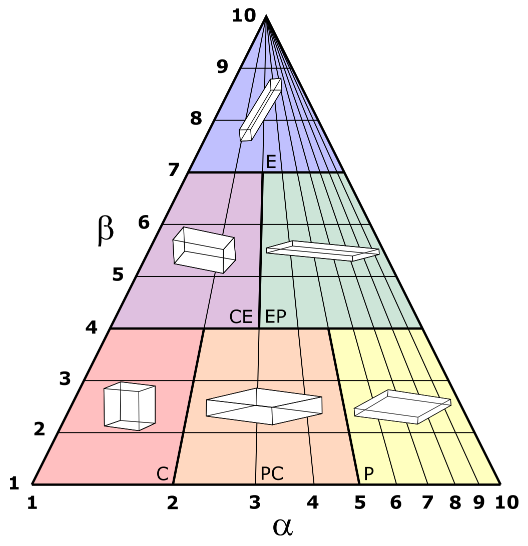

Fig. 2. Block shape diagram proposed by Kalenchuk et al.
Source: Modified from [19].

Inscribed sphere radius is the radius value of the biggest sphere that can be fitted within a rock block geometry; aspect ratio is another parameter that reflects the elongation of a block and is calculated as the ratio of a block's bounding sphere radius to its inscribed sphere radius; and order refers to the number of facets a block presents. *UnBlocks^{gen}* uses linear programming to calculate the bounding and inscribed sphere radius of a block following the scheme proposed by Boon et al. [24].

In addition to the block shape diagram previously presented, *UnBlocks^{gen}* library also generates a block volume and block shape distribution based on the blocks' volume, α and β values. All plots are produced using functions coded in a Python script, provided with the library. These plots have been originally suggested by Kalenchuk et al. [19] to further assist in the blocks' shape and size analysis process.

2.4. Software architecture

While the Discrete Fracture Network construction and block system generation procedures have been implemented in C++, the blocks' shape and size analysis has been coded in Python 3.

The DFN construction code is comprised of six classes: DFN, FractureSet, Fracture, LineMapping, SurfaceMapping and VolumeMapping. DFN is the main class, which contains vector containers for objects derived from the remaining classes, except Fracture class. It provides functions in order to add new fracture sets as well as line, surface and volume mappings. The FractureSet class contains the vector container for Fracture objects and is responsible for providing the functions to generate new fractures. The mapping classes calculate fracture intensities based on the P-system.

The block system generation code consists of three classes: Generator, Block and ExcavationElement. Generator is the main class, which provides the functions to convert a DFN object into a collection of Block objects that are stored in a vector container. The Block class contains functions for defining the vertices, edges and facets of a block. The ExcavationElement class provides fictitious planes which assist in the excavation of the formed block system.

Fig. 3 shows a simplified UML diagram for the DFN construction and block system generation C++ codes. In this diagram, only the public methods are being shown for each class. From

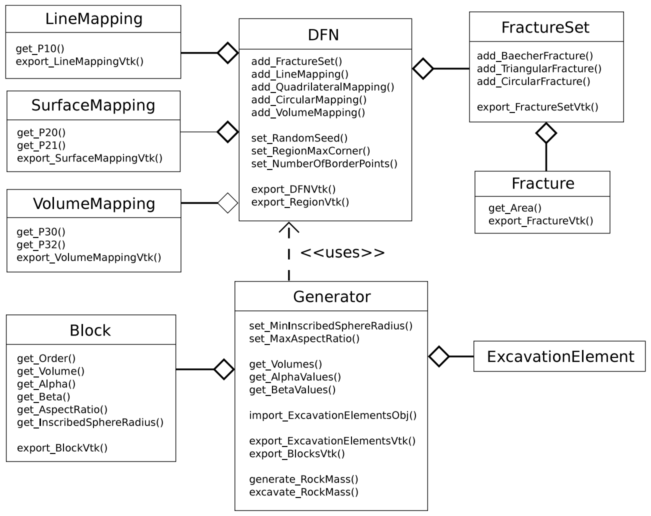

Fig. 3. Simplified UML diagram for the DFN construction and block system generation C++ codes.

the figure, the relations between the different classes and the code's data flow can be understood. In short, a DFN object is instantiated within the Python environment and its methods are used to construct either a deterministic or stochastic DFN. Afterwards, a Generator object is instantiated and its variables configured. Once ready, the Generator object generates the rock block system based on the previously created DFN object by means of its "generate_RockMass" method. Once the rock block system has been generated, functions provided by a Python script named "plotTools" are used to produce the blocks' shape and size analysis plots.

Auxiliary classes are also present in the C++ code, which provide assistance to the DFN and Generator ones. The following geometrical entities are organized as separated classes: Plane, Polygon, Edge, Triangle and Line. A Functions class, consisting of static analytical geometry functions, is available to provide for geometrical calculations, such as intersection tests.

It is relevant to point that the code makes use of Eigen library [25] for vector algebra, Coin-or CLP [26] for the application of the simplex method, Matplotlib [27] for plots generation, and Boost [28] for statistical functions as well as for wrapping parts of the C++ code into the Python 3 library.

2.5. Software functionalities

The DFN is constructed within a rectangular prism region with width, length and height defined by the user. Any number of fracture sets can be included in a DFN. Fracture intensity of each set can be calculated by means of line, surface or volume mappings. A line mapping provides for P10 calculation; a surface mapping is used to calculate P20 and P21 values; and a volume mapping is used to calculate P30 and P32 values. A surface mapping may be either a quadrilateral or a circular area. In the current version of the library, whenever a volume mapping is created, it is automatically set to encompass the whole model extent.

The library permits the inclusion of circular fractures with orientation defined by the dip direction and dip angle values, as obtained in the field by means of geological compass. The convention assumed is that dip direction defines a clockwise angular rotation from the *X*-axis direction, considered to be the model's north direction. Dip angle refers to the angle formed between the horizontal plane (i.e. X and Y axes plane) and the discontinuity.

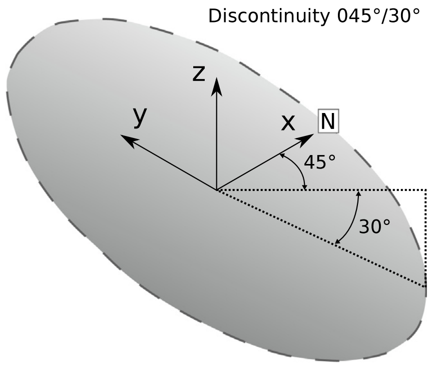

Fig. 4. Circular discontinuity with dip direction equal to 45° and dip angle equal to 30° illustrating the adopted convention.

Fig. 4 provides an example of a circular discontinuity with dip direction equal to 45° and dip angle equal to 30°.

Deterministic DFNs are also possible and the user may add either triangular or circular fractures. Once constructed, the complete DFN or selected fracture sets can be exported to a VTK file. With this file, it is then possible to perform visualization in Paraview software [29] or import its content to other programs. In the file, the fractures are represented as polygons following the legacy VTK file format [30].

Once a DFN has been constructed, the library offers the tools for converting the DFN into a collection of rock blocks (i.e. polyhedron elements with associated geometrical structure: vertices, edges and facets data). It is also possible to perform virtual excavations in the model, such as a tunnel or an open pit. This is achieved by importing a watertight triangular mesh representing the excavation zone. The mesh's triangles are used as fictitious planes during the sequential rock slicing procedure. It should be observed that, in the final result, rock blocks presenting the same ID value are to be considered as a single unit.

The 3D block system can be exported to a VTK file for visualization in Paraview. For the fact that the VTK file is written in ASCII format, it is a simple task to extract the blocks' geometrical information. In this way, the VTK file can possibly be used as input to other programs which use such type of information, for example, to perform numerical or limit equilibrium analysis. In the file, rock blocks are represented as polyhedrons according to the legacy VTK file format [31].

3. Illustrative example

To illustrate the application of the *UnBlocks^{gen}* library, the DFN construction and block system generation of a rock mass surrounding a tunnel excavation is presented. This example is based on the Monte Seco Tunnel, excavated in the state of Espírito Santo in Brazil.

The tunnel is 5.4 m wide and 6 m high, with a horseshoe-shaped profile and located approximately 11 m below rock–soil interface. It was excavated through gneiss containing three discontinuity sets: a foliation one and two joint sets. Rasmussen et al. [32] presented the calibrated parameters for the construction of representative stochastic DFN for a section of the tunnel, which are repeated in Table 2.

Based on information provided by Table 2, the stochastic DFN can be constructed using *UnBlocks^{gen}* by means of the Python commands below. First, the library and plotting tools are imported and a DFN object is created with dimensions 10 m x 30 m x 30 m. Next, a seed value is provided to the pseudo-random number generation system. Afterwards, three fracture sets are

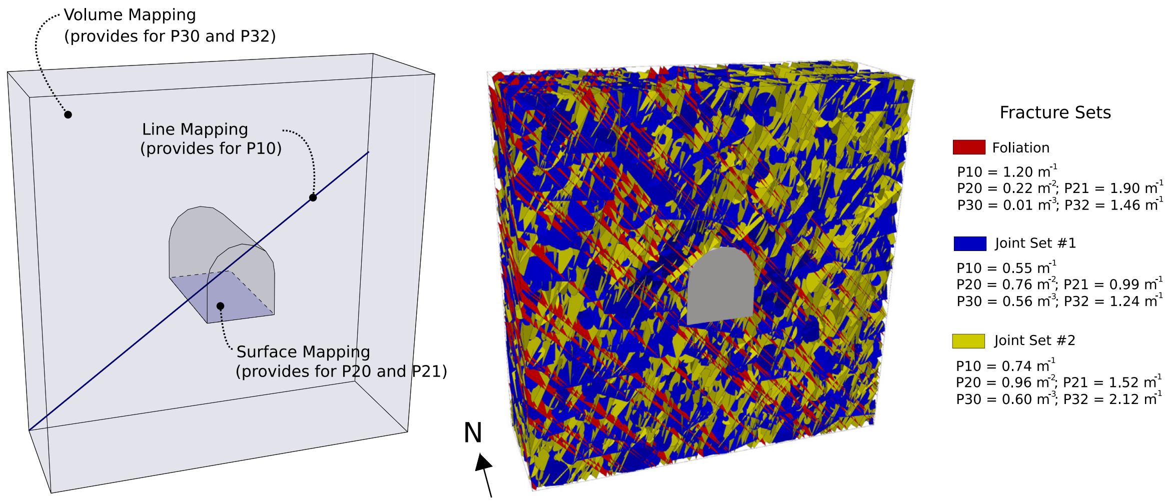
Volume Mapping
(provides for P30 and P32)
Line Mapping
(provides for P10)
Surface Mapping
(provides for P20 and P21)
N
Fracture Sets
Foliation
P10 = 1.20 m^{-1}
P20 = 0.22 m^{-2}; P21 = 1.90 m^{-1}
P30 = 0.01 m^{-3}; P32 = 1.46 m^{-1}
Joint Set #1
P10 = 0.55 m^{-1}
P20 = 0.76 m^{-2}; P21 = 0.99 m^{-1}
P30 = 0.56 m^{-3}; P32 = 1.24 m^{-1}
Joint Set #2
P10 = 0.74 m^{-1}
P20 = 0.96 m^{-2}; P21 = 1.52 m^{-1}
P30 = 0.60 m^{-3}; P32 = 2.12 m^{-1}

Fig. 5. Mapping regions used, constructed stochastic 3D DFN and fracture intensity values obtained for each fracture set.

Table 2
Stochastic Discrete Fracture Network parameters for the illustrative example.

Fracture	Orientation			Persistence[a]			Intensity
Set	Dip (°)	Dip Dir. (°)	Fisher K	Mean (m)	Std. (m)	Dist.	(1/m)
Foliation	48	110	–	∞	–	–	1.2 (P10)
Joint Set 1	71	180	23	0.725	0.52	Log-N.	1.24 (P32)
Joint Set 2	65	258	30	1.02	0.445	Log-N.	2.12 (P32)

[a]Persistence here refers to the radius distribution of the fractures.

added to the DFN and a line, a surface and a volume mapping are added so as to provide for the calculations of discontinuity intensities. Subsequently, loops are performed to add new circular fractures following the calibrated DFN parameters. Lastly, the DFN is exported to a VTK file for visualization in Paraview software. For detailed information concerning the commands applied, it is recommended that the library manual be consulted. The mapping regions, the constructed stochastic 3D DFN and the fracture intensity values obtained for each fracture set are shown in Fig. 5 together with the tunnel excavation region.

```
from unblocks import *
import plotTools
dfn = DFN()
dfn.set_RegionMaxCorner([10,30,30])
dfn.set_RandomSeed(100)
dfn.add_FractureSet()
dfn.add_FractureSet()
dfn.add_FractureSet()
dfn.add_LineMapping([10,30,0],
 [0.766582,4.631392,24.30794])
dfn.add_QuadrilateralMapping
 ([0.0,12.3,12.3],[0.0,17.7,12.3],
 [10.0,17.7,12.3],[10.0,12.3,12.3])
dfn.add_VolumeMapping()
while (dfn.linesMapping[0].get_P10(0) < 1.2):
 dfn.fractureSets[0].add_BaecherFracture
 (110, 48, 50000,"det", 200, 0)
while (dfn.volumesMapping[0].get_P32(1) < 1.24):
 dfn.fractureSets[1].add_BaecherFracture
 (180, 71, 23,"log", 0.725, 0.52)
while (dfn.volumesMapping[0].get_P32(2) < 2.12):
 dfn.fractureSets[2].add_BaecherFracture
 (258, 65, 30,"log", 1.02, 0.445)
dfn.export_DFNVtk("dfn")
```

Once the DFN is ready, it can be imported to the generator object for generating the 3D rock block system using the commands below. First, a generator object is created, followed by the definition of permissible minimum inscribed sphere radius and maximum block aspect ratio. It is important to define these permissible values carefully, so that blocks are still representative but bad geometries are not formed. Subsequently, the block system is generated based on the previously constructed DFN.

```
generator = Generator()
generator.set_MinInscribedSphereRadius(0.05)
generator.set_MaxAspectRatio(30)
generator.generate_RockMass(dfn)
```

Based on the block system, it is possible to analyze the block's shape and size using the 'plotTools' Python script developed as part of *UnBlocks^{gen}*. The following commands generate the block volume distribution, block shape diagram and block shape distribution respectively. Fig. 6 presents the mentioned plots.

```
plotTools.blockVolumeDistribution
 (gerador.get_Volumes(False))
plotTools.blockShapeDiagram
 (gerador.get_AlphaValues(False),
 gerador.get_BetaValues(False),
 gerador.get_Volumes(False), 0.05)
plotTools.BlockShapeDistribution
 (gerador.get_AlphaValues(False),
 gerador.get_BetaValues(False),
 gerador.get_Volumes(False))
plotTools.showPlots()
```

It is recommended to perform the analyses and plot the graphs before the tunnel region is excavated, otherwise cut blocks surrounding the tunnel, which are no longer representative of the constructed DFN, would be analyzed as well. Since the analyses have already been made, it is possible to perform the excavation.

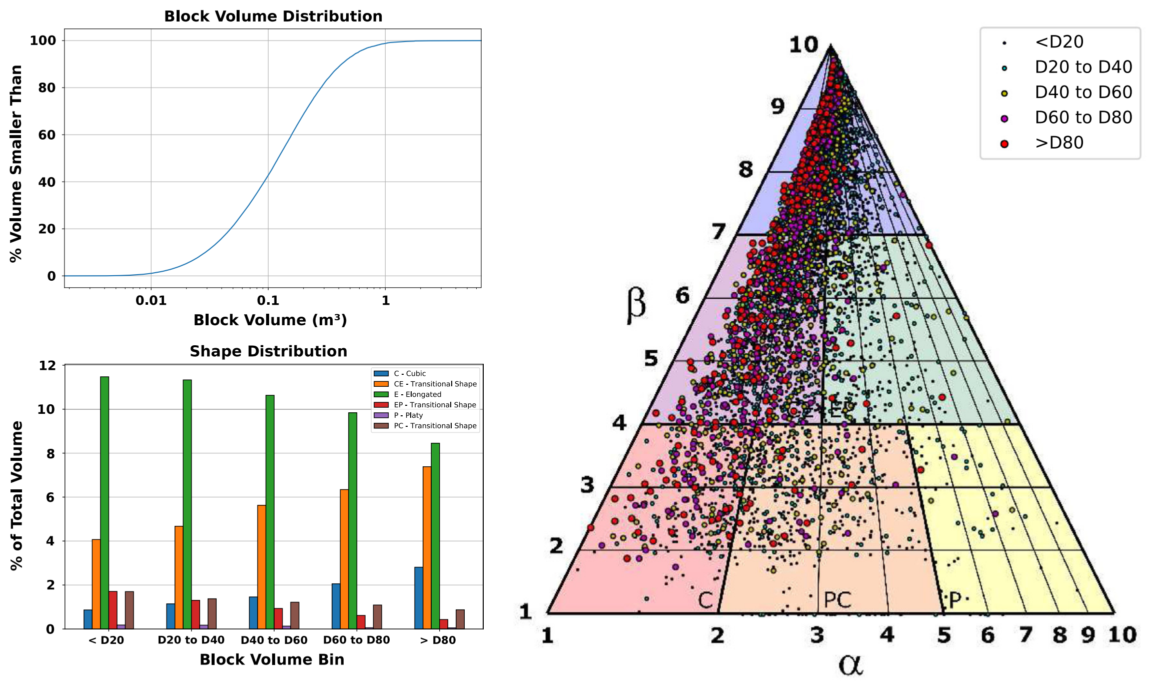

Fig. 6. Generated plots from the blocks' shape and size analysis: block volume distribution, block shape diagram and block shape distribution.

This is done by importing a watertight triangular mesh from a OBJ file representing the tunnel zone. Below, it is show the required Python commands to import the mesh, perform the excavation, and export the block system to a VTK file for visualization in Paraview software. Fig. 7 shows the block system generated after the excavation was performed.

```
generator.import_ExcavationElementsObj("tunnel")
generator.excavate_RockMass()
generator.export_BlocksVtk
 ("blocksAfterExcavation")
```

4. Impact

Several examples can be given as to how the *UnBlocks^{gen}* library ought to impact the rock mechanics and engineering scientific community. The library provides an easy-to-use tool to construct DFNs and to generate and analyze rock masses' block system. Although researchers have shown the importance of blocks' shape and size analyses, up to this moment no software package has been developed which was capable of automatically performing such calculations and plotting the results, leaving to the researcher the task of developing spreadsheets to achieve this goal.

It should be pointed that the library can be used to pursue new research questions. For instance, it is possible to further explore the relations between the blocks' shape and size of a rock mass and its stability condition, as observed during an engineering work in the field.

The library also paves the way to the development of new computational tools for 3D rock mass simulation. Numerical methods (e.g. Discrete Element Method and Discontinuous Deformation Analysis) and limit equilibrium analysis techniques require geometrical information regarding the rock blocks that are simulated. *UnBlocks^{gen}* can be used in combination with these methods for the fact that it calculates the required geometrical information (i.e. vertices, edges and facets data) for each rock block it generates.

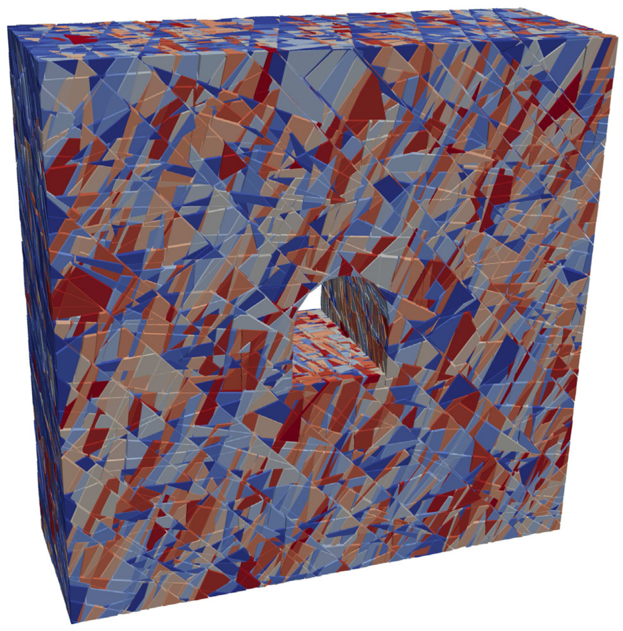

Fig. 7. 3D block system generated from the constructed DFN with tunnel excavation. Approximately 130,000 rock blocks were generated. The blocks are slightly shrunk to facilitate their visualization.

Concluding, it is interesting to remind that the library deals with a geometrical problem. Polyhedra are generated based on the intersection of planar elements in space. Therefore, the library may also be of interest to researchers outside its intended community, potentially including mathematics and physics related fields.

5. Conclusions

In this paper the *UnBlocks^{gen}* was presented, a Python library for constructing Discrete Fracture Networks, generating 3D rock block systems and analyzing rock blocks' geometrical characteristics, such as shape and size.

It is believed that the library will be of interest to the rock mechanics and engineering scientific community. It could either be used in research or incorporated into novel rock mass analysis software packages.

There are various possible future directions for the library. Currently, a limit equilibrium analysis methodology is being implemented into the *UnBlocks^{gen}* code, which will permit the rapid calculation of the factor of safety value for rock blocks around excavation zones. Furthermore, the Discrete Fracture Network construction scheme may as well be extended to cover more complex fracture system models.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] International society for rock mechanics commission on standardization of laboratory and field tests: Suggested methods for the quantitative description of discontinuities in rock masses. Int J Rock Mech Min Sci Geomech Abstr 1978;15(6):319–68. http://dx.doi.org/10.1016/0148-9062(78)91472-9.

[2] Goodman R, Shi G. Block theory and its application to rock engineering. Englewood Cliffs, NJ: Prentice-Hall; 1985, p. 338.

[3] Buyer A, Aichinger S, Schubert W. Applying photogrammetry and semi-automated joint mapping for rock mass characterization. Eng Geol 2020;264:105332. http://dx.doi.org/10.1016/j.enggeo.2019.105332.

[4] Wang L, Yamashita S, Sugimoto F, Pan C, Tan G. A methodology for predicting the in situ size and shape distribution of rock blocks. Rock Mech Rock Eng 2003;36:121–42. http://dx.doi.org/10.1007/s00603-002-0039-8.

[5] Palmstrom A. Measurements of and correlations between block size and rock quality designation (RQD). Tunnel Undergr Space Technol 2005;20(4):362–77. http://dx.doi.org/10.1016/j.tust.2005.01.005.

[6] Dershowitz W, Lee G, Geier J, LaPointe P. FracMan: interactive discrete feature data analysis, geometric modelling and exploration simulation. User Documentation. Seattle: Golder Associates Inc; 1998.

[7] Elmouttie M, krähenbühl G. A new excavation analysis method for slope design using discrete fracture network based polyhedral modelling. Comput Geotech 2016;76:93–104. http://dx.doi.org/10.1016/j.compgeo.2016.02.014.

[8] Itasca. 3DEC (3 dimensional distinct element code) version 5.0. Minneapolis, MN, USA: Itasca Consulting Group Inc; 2013.

[9] Warburton P. A computer program for reconstructing blocky rock geometry and analyzing single block stability. Comput Geosci 1985;11(6):707–12. http://dx.doi.org/10.1016/0098-3004(85)90013-5.

[10] Heliot D. Generating a blocky rock mass. Int J Rock Mech Min Sci Geomech Abstr 1988;25(3):127–38. http://dx.doi.org/10.1016/0148-9062(88)92295-4.

[11] Jakubowski J. Prediction of the load of tunnel support in the rock mass of blocky structure by statistical methods. (Ph.D. thesis), Kraków, Poland: University of Mining and Metallurgy; 1994.

[12] Shi G. Producing joint polygons, cutting joint blocks and finding key blocks for general free surfaces. Chin J Rock Mech Eng 2006;25(11):2161–70.

[13] Merrien-Soukatchoff V, Korini T, Thoraval A. Use of an integrated discrete fracture network code for stochastic stability analyses of fractured rock masses. Rock Mech Rock Eng 2012;45:159–81. http://dx.doi.org/10.1007/s00603-011-0136-7.

[14] Wang S, Ni P, Guo M. Spatial characterization of joint planes and stability analysis of tunnel blocks. Tunnel Undergr Space Technol 2013;38:357–67. http://dx.doi.org/10.1016/j.tust.2013.07.017.

[15] Xia L, Yu Q, Chen Y, Li M, Xue G, Chen D. Generalblock: A c++ program for identifying and analyzing rock blocks formed by finite-sized fractures. In: Environmental software systems. infrastructures, services and applications. IFIP advances in information and communication technology, vol. 448, Melbourne, Australia; 2015.

[16] Fu G, Ma G, Qu X, Huang D. Stochastic analysis of progressive failure of fractured rock masses containing non-persistent joint sets using key block analysis. Tunnel Undergr Space Technol 2016;51:258–69. http://dx.doi.org/10.1016/j.tust.2015.10.013.

[17] Fu X, Sheng Q, Wang L, Chen J, Zhang Z, Du Y, et al. Spatial topology identification of three-dimensional complex block system of rock masses. Int J Geomech 2019;19(12):04019127. http://dx.doi.org/10.1061/(ASCE)GM.1943-5622.0001522.

[18] Zhang L, Sherizadeh T, Zhang Y, Sunkpal M, Liu H, Yu Q. Stability analysis of three-dimensional rock blocks based on general block method. Comput Geotech 2020;124:103621. http://dx.doi.org/10.1016/j.compgeo.2020.103621.

[19] Kalenchuk KS, Diederichs MS, McKinnon S. Characterizing block geometry in jointed rockmasses. Int J Rock Mech Min Sci 2006;43(8):1212–25. http://dx.doi.org/10.1016/j.ijrmms.2006.04.004.

[20] Lei Q, Latham J-P, Tsang C-F. The use of discrete fracture networks for modelling coupled geomechanical and hydrological behaviour of fractured rocks. Comput Geotech 2017;85:151–76. http://dx.doi.org/10.1016/j.compgeo.2016.12.024.

[21] Dershowitz W, Einstein H. Characterizing rock joint geometry with joint system models. Rock Mech Rock Eng 1988;21:21–51. http://dx.doi.org/10.1007/BF01019674.

[22] Dershowitz WS, Herda H. Interpretation of fracture spacing and intensity. In: 33rd US Symp. Rock Mech., Sante Fe, USA; 1992. p. 757.

[23] Priest S. Discontinuity analysis for rock engineering. Springer; 1993, p. 473. http://dx.doi.org/10.1007/978-94-011-1498-1.

[24] Boon C, Houlsby G, Utili S. A new rock slicing method based on linear programming. Comput Geotech 2015;65:12–29. http://dx.doi.org/10.1016/j.compgeo.2014.11.007.

[25] Guennebaud G, Jacob B, et al. Eigen v3. 2010, http://eigen.tuxfamily.org.

[26] Forrest J, Hafer L, Hall J, Saltzman M. coin-or/Clp: Version 1.17.6. Zenodo; 2020, http://dx.doi.org/10.5281/zenodo.3748677.

[27] Hunter J. Matplotlib: A 2D graphics environment. Comput Sci Eng 2007;9(3):90–5. http://dx.doi.org/10.1109/MCSE.2007.55.

[28] Boost. Boost C++ Libraries. 2020, http://www.boost.org/, (Last Accessed 15 January 2020).

[29] Ayachit U. The paraview guide: A parallel visualization application. Kitware; 2015.

[30] Vtk file formats. 2020, https://vtk.org/wp-content/uploads/2015/04/file-formats.pdf, (Accessed 10 July 2020).

[31] VTK Polyhedron support. 2020, https://vtk.org/Wiki/VTK/Polyhedron_Support, (Accessed 10 July 2020).

[32] Rasmussen LL, Cacciari PP, Futai MM, de Farias MM, de Assis AP. Efficient 3D probabilistic stability analysis of rock tunnels using a Lattice model and cloud computing. Tunnel Undergr Space Technol 2019;85:282–93. http://dx.doi.org/10.1016/j.tust.2018.12.022.