Mesh Generation and Flow Simulation in Large Tridimensional Fracture Networks

Patrick Laug, Géraldine Pichot

Mesh Generation and Flow Simulation in Large Tridimensional Fracture Networks

Patrick Laug
Inria Saclay Île-de-France, Gamma3 team,
1 rue Honoré d'Estienne d'Orves, 91120 Palaiseau, France
patrick.laug@inria.fr

Géraldine Pichot
Inria Paris & Université Paris-Est, CERMICS (ENPC), Serena team,
2 rue Simone Iff, 75012 Paris, France
geraldine.pichot@inria.fr

Abstract

Fractures in the Earth's subsurface have a strong impact in many physical and chemical phenomena, as their properties are very different from those of the surrounding rocks. They are generally organized as multiscale structures, which can be modeled by Discrete Fracture Networks (DFNs) that may contain hundreds of thousands of ellipses in the tridimensional space. This paper presents our approach to generate meshes of such large DFNs and to simulate single-phase flow problems using these meshes.

Keywords: Geologic formation, Discrete Fracture Network (DFN), Surface meshing, Large meshes, Flow simulation.

1. Introduction

Fractured rocks are commonly encountered under the Earth's surface. Fractures by themselves have a strong impact in many physical and chemical phenomena, as their properties (in particular their permeabilities) are very different from those of the surrounding rocks. They thus play a major role in diverse fields of applications such as groundwater extraction, oil and gas exploitation, geothermal energy production, CO2 sequestration, etc. In this paper, we focus on large Discrete Fracture Network (DFN) models and on efficient techniques to mesh them and to carry out numerical single phase flow simulations, using the generated meshes.

2. Modeling and meshing large DFNs

This section describes the successive steps for modeling (2.1 to 2.4 below) and meshing (2.5) large DFNs. All these steps present special difficulties in our context with a large number of fractures with distances, lengths and angles spanning over several orders of magnitude. The corresponding algorithms have been implemented in `BLSURF_FRAC` software.

2.1. Read a DFN model and meshing parameters

We assume here that a geometric model of a DFN has already been generated, using stochastic methods based on experimental statistics (for further details, see [1, 2, 3, 4]). In this model, each fracture is represented by a *disk*, that is, a planar region bounded by a circle or more generally an ellipse. Each disk is defined by the 3D coordinates of its center, its normal vector, and its radius (for a circle) or major and minor axes (for an ellipse). A representative sample may contain over one million disks, over one million intersections between each other, and radii ranging from one meter or less to one kilometer or more.

In the following, the computational domain will be called the *cube*, although it may more generally be a rectangular parallelepiped. Figure 1 shows the part of a DFN contained in a cube.

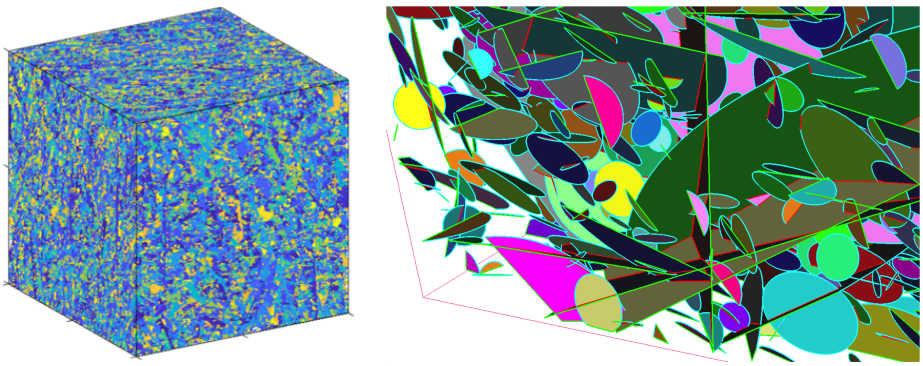

Figure 1: *Discrete Fracture Network (DFN) including 1,176,566 fractures and 2,410,537 intersections inside a cube (full view and close-up).*

In addition to this geometric model, some parameters can also be given. Classical parameters like the desired size of the elements control the mesh generator [5]. More specific parameters have been incorporated, as for instance:

- L_x, L_y, L_z*:* dimensions of the *cube* or more generally the parallelepiped-shaped computational domain.
- *conforming:* boolean parameter indicating if the final mesh must be conforming (*i.e.*, matching) or not at the intersection between fractures.
- *cube_faces:* among the six faces of the *cube*, selection of those where boundary conditions (BCs) are prescribed.

- *qmin:* minimum mesh quality required.

2.2. Compute disk/disk and disk/cube intersections

In this 2nd step, the intersections of the disks between each other must be computed, as well as the intersections of the disks with each face of the *cube*. In both cases, any intersection is a line segment. Disks that are completely outside the cube are not involved and are thus eliminated for efficiency reasons.

For disk/disk intersections, a naive algorithm would consist of two nested loops on the disks, which would be too costly if the number of disks is large. Therefore, an "optimizing grid" is built beforehand. This grid is a regular subdivision of the *cube*, where each cell of the grid points to a list of the disks crossing the cell. The optimized algorithm consists of a loop on the cells, and for each cell the intersections between its crossing disks, which is much faster.

To find the intersection between two disks D_1 and D_2, if it exists, first an intersection of their containing planes Π_1 and Π_2 is calculated by solving a linear system. If a solution exists, an equation $P + V\,t, t \in]-\infty, +\infty[$ of a line L is obtained, where P is a point and V a vector in $\mathbb{R}^3$. The intersection of line L with a disk $D_i, i = 1, 2$, is a line segment $P + V\,t, t \in [a_i, b_i]$, where a_i and b_i are the solutions (if they exist) of a second degree equation in plane Π_i. Finally, the intersection of disks D_1 and D_2 is the line segment $P + V\,t, t \in [a, b]$ where $[a, b] = [a_1, b_1] \cap [a_2, b_2]$ if it is not the empty set.

Computing the intersection between a disk and a face of the *cube* is similar, the boundary of D_2 becoming a rectangle instead of an ellipse.

2.3. Select disks taking part in the numerical simulation

At this stage, we have a set $D = \{D_i\}$ of disks, or parts of disks, which are all inside the *cube* (see Figure 1). It is useful to represent their connectivity by an undirected graph $G = (F, E)$ where F and E are the sets of nodes and edges of graph G. The first set $F = \{F_i\}$ is the set of geometric faces $F = D \cup C$ where C is the set of the six faces of the cube. The second set $E = \{E_i\}$ is such that each element $E_i = \{F_j, F_k\}$ indicates the presence of an intersection between faces F_j and F_k.

Searches can be optimized thanks to this graph. In particular, this 3rd step aims to keep disks taking part in the numerical simulation and to remove all the others. For example, suppose that Dirichlet BCs are only given on two faces C_1 and C_2 of the cube. The transitive closure of G gives all the faces that can be reached from face C_1. All the other faces can be removed, as well as the edges of G involving these removed faces. This provides a reduced graph $G' = (F', E')$. Then, the transitive closure of this reduced graph G' gives all the faces that can be reached from face C_2, providing an even more reduced graph $G'' = (F'', E'')$. This algorithm can obviously be applied for an arbitrary number of initial cube faces or disks (cf. parameter *cube_faces* in Section 2.1).

2.4. On each face, make a conforming model of the intersections with other faces

At the beginning of this 4^{th} step, all the relevant faces (disks or cube faces) are known, as well as their intersections in the tridimensional space. Each face F_i is simply a bidimensional domain containing line segments that are the intersections of F_i with other faces. In turn, these segments can intersect each other at some points, and then a conforming set of segments must be made before meshing the domain. Like in Section 2.2, a bidimensional "optimizing grid" is built at the beginning to optimize subsequent computations. Vertices are added at the intersections of the segments, splitting segments into subsegments. Close vertices (points within a given small distance) are merged. Also, topological consistency between faces must be ensured: if a segment S is an intersection between two faces F_i and F_j, the intersection points found on S for face F_i must also be found when considering domain F_j. The whole algorithm must be robust in the case of small distances between vertices or segments, small angles between segments, and large numbers of segments. As an illustration, Figure 2 shows an initial set of line segments and the conforming model obtained.

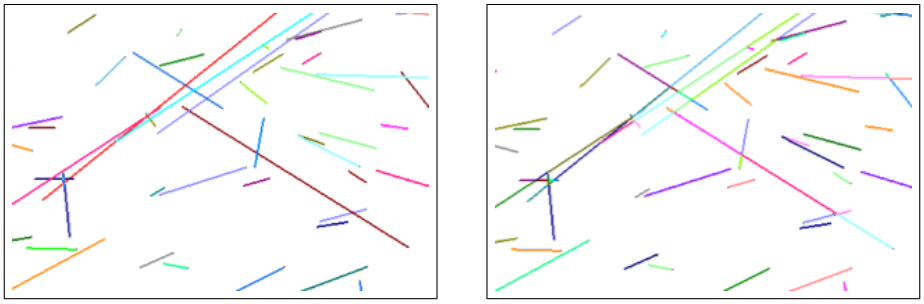

Figure 2: *Intersections of a face with others, before and after making a conforming model (one color per line segment or subsegment).*

2.5. Build the tridimensional mesh of the DFN

By now, our geometric model consists of a collection of connected faces. Each face is the image of a bidimensional domain by an affine function. This is a particular case of a CAD (computer-aided design) model, where a surface Σ is defined by a collection of patches Σ_i in $\mathbb{R}^3$, fitted together in a conforming manner, and verifying $\Sigma_i = \sigma_i(\Omega_i)$, where Ω_i is a parametric domain in $\mathbb{R}^2$ and σ_i is a C^1 continuous mapping. These CAD surfaces can be meshed by different strategies, in particular an indirect method [6, 7] whose general scheme is briefly recalled hereafter: (1) build a discretization of each 3D curve (disk contours or face intersections); (2) project each 3D discretization to define the 2D discretized boundaries of the corresponding parametric domains; (3) ensure that each parametric domain is well defined; (4) generate a mesh of each parametric

domain Ω_i from the discretization of its boundary (obtained in the previous step); (5) map the mesh of each Ω_i onto Σ_i; (6) make a surface mesh of Σ from meshes of Σ_i; (7) if necessary for the numerical simulation, make a volume mesh with this surface mesh as input.

Step 3 of this general scheme is particularly efficient in the present context. Usually, for each parametric domain, it should be checked that each edge of the discretized curves does not intersect another edge. Here, the set of discretized subsegments at the intersections of faces is already conforming (cf. Section 2.4) and the discretized set is necessarily correct. It is thus sufficient to check that each vertex of this discretized set is inside the discretized contour of the disk. If not, the incorrect edge of the contour is subdivided (with a new vertex on the ellipse-shaped contour) and checks are restarted. In this process, an "optimizing grid" (cf. Section 2.2) can also be used.

In this meshing process, nonconforming or conforming meshes can be made, depending on the user-defined parameter *conforming* (cf. Section 2.1). Figure 3 shows two surface meshes with, for each patch Σ_i, a prescribed size h_i of the elements. The first method consists in meshing each patch regardless of adjacent patches. Each element of a given patch Σ_i then has a size close to h_i, including around its boundaries. Although the subsegments of the geometric model are conforming (cf. Section 2.4), the obtained mesh is nonconforming, as can be seen on the left side of the figure. In a second method that is more conventional in CAD environments, if a subsegment is at the intersection of two patches Σ_i and Σ_j, it is discretized only once with a size h_{ij} between h_i et h_j. To avoid excessive variation between h_{ij} and h_i on Σ_i, and between h_{ij} et h_j on Σ_j, which would affect the quality of the mesh, a size gradation is applied. The right side of the figure shows a conforming mesh with gradation 1.2.

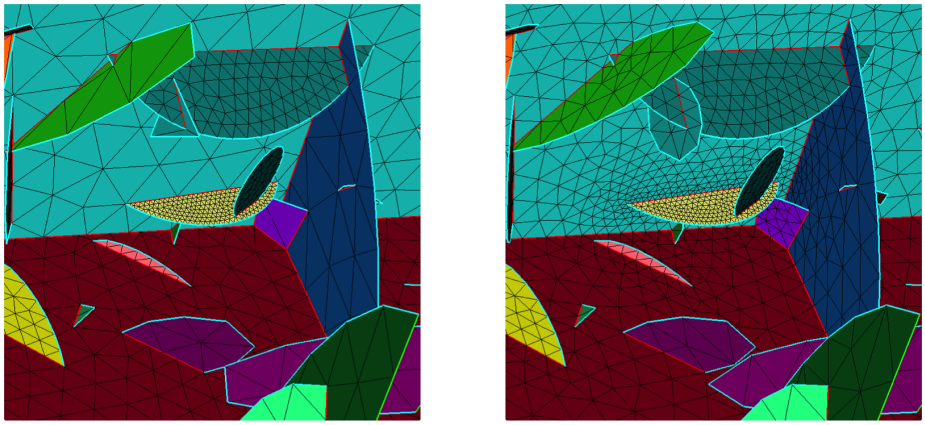

Figure 3: *Nonconforming and conforming meshes.*

3. Flow simulation in large DFNs

We now propose to solve single phase flow problems in these large DFNs. Section 3.1 recalls the governing equations, Section 3.2 presents the numerical method, and Section 3.3 its implementation in `NEF-Flow` software.

3.1. Governing equations

We consider single phase flow problems within the network of fractures. The rock matrix is assumed to be impervious. Let $\mathbf{x}$ be the local 2D coordinates of fracture D_i. Let N be the total number of intersections between fractures, I_k be the k^{th} intersection, $k = 1, ..., N$, and F_k be the set of fractures containing I_k. In each fracture D_i, we assume that the governing equations for the hydraulic head scalar function $p(\mathbf{x})$ and for the flux per unit length function $\mathbf{u} = \mathbf{u}(\mathbf{x})$ are the mass conservation equation and Poiseuille's law [8]:

$$\nabla \cdot \mathbf{u}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) \qquad \text{for } \mathbf{x} \in D_i\,, \tag{1a}$$

$$\mathbf{u}(\mathbf{x}) = -\mathcal{T}(\mathbf{x})\,\nabla p(\mathbf{x}) \qquad \text{for } \mathbf{x} \in D_i\,. \tag{1b}$$

The parameter $\mathcal{T}(\mathbf{x})$ is a given transmissivity field (unit $[\mathrm{m}^2.\mathrm{s}^{-1}]$). The function $\mathbf{f}(\mathbf{x}) \in L^2(D_i)$ represents the sources/sinks. Additionally, continuity of the hydraulic head and continuity of the transversal flux apply at the intersections between the fractures [9, 10]:

$$p_{k,i} = p_k \qquad \text{on } I_k,\ \forall i \in F_k\,, \tag{2a}$$

$$\sum_{i \in F_k} \mathbf{u}_{k,i} \cdot \mathbf{n}_{k,i} = 0 \qquad \text{on } I_k\,, \tag{2b}$$

where $p_{k,i}$ is the trace of hydraulic head on I_k in fracture D_i, p_k is the unknown hydraulic head on the intersection I_k and $\mathbf{u}_{k,i} \cdot \mathbf{n}_{k,i}$ is the normal flux through I_k coming from fracture D_i, with $\mathbf{n}_{k,i}$ the outward normal unit vector of the intersection I_k with respect to the fracture D_i.
BCs on the cube faces are of Dirichlet or Neumann type. For edges that belong to the border of the fractures but not to a cube face, a homogeneous Neumann BC is applied to express the imperviousness of the rock matrix.

3.2. Mixed hybrid finite element method

Once a mesh of the network of fractures is built (see Section 2), the flow problem (1a-2b) can be solved with a mixed hybrid finite element method [11, 12, 13]. For lowest order Raviart-Thomas RT0 finite elements, the unknowns on each finite element are the traces of hydraulic head, the mean head and the fluxes through the edges. There are several advantages of the mixed hybrid formulation: (i) there are unknowns on the edges which makes it easier to impose conditions (2a)-(2b) at the intersections and boundary conditions; (ii) algebraically, for conforming or nonconforming meshes [14, 15, 16, 17], a Schur complement system is obtained for only the traces of hydraulic head. The matrix has an arrow shape and is symmetric positive definite; (iii) this method guarantees local and global mass conservation.

3.3. `NEF-Flow` software

The `NEF-Flow` software is written in Matlab and implements the mixed hybrid finite element method to solve single phase flow in large scale DFNs. `NEF` stands for Numerical Experiments involving Fractures. `NEF-Flow` handles either matching or nonmatching meshes at the intersection between fractures, sink/source terms and contrasts in transmissivity. Loops are known to be very costly in Matlab code. Matlab vectorization is used massively in `NEF-Flow` to decrease the computational time. Whenever possible, operations are applied simultaneously to a large set (or even all) triangles/edges. The following steps are implemented in `NEF-Flow`:

1. **Load the mesh and BCs.** For efficiency purpose, in addition to the ".mesh" file, several data structures are saved in files by the mesh generator. All those files are directly loaded in Matlab using the *textscan* function (much more efficient than *textread*). The content of these files are given in [2] together with an additional file to store, for each intersection, the fracture it belongs to and its list of edges with their global edge indexes. It is useful for conforming and nonconforming meshes. As each intersection is shared by two fractures, its index appears twice in this file.
2. **Linear system numbering.** To get the arrow shape of the Schur complement matrix, inner edges are numbered first, followed by intersection edges. For efficiency purpose, the correspondence between local numbering and global numbering is stored in a dedicated structure.
3. **Build local matrices.** Local matrices were originally built according to RT0 finite element basis functions. Now P^1 nonconforming finite elements are preferred [18, 19] as the computation is less expensive. If the mesh is nonconforming, coupling Mortar conditions are also computed (according to the formula given in [16, 17]).
4. **Linear system assembling.** A sparse storage is used for the Schur complement matrix and second member. The assembling contains two steps: (i) local matrices are all assembled without taking the BCs into account; (ii) then loops on boundary edges are done to add Neumann, Dirichlet and sink/sources terms conditions. Step (i) is done using two loops on the edges of triangles (loops from 1 to 3) instead of costly loops over the triangles. It follows algorithm 4.1 given in [20].
5. **Solving the linear system.** A Matlab direct solver is used to solve the Schur complement system in order to compute the traces of hydraulic heads on each triangle. As the matrix of the linear system is symmetric positive definite, Cholesky factorization is preferred.
6. **Compute mean hydraulic heads and fluxes on each triangle.** These computations can be done independently from one triangle to another.
7. **Check steps.** Several tests are performed to check if the continuity conditions on inner edges, local and global mass conservations and BCs are satisfied, up to a user input tolerance.

4. Benchmark test cases

We extend the benchmark test case proposed in [1, 2] to DFNs generated with the UFM framework [3, 4]. These DFNs are large scale DFNs where the fracture size distribution matches the observations. Fractures are organized so that large fractures inhibit the smaller ones, creating T-termination configurations. Two PCs are used for the simulations: a MacBook Pro, 4 processors Intel Core i7 at 2.9 GHz, 16 GB RAM (referred to as Mac in the following) and a Dell PC Intel Core i7 4 core CPU at 2.90 GHz, 32 GB RAM (referred to as Dell in the following). The given user input tolerance in step 7 of Section 3.3 is 2e-07.

4.1. Description of the test cases and examples of computational time for the mesh generation

In the following, we consider a cubic domain, $L_x = L_y = L_z = L$. We consider 4 networks labeled DFN50, DFN100, DFN150 and DFN200 with cube size 50, 100, 150 and 200 respectively. Table 1 summarizes the test cases, and Table 2 gives examples of the computational time for the mesh generation. Memory swapping is observed for the mesh generation of the test case DFN150.

Label	#fractures	#intersections
DFN50	2,401	3,526
DFN100	19,007	28,727
DFN150	508,339	1,031,231
DFN200	146,487	219,521

Table 1: *Test cases.*

Label	#triangles	Meshing	PC
DFN50	28,856	0.044s	Mac
DFN100	237,427	0.331s	Mac
DFN150	8,112,299	15min53s (swapping)	Dell
DFN200	1,829,802	2.563s	Mac

Table 2: *Mesh generation.*

4.2. Computational time to solve the flow problem

For DFN150, the transmissivity is set as a constant for all fractures and equal to 1 $m^2.s^{-1}$. For the other test cases, it has one given value per fracture in the range [2.6e-06; 47.4] $m^2.s^{-1}$. Table 3 presents the results and the computational time associated with the different steps of the algorithm described in subsection 3.3. Simulations are performed on the Dell PC. We impose a permeameter boundary condition with a difference of hydraulic head $\Delta h = 10\,\text{m}$ in the x direction. The equivalent permeability K_x [$m^2.s^{-1}$] [21] in the x direction is

Label	#fractures	#triangles	Step 1	Steps 2-7	K_x
DFN50	2,401	28,856	2.96s	1.52s	5.99e-03
DFN100 (coarse)	19,007	1,051,949	11.96s	36.28s	4.16e-03
DFN100 (fine)	19,007	12,214,867	64.54s	7min (swapping)	4.33e-03
DFN150	508,339	8,112,299	5min25s	21min (swapping)	1.07e-01
DFN200	146,487	1,829,802	42.89s	2min38s	3.61e-03

Table 3: *Flow simulations.*

computed. Figure 4 shows the mean hydraulic heads obtained for the test cases DFN150 (left) and DFN200 (right). In these two cases, memory swapping was observed in the solving of the linear system (see table 3).

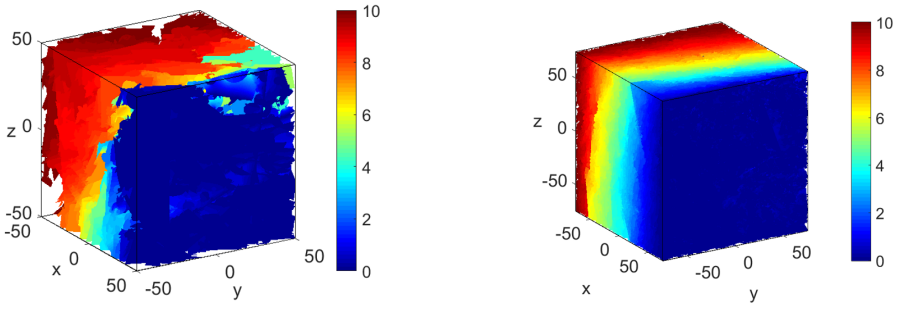

Figure 4: *Mean hydraulic heads computed for permeameter boundary conditions in the x direction. Left, DFN100 (fine) test case. Right, DFN150 test case.*

5. Conclusion

We have presented a workflow for mesh generation and flow simulation in large tridimensional fracture networks. The geometric modeling and mesh generation are implemented in `BLSURF_FRAC` software. The flow simulation is performed by `NEF-Flow` software. Very promising results are obtained with this combination of software: robustness, efficiency, accuracy (up to hundreds of thousands fractures). Recently, flow has been simulated in a dense case, with 1,176,566 of fractures, 2,410,537 of intersections and 18,755,684 of triangles, but this requested more than one hour of computation. We are currently working on code optimization (parallelization, iterative solvers to avoid memory swappings, ...). We also started a work on volume meshing to consider the case of a pervious rock matrix.

References

[1] P. Laug, G. Pichot & J.R. de Dreuzy, Realistic geometric modeling of fracture networks, *ADMOS Int. Conf.*, Verbania, Italy, June 2017.

[2] J.R. de Dreuzy, G. Pichot, B. Poirriez & J. Erhel, Synthetic benchmark for modeling flow in 3D fractured media, *Computers & Geosciences* **50**, 59–71 (2013).

[3] P. Davy, R. Le Goc, C. Darcel, O. Bour, J.R. de Dreuzy & R. Munier, A likely universal model of fracture scaling and its consequence for crustal hydromechanics, *Journal of Geophysical Research: Solid Earth*, **115 (B10)** (2010).

[4] P. Davy, R. Le Goc & C. Darcel, A model of fracture nucleation, growth and arrest, and consequences for fracture density and scaling, *Journal of Geophysical Research: Solid Earth*, **118 (4)**, 1393–1407 (2013).

[5] P. Laug & H. Borouchaki, BLSURF – Mesh Generator for Composite Parametric Surfaces – User's Manual, *Inria Technical Report* RT-0235, Nov. 1999.

[6] H. Borouchaki, P. Laug & P.L. George, Parametric surface meshing using a combined advancing-front – generalized-Delaunay approach, *Int. Journal for Numerical Methods in Eng ineering*, **49 (1–2)**, pp. 233–259 (2000).

[7] P.L. George, H. Borouchaki, F. Alauzet, P. Laug, A. Loseille, L. Maréchal, *Maillage, modélisation géométrique et simulation numérique 2 - Métriques, maillages et adaptation de maillages*, ISTE Editions, 412 pages, sept. 2018 (English translation to appear).

[8] V. Martin, J. Jaffré & J. E. Roberts, Modeling fractures and barriers as interfaces for flow in porous media, *SIAM J. Sci. Comput.*, 26 (5), pp. 1667–1691 (2005).

[9] J. Erhel, J.- R. de Dreuzy, B. Poirriez, Flow simulation in three-dimensional discrete fracture network, *SIAM Journal on Scientific Computing*, Vol. 31, No. 4, pp. 2688-2705, 2009.

[10] J. Maryška, O. Severýn & M. Vohralík, Numerical simulation of fracture flow with a mixed-hybrid FEM stochastic discrete fracture network model, *Computational Geosciences*, 8, pp. 217-234 (2004).

[11] P.A. Raviart & J. Thomas, A mixed finite element method for 2nd order elliptic problems, *Mathematical Aspects of the Finite Element Methods, Lectures Notes in Math.*, Springer, Berlin, 606, pp. 292–315 (1977).

[12] F. Brezzi & M. Fortin, *Mixed and Hybrid Finite Element Methods*, Springer-Verlag, New York, 1991.

[13] G. Chavent & J. E. Roberts, A unified physical presentation of mixed, mixed-hybrid finite elements and standard finite difference approximations for the determination of velocities in waterflow problems, *Advances in Water Resources*, 14 (6), pp. 329–348 (1991).

[14] D. N. Arnold & F. Brezzi, Mixed and nonconforming finite element methods: postprocessing, and error estimates, *RAIRO M2AN*, vol. 19, 7–32 (1985).

[15] T. Arbogast, L. C. Cowsar, M. F. Wheeler & I. Yotov, Mixed finite element methods on nonmatching multiblock grids, *SIAM J. Numer. Anal.*, 37 (4), pp. 1295-1315 (2000).

[16] G. Pichot, J. Erhel & J.-R. de Dreuzy, A mixed hybrid Mortar method for solving flow in discrete fracture networks, *Applicable Analysis*, 89 (10), pp. 1629-1643 (2010).

[17] G. Pichot, J. Erhel, & J.-R. de Dreuzy, A generalized mixed hybrid mortar method for solving flow in stochastic discrete fracture networks, *SIAM J. Sci. Comput.*, 34(1):B86–B105 (2012).

[18] T. Arbogast & Z. Chen, On the Implementation of Mixed Methods as Nonconforming Methods for Second- Order Elliptic Problems, *Mathematics of Computation*, 64 (211), pp. 943–972 (1995).

[19] M. Vohralík, J. Maryška & O. Severýn, Mixed and Nonconforming Finite Element Methods on a System of Polygons, *Appl. Numer. Math.*, 57 (2), pp. 176–193 (2007).

[20] F. Cuvelier, C. Japhet & G. Scarella, An efficient way to assemble finite element matrices in vector languages, *BIT Numerical Mathematics*,56 (3), pp. 833–864 (2016).

[21] P. Renard & G. de Marsily, Calculating Equivalent Permeability, Advances in Water Resources, 20, pp. 253-278 (1997).