SMAI-JCM
SMAI Journal of Computational Mathematics

Hybrid high-order methods for flow simulations in extremely large discrete fracture networks

Alexandre Ern, Florent Hédin, Géraldine Pichot & Nicolas Pignet

Volume 8 (2022), p. 375-398.
https://doi.org/10.5802/smai-jcm.92

The SMAI Journal of Computational Mathematics is a member of the Centre Mersenne for Open Scientific Publishing
http://www.centre-mersenne.org/
Submissions at https://smai-jcm.centre-mersenne.org/ojs/submission
e-ISSN: 2426-8399

SMAI Journal of Computational Mathematics
Vol. 8, 375-398 (2022)

Hybrid high-order methods for flow simulations in extremely large discrete fracture networks

Alexandre Ern [1]
Florent Hédin [2]
Géraldine Pichot [3]
Nicolas Pignet [4]

[1] Université Paris-Est, CERMICS (ENPC), 6 et 8 av. Blaise Pascal, 77455 Marne-la-Vallée Cedex 2, France and Inria, 2 rue Simone Iff, 75589 Paris, France
E-mail address: alexandre.ern@enpc.fr
[2] Inria, 2 rue Simone Iff, 75589 Paris, France and Université Paris-Est, CERMICS (ENPC), 6 et 8 av. Blaise Pascal, 77455 Marne-la-Vallée Cedex 2, France
E-mail address: florent.hedin@inria.fr
[3] Inria, 2 rue Simone Iff, 75589 Paris, France and Université Paris-Est, CERMICS (ENPC), 6 et 8 av. Blaise Pascal, 77455 Marne-la-Vallée Cedex 2, France
E-mail address: geraldine.pichot@inria.fr
[4] EDF Lab Paris-Saclay, 7 boulevard Gaspard Monge, 91120 Palaiseau, France
E-mail address: nicolas.pignet@edf.fr.

Abstract. We investigate the computational performance of hybrid high-order methods applied to flow simulations in extremely large discrete fracture networks (over one million of fractures). We study the choice of basis functions, the trade-off between increasing the polynomial order and refining the mesh, and how to take advantage of polygonal cells to reduce the number of degrees of freedom.

2020 Mathematics Subject Classification. 65N30, 65N50, 86A05.

Keywords. discrete fracture networks, flow simulations, hybrid high-order methods.

1. Introduction

It is well known that fractures play a major role in a wide range of applications (water resources, deep geological waste storage, geothermy, among others) and cannot be neglected in subsurface modelling as they are preferential flow paths [17]. Fractures are ubiquitous and have a large range of sizes (from centimeters to kilometers) and apertures. In this paper, we consider flow simulations in crystalline rocks where the rock matrix is assumed to be impervious and the fractures to have large transmissivities. We do not consider the case of fractures acting like geological barriers and situations where flow occurs in the rock matrix. A more general model can be found in [37]. A possible modelling strategy of the network of fractures is the Discrete Fracture Network (DFN) approach. Fractures are modelled as ellipses or disks whose properties (orientation, size, position, transmissivity) are governed by statistical laws derived from field observations [8, 12, 24]. This model has been enriched in [17, 18] to include the evolution of fracture network formation (nucleation, growth and arrest). The networks generated with an arrest rule contain large numbers of T-shape intersections (where a fracture stops on another one) compared to X-shape intersections (the two fractures cross each other). With no arrest rule, the networks rather contain X-shape intersections. In this paper, the DFNs we consider are generated with the software DFN.lab with no arrest rule. Examples of such networks are displayed in Figure 1.1.

This research has been partially supported by the LABEX AMIES ANR-10-LBX-0002-01 project.
https://doi.org/10.5802/smai-jcm.92

The largest test case contains 1,176,566 fractures embedded in a cubic domain of size $L = 200$m (labelled "L200": right panel). By decreasing the cube size, one obtains smaller (and embedded) DFNs: $L = 100$m (left panel, labelled "L100": 152,405 fractures) and $L = 150$m (central panel, labelled "L150": 508,339 fractures). To better emphasize the complexity of these DFNs, Table 1.1 gives the total number of intersections, the maximum number intersections per fracture, and the range of transmissivity values, given as input (one value per fracture).

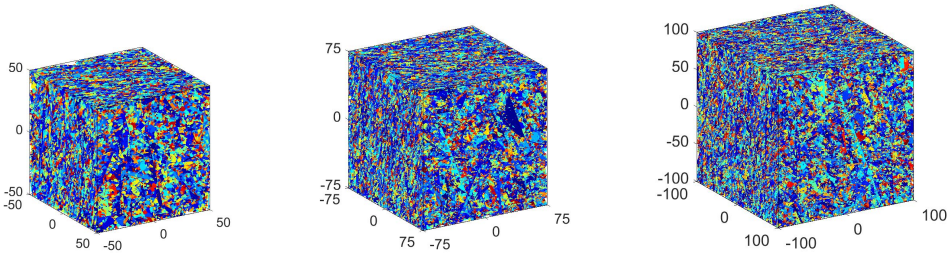

Figure 1.1. DFN test cases: (left) L100: 152,405 fractures; (center) L150: 508,339 fractures; (right) L200: 1,176,566 fractures. A different color is used for each fracture.

Table 1.1. Description of the three DFNs.

DFN	#fractures	#intersections	Max(#intersections per fracture)	transmissivity range ($[m^2.s^{-1}]$)
L100	152k	302k	1,022	[3.5e-06; 20.33]
L150	508k	1,031k	4,930	[3.38-06; 25.8]
L200	1,176k	2,410k	11,710	[3.35-06; 25.8]

Figure 1.2 displays the histogram of the number of intersections per fracture for the test case L200.

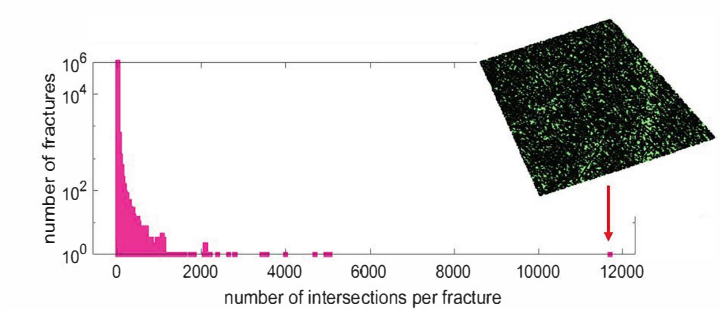

Figure 1.2. L200: histogram of the number of intersections per fracture.

As shown in Figure 1.2 and in agreement with natural field observations, there are mostly fractures with a few number of intersections (below 1000) and a few ones with a very large number of intersections. Figure 1.3 displays, for the test case L200, the fracture with the maximum number of intersections (11,710 intersections) and indicated on Figure 1.2 (right). Figure 1.3 (center and right) emphasizes the possible complexity of the network of intersections within a fracture.

Flow simulations are of primal importance to consider more involved physical and chemical couplings in these complex DFNs. Flows in fractured rocks have been extensively studied in the litterature and dedicated numerical methods have been developed to handle the challenging geometry of DFNs; see [28] for a review and the references therein. Considering extremely large DFNs is challenging in terms of

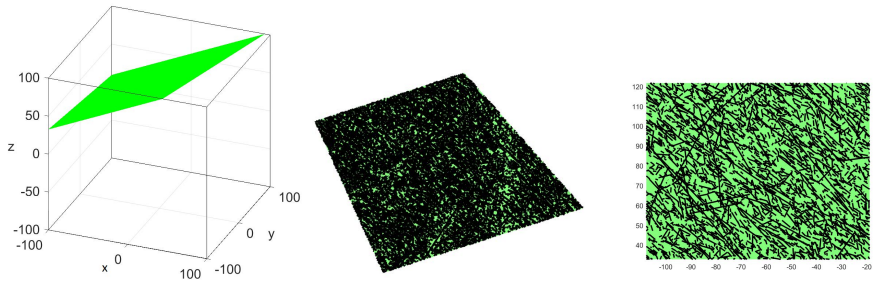

FIGURE 1.3. L200: (left) fracture (drawn in green) that contains the maximum number of intersections (11,710); (center) 2D view: intersections are drawn in black; (right) 2D view: zoom.

geometrical complexity, number of degrees of freedom and, therefore, computational ressources. The major difference between the proposed methods lies in the way the DFN is meshed.

- The first possibility is to consider a mesh that is conforming to the intersections between fractures. This can be realized in a matching way with nodes matching at the intersections [19, 26, 31, 32, 40] or in a nonmatching way with nodes from each fracture that may differ. This second situation arises, for example, when choosing a different mesh step from one fracture to another, and leads to the appearance of hanging nodes that can be managed either with mortar methods [41] or with polytopal (polygonal/polyhedral) discretization methods such as the virtual element method (VEM) [2] or the hybrid high-order (HHO) method [21, 22]. Flow simulations in DFNs using intersection-conforming meshes and polytopal discretization methods have been reported in [27] for VEM and [13, 30] for HHO.

- The alternative is to use a mesh that does not take the intersections into account while the numerical methods do. In the litterature, different methods have been proposed to handle intersections that may cut the mesh cells: with an optimization approach [5, 7], by means of the extended finite element method (XFEM) [25] or with VEM [3]. In [4], a node insertion procedure ensures a global conformity of the mesh used with VEM.

In this work, we focus on intersection-conforming meshes that are constrained by the intersections (first item above). The advantage of keeping the intersections explicitely in the mesh generation is that it allows one to attach unknowns to the intersection edges so that continuity conditions at these intersections are easier to impose. Moreover, we focus on HHO methods since they possess, in our view, various interesting features for flow simulation in DFNs. Indeed, HHO methods are locally conservative, they support polytopal meshes, their order of accuracy can be easily increased, and they are computationally effective owing to the use of static condensation. Moreover, HHO methods have been bridged to hybridizable discontinuous Galerkin methods in [16]. HHO methods have been applied to a broad range of applications in computational mechanics and beyond; we refer the reader to the two recent textbooks [15, 20] and the references therein. The main goal of the present work is to study the computational performance of HHO methods in the context of extremely large DFNs (over one million fractures). More particularly, we focus on the choice of basis functions, the trade-off between increasing the polynomial order and refining the mesh, and how to take advantage of polygonal cells to reduce the number of degrees of freedom.

The rest of this paper is organized as follows. Section 2 briefly recalls the mathematical model of single-phase flow in DFNs. Section 3 presents the extremely large DFNs considered herein, as well as a brief assessment of the quality of some generated meshes. Section 4 shortly presents the HHO method

applied to flow simulation in a DFN. Section 5 discusses our results concerning the computational performances of the HHO method on extremely large DFNs. Finally, conclusions and perspectives are given in Section 6.

2. The flow problem

In this work, fracture networks are modeled using the DFN approach proposed in [17, 18]. We consider a single-phase flow problem posed in a cubic domain of size L, where there are N_f intersecting fractures that form the computational domain. The rock matrix is assumed to be impervious. Let f_i be the i^{th} fracture, $i = 1, \ldots, N_f$. Every fracture is assumed to be planar. Let Γ_i be the boundary of the fracture f_i (which may be truncated by the cube). Let $\mathbf{x}$ be the local 2D coordinates in the fracture f_i. Let N_I be the total number of intersections between fractures, I_m be the m^{th} intersection, $m = 1, \ldots, N_I$, and S_m be the set of the indices of the fractures containing I_m. In our applications, it is unlikely to encounter intersection segments common to more than two fractures nor overlapping fractures. Therefore, the set S_m only consists of two indices, the indices of the two fractures that intersect. In each fracture f_i, we assume that the governing equations for the hydraulic head p_i (scalar-valued) and for the flux per unit length $\mathbf{u}_i$ (vector-valued) are the mass conservation equation and Darcy's law [37]:

$$\nabla \cdot \mathbf{u}_i(\mathbf{x}) = g_i(\mathbf{x}), \qquad \text{for } \mathbf{x} \in f_i, \tag{2.1}$$

$$\mathbf{u}_i(\mathbf{x}) = -\boldsymbol{\kappa}_i \nabla p_i(\mathbf{x}), \qquad \text{for } \mathbf{x} \in f_i. \tag{2.2}$$

The transmissivity field $\boldsymbol{\kappa}_i$ (unit $[\mathrm{m}^2.\mathrm{s}^{-1}]$) is taken as a positive-definite tensor that may be different from one fracture to another (in Table 1.1, the transmissivity is proportional to the identity and we report the scalar value). The function $g_i \in L^2(f_i)$ represents the sources/sinks. Boundary conditions are imposed on the cube faces. Let us denote by Γ_N the boundaries of the cube with Neumann boundary conditions and Γ_D the ones with Dirichlet boundary conditions with $\mathrm{meas}(\Gamma_D) > 0$. In every fracture, the boundary conditions are as follows:

$$p_i(\mathbf{x}) = p_i^D(\mathbf{x}), \qquad \text{for } \mathbf{x} \in \Gamma^D \cap \Gamma_i, \tag{2.3}$$

$$\mathbf{u}_i(\mathbf{x}) \cdot \mathbf{n} = q_i^N(\mathbf{x}), \qquad \text{for } \mathbf{x} \in \Gamma^N \cap \Gamma_i, \tag{2.4}$$

$$\mathbf{u}_i(\mathbf{x}) \cdot \mathbf{n} = 0, \qquad \text{for } \mathbf{x} \in \Gamma_i \backslash (\Gamma_D \cup \Gamma_N), \tag{2.5}$$

where $\mathbf{n}$ denotes the outward normal unit vector on Γ_N, p_i^D the Dirichlet load, and q_i^N is the Neumann load. As the rock matrix is supposed impervious, for every fracture i, one also imposes a homogeneous Neumann boundary condition on $\Gamma_i \backslash (\Gamma_D \cup \Gamma_N)$.

Additionally, coupling conditions hold at the intersections between the fractures [24, 38]. For each intersection I_m, $m = 1, \ldots, N_I$, let us consider the associated intersecting fractures f_j, $j \in S_m$. For all $j \in S_m$, the fracture f_j can have either one side (T-junction) or two sides (X-junction) at I_m. These sides are enumerated using a generic index $\sigma \in \Sigma_{j,m}$, where $\Sigma_{j,m}$ consists of a single element for a T-junction and of two elements for an X-junction. Let $\mathbf{n}_{k,m}^{\sigma}$ be the unit vector normal to I_m, pointing outward the side σ of f_j, and tangent to f_j. Then, for each intersection I_m, $m = 1, \ldots, N_I$, the following coupling conditions are enforced:

$$\exists p_m^0 \text{ s.t. } p_{j,m}^{\sigma} = p_m^0, \ \forall j \in S_m, \ \forall \sigma \in \Sigma_{j,m}, \tag{2.6}$$

$$\sum_{j \in S_m} \sum_{\sigma \in \Sigma_{j,m}} \mathbf{u}_j \cdot \mathbf{n}_{j,m}^{\sigma} = 0, \tag{2.7}$$

where $p_{j,m}^{\sigma}$ is the trace on I_m from the side σ of the hydraulic head in the fracture f_j. Condition (2.6) enforces the continuity of the hydraulic head at the fracture intersections and condition (2.7) imposes the conservation of mass.

3. Mesh generation

3.1. Algorithm

The mesh of the DFN is generated according to the algorithm described in [10, 35]. It is implemented in the C software `MODFRAC`[1]. Parallelism is achieved using posix-threads. The mesh step can be the same in the entire network, in which case the mesh is said to be matching at the intersections between fractures. One can also vary the mesh step from one fracture to another, in which case the mesh is generally nonmatching at the intersections between fractures. Figure 3.1 shows an example of a matching and an example of a nonmatching mesh.

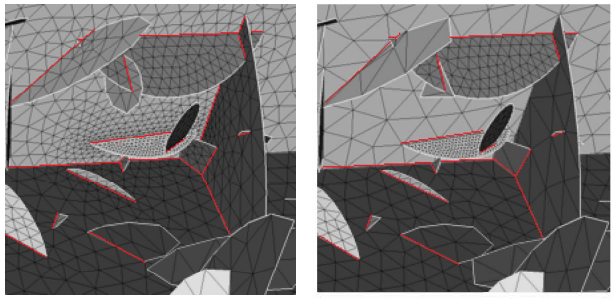

FIGURE 3.1. (left) A matching mesh at the fracture intersections; (right) A nonmatching mesh at the fracture intersections. The intersections are highlighted by red lines.

3.2. Notation

Let $\mathcal{T}_i$ be the mesh of the fracture f_i and let $\mathcal{T} = \bigcup_i \mathcal{T}_i$ be the mesh of the DFN. Let F denote a (boundary or interior) face of the mesh. Since the fractures are two-dimensional objects, a face F is actually an edge. For every fracture f_i, the interior faces are collected in the set $\mathcal{F}_i^{\circ}$, the intersection faces in the set $\mathcal{F}_i^{I}$ and the boundary faces in the set $\mathcal{F}_i^{\partial}$, so that the set

$$\mathcal{F}_i := \mathcal{F}_i^{\circ} \cup \mathcal{F}_i^{I} \cup \mathcal{F}_i^{\partial} \tag{3.1}$$

collects all the mesh faces contained in the fracture f_i. Let $\mathcal{F} = \bigcup_i \mathcal{F}_i$ be the set of mesh faces. Notice that every face that lies at the intersection between two fractures appears only once in the set $\mathcal{F}$. Moreover, we assume that the mesh faces are compatible with the partitioning of the domain boundary related to the Dirichlet and Neumann boundary conditions, so that we can consider the partition $\mathcal{F} = \mathcal{F}^D \cup \mathcal{F}^{\setminus D}$, where $\mathcal{F}^D$ (resp., $\mathcal{F}^{\setminus D}$) is the set of faces located in (resp., outside) Γ^D.

For a generic mesh cell $T \in \mathcal{T}$, its boundary is denoted by ∂T and its unit outward normal by $\boldsymbol{n}_T$, $\mathcal{F}_T$ denotes the collection of the mesh faces composing its boundary ∂T and for all $F \in \mathcal{F}_T$, $\boldsymbol{n}_{T|F}$ denotes the unit normal to F pointing outward T.

[1] `MODFRAC` is a proprietary software owned by Inria and the University of Technology of Troyes (UTT)

3.3. **Computational resources and mesh quality**

We perform the mesh generation using the software `MODFRAC` for the three test cases of Figure 1.1 on a Laptop Intel Core i7 4 cores CPU 32GiB RAM with 4 posix-threads. The quality $\delta(T)$ of a triangle T with edge lengths l_i, $i = 1, 2, 3$ and area $|T|$, is defined as in [9]:

$$\delta(T) := 4\sqrt{3}\frac{|T|}{l_1^2 + l_2^2 + l_3^2}. \tag{3.2}$$

The quality $\delta(T)$ is equal to 1 for an equilateral triangle. The lower $\delta(T)$, the worse the quality of T. Table 3.1 gives examples of a mesh and its properties for the three test cases L100, L150, L200 and also the mesh generation time.

Table 3.1. Examples of a mesh for the three DFNs: L100, L150 and L200.

DFN	#fractures	#Triangles	$\mathrm{Min}_T(\lvert T\rvert)$	Mean(δ)	Min(δ)	Time (hh:mm:ss)
L100	152k	2,770k	1.4e-09	0.66	3.10e-05	00:00:52
L150	508k	12,890k	1.16e-11	0.77	1.00e-04	00:02:25
L200	1,176k	20,522k	1.16e-11	0.63	9.25e-05	00:11:15

4. **The HHO method**

In this section, we describe how to apply the HHO method to DFN flow simulations. We recall that the DFN is composed of fractures f_i, $i = 1, \ldots, N_f$, each with its mesh $\mathcal{T}_i$ and (tensor-valued) transmissivity $\boldsymbol{\kappa}_i$. The flow problem (2.1)-(2.2) in the fracture f_i is a diffusion problem, and the fracture problems are coupled together according to the conditions (2.6)-(2.7).

4.1. **Local HHO spaces and operators**

The HHO method attaches unknowns to the mesh faces and to the mesh cells. HHO uses one polynomial of order $k \geq 0$ on each mesh face and one polynomial of order l on each mesh cell. At the algebraic level, the unknowns are therefore polynomial coefficients. We make the choice $l := k + 1$ for the cell polynomial degree, which offers the advantage of leading to a simpler stabilization operator than the equal-order choice $l := k$ (see [16]).

Let $\mathbb{P}_2^l$ be the space composed of divariate (real-valued) polynomials of total degree at most l. For every mesh cell $T \in \mathcal{T}$, $\mathbb{P}_2^{k+1}(T)$ denotes the space composed of the restriction to T of the polynomials in $\mathbb{P}_2^{k+1}$ and $\mathbb{P}_1^k(F)$ the univariate polynomial space attached to a mesh face $F \in \mathcal{F}_i$ (defined using an affine geometric mapping from the reference interval in $\mathbb{R}$ to F). The local HHO space in the mesh cell $T \in \mathcal{T}$ is

$$\hat{V}_T^k := \mathbb{P}_2^{k+1}(T) \times \mathbb{P}_1^k(\mathcal{F}_T), \qquad \mathbb{P}_1^k(\mathcal{F}_T) := \bigtimes_{F \in \mathcal{F}_T} \mathbb{P}_1^k(F)\,. \tag{4.1}$$

A generic element in $\hat{V}_T^k$ is denoted by $\hat{p}_T := (p_T, p_{\partial T})$ with $p_{\partial T} := (p_F)_{F \in \mathcal{F}_T}$.

The HHO method requires two main ingredients in every mesh cell $T \in \mathcal{T}$:

- a local reconstruction operator to reconstruct locally a gradient operator from the cell and the face unknowns;
- a stabilization term to ensure that the trace of the cell unknows and the face unknowns weakly match.

Let $i = 1, \dots, N_f$ be the index of the fracture to which T belongs, i.e., $T \in \mathcal{T}_i$ (notice that i is uniquely defined from T since every mesh cell belongs to one and only one fracture). Then, the local reconstruction operator $R_T : \hat{V}_T^k \to \mathbb{P}_2^{k+1}(T)$ is such that for all $\hat{p}_T := (p_T, p_{\partial T}) \in \hat{V}_T^k$, the function $R_T(\hat{p}_T) \in \mathbb{P}_2^{k+1}(T)$ is uniquely defined by the following equations:

$$(\boldsymbol{\kappa}_i \boldsymbol{\nabla} R_T(\hat{p}_T), \boldsymbol{\nabla} q)_{\boldsymbol{L}^2(T)} = (\boldsymbol{\kappa}_i \boldsymbol{\nabla} p_T, \boldsymbol{\nabla} q)_{\boldsymbol{L}^2(T)} + (p_{\partial T} - p_T, \boldsymbol{\nabla} q \cdot \boldsymbol{\kappa}_i \boldsymbol{n}_T)_{L^2(\partial T)}, \tag{4.2}$$

$$(R_T(\hat{p}_T), 1)_{L^2(T)} = (p_T, 1)_{L^2(T)}, \tag{4.3}$$

where (4.2) holds for all $q \in \mathbb{P}_2^{k+1}(T)^\perp := \{q \in \mathbb{P}_2^{k+1}(T) \mid (q,1)_{L^2(T)} = 0\}$. Moreover, the stabilization operator $S_{\partial T} : \hat{V}_T^k \to \mathbb{P}_1^k(\mathcal{F}_T)$ is such that for all $\hat{p}_T \in \hat{V}_T^k$,

$$S_{\partial T}(\hat{p}_T) := \Pi_{\partial T}^k \Big(p_{T|\partial T} - p_{\partial T} \Big), \tag{4.4}$$

where $\Pi_{\partial T}^k : L^2(\partial T) \to \mathbb{P}_1^k(\mathcal{F}_T)$ is the L^2-orthogonal projection defined such that for all $w \in L^2(\partial T)$,

$$(\Pi_{\partial T}^k(w) - w, q)_{L^2(\partial T)} = 0, \quad \forall q \in \mathbb{P}_1^k(\mathcal{F}_T). \tag{4.5}$$

Finally, the local bilinear form $a_T : \hat{V}_T^k \times \hat{V}_T^k \to \mathbb{R}$ is devised by setting

$$a_T(\hat{p}_T, \hat{w}_T) := (\boldsymbol{\kappa}_i \boldsymbol{\nabla} R_T(\hat{p}_T), \boldsymbol{\nabla} R_T(\hat{w}_T))_{\boldsymbol{L}^2(T)} + \frac{\rho_i}{\ell_T} (S_{\partial T}(\hat{p}_T), S_{\partial T}(\hat{w}_T))_{L^2(\partial T)}, \tag{4.6}$$

where $\rho_i := \rho(\boldsymbol{\kappa}_i)$ is the spectral radius of $\boldsymbol{\kappa}_i$ and $\ell_T := h_T$ is a local length scale associated with T and here taken equal to the diameter of T. (It is also possible to consider a specific length scale for each face $F \in \mathcal{F}_T$, but our numerical experiments do not indicate any advantage for this alternative choice).

4.2. **Assembly of the discrete problem in the DFN**

The HHO space in every fracture f_i, $i = 1, \dots, N_f$, is defined as follows:

$$\hat{V}_{h,i}^k := V_{\mathcal{T}_i}^{k+1} \times V_{\mathcal{F}_i}^k, \qquad V_{\mathcal{T}_i}^{k+1} := \bigtimes_{T \in \mathcal{T}_i} \mathbb{P}_2^{k+1}(T), \qquad V_{\mathcal{F}_i}^k := \bigtimes_{F \in \mathcal{F}_i} \mathbb{P}_1^k(F). \tag{4.7}$$

A generic element in $\hat{V}_{h,i}^k$ is denoted by $\hat{p}_{h,i} := (p_{\mathcal{T},i}, p_{\mathcal{F},i})$ with $p_{\mathcal{T},i} := (p_T)_{T \in \mathcal{T}_i}$ and $p_{\mathcal{F},i} := (p_F)_{F \in \mathcal{F}_i}$. As illustrated in Figure 4.1, the face unknowns are uniquely defined for all $F \in \mathcal{F}_i$. Similarly, the HHO space in the DFN is defined as follows:

$$\hat{V}_h^k := V_{\mathcal{T}}^{k+1} \times V_{\mathcal{F}}^k, \qquad V_{\mathcal{T}}^{k+1} := \bigtimes_{i=1,\dots,N_f} V_{\mathcal{T}_i}^{k+1}, \qquad V_{\mathcal{F}}^k := \bigtimes_{i=1,\dots,N_f} V_{\mathcal{F}_i}^k. \tag{4.8}$$

A generic element in $\hat{V}_h^k$ is denoted by $\hat{p}_h := (p_{\mathcal{T}}, p_{\mathcal{F}})$ with $p_{\mathcal{T}} := (p_{\mathcal{T},i})_{i=1,\dots,N_f}$ and $p_{\mathcal{F}} := (p_{\mathcal{F},i})_{i=1,\dots,N_f}$. Recall that every face F that lies at the intersection between two fractures appears only once in the set $\mathcal{F}$, so that the discrete unknowns attached to the face F appear only once in $V_{\mathcal{F}}^k$. This automatically enforces at the discrete level the condition (2.6) at any intersection. Moreover, to account for the Dirichlet boundary conditions on Γ^D, we consider the subspaces

$$\hat{V}_{h,0}^k := V_{\mathcal{T}}^{k+1} \times V_{\mathcal{F},0}^k, \qquad V_{\mathcal{F},0}^k := \{p_{\mathcal{F}} \in V_{\mathcal{F}}^k \mid p_F = 0, \ \forall F \in \mathcal{F}^D\}, \tag{4.9}$$

$$\hat{V}_{h,D}^k := V_{\mathcal{T}}^{k+1} \times V_{\mathcal{F},D}^k, \qquad V_{\mathcal{F},D}^k := \{p_{\mathcal{F}} \in V_{\mathcal{F}}^k \mid p_F = \Pi_F^k(p_{|F}^D), \ \forall F \in \mathcal{F}^D\}. \tag{4.10}$$

The discrete bilinear form $a_h : \hat{V}_h^k \times \hat{V}_h^k \to \mathbb{R}$ at the DFN level reads as follows:

$$a_h(\hat{p}_h, \hat{w}_h) = \sum_{i=1,\dots,N_f} \sum_{T \in \mathcal{T}_i} a_T(\hat{p}_T, \hat{w}_T), \tag{4.11}$$

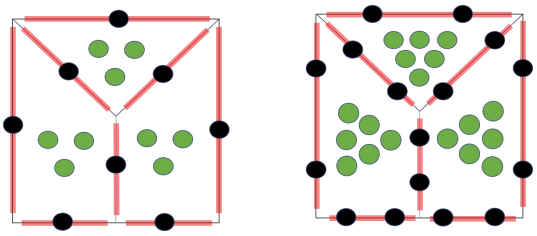

FIGURE 4.1. Discrete unknowns in three mesh cells. Here, the order of the cell polynomials is $l = k+1$, and k is the order of the face polynomials. (left) $k = 0$ and (right) $k = 1$, where black dots represent face unknowns and green dots cell unknowns (without necessarily meaning point evaluation).

where for all $i = 1, \ldots, N_f$ and all $T \in \mathcal{T}_i$, $\hat{p}_T$ (resp., $\hat{w}_T$) denotes the components of $\hat{p}_h$ (resp., $\hat{w}_h$) attached to the mesh cell T and to the faces in $\mathcal{F}_T$. The discrete problem is as follows:

$$\begin{cases} \text{Find } \hat{p}_h \in \hat{V}_{h,D}^k \text{ such that} \\ a_h(\hat{p}_h, \hat{w}_h) = \ell(w_{\mathcal{T}}), \quad \forall \hat{w}_h \in \hat{V}_{h,0}^k, \end{cases} \tag{4.12}$$

with $\ell(\hat{w}_h) := \sum_{i=1,\ldots,N_f} \big(\sum_{T \in \mathcal{T}_i} (g_i, w_T)_{L^2(T)} + \sum_{F \in \mathcal{F}_i \cap \Gamma^N} (q_i^N, w_F)_{L^2(F)} \big)$. As mentioned above, the condition (2.6) at any intersection between fractures is automatically enforced owing to the choice of the face unknowns, whereas the condition (2.7) is a consequence of the recovery of equilibrated fluxes as detailed in Subsection 4.3 (see in particular (4.20)).

At the algebraic level, the components of $\hat{p}_h$ attached to the Dirichlet boundary faces are eliminated, that is, we only seek for the components of $\hat{p}_h$ in

$$\hat{V}_{h,\mathcal{F}\setminus D}^k := V_{\mathcal{T}}^{k+1} \times V_{\mathcal{F}\setminus D}^k, \quad V_{\mathcal{F}\setminus D}^k := \bigtimes_{F \in \mathcal{F}\setminus D} \mathbb{P}_1^k(F). \tag{4.13}$$

Let $(\mathsf{P}_{\mathcal{T}}, \mathsf{P}_{\mathcal{F}}) \in \mathbb{R}^{N_{\mathcal{T}}^{k+1} \times N_{\mathcal{F}\setminus D}^k}$ be the corresponding component vectors of the discrete solution $\hat{p}_h := (p_{\mathcal{T}}, p_{\mathcal{F}}) \in V_{\mathcal{T}}^{k+1} \times V_{\mathcal{F}\setminus D}^k$(with obvious notation). Ordering the cell unknowns and then the face unknowns that are not Dirichlet, the algebraic realization of (4.12) is

$$\mathsf{A} \begin{bmatrix} \mathsf{P}_{\mathcal{T}} \\ \mathsf{P}_{\mathcal{F}} \end{bmatrix} = \begin{bmatrix} \mathsf{F}_{\mathcal{T}} \\ \mathsf{F}_{\mathcal{F}} \end{bmatrix} \quad \text{with} \quad \mathsf{A} := \begin{bmatrix} \mathsf{A}_{\mathcal{T}\mathcal{T}} & \mathsf{A}_{\mathcal{T}\mathcal{F}} \\ \mathsf{A}_{\mathcal{F}\mathcal{T}} & \mathsf{A}_{\mathcal{F}\mathcal{F}} \end{bmatrix}, \tag{4.14}$$

and the right-hand side results from the sink/source terms and the Dirichlet/Neumann boundary conditions. The matrix A is symmetric positive-definite. As the submatrix $\mathsf{A}_{\mathcal{T}\mathcal{T}}$ is block-diagonal, a computationally effective way to solve the linear system (4.14) is to eliminate locally the cell unknowns and solve first for the face unknowns only. Defining the Schur complement matrix

$$\mathsf{A}_{\mathcal{F}\mathcal{F}}^{\mathrm{S}} := \mathsf{A}_{\mathcal{F}\mathcal{F}} - \mathsf{A}_{\mathcal{F}\mathcal{T}} \mathsf{A}_{\mathcal{T}\mathcal{T}}^{-1} \mathsf{A}_{\mathcal{T}\mathcal{F}}, \tag{4.15}$$

the global transmission problem coupling all the face unknowns is

$$\mathsf{A}_{\mathcal{F}\mathcal{F}}^{\mathrm{S}} \mathsf{P}_{\mathcal{F}} = \mathsf{F}_{\mathcal{F}} - \mathsf{A}_{\mathcal{F}\mathcal{T}} \mathsf{A}_{\mathcal{T}\mathcal{T}}^{-1} \mathsf{F}_{\mathcal{T}}. \tag{4.16}$$

This linear system is only of size $N_{\mathcal{F}\setminus D}^k$. Once it is solved, one recovers locally the cell unknowns by using that $\mathsf{P}_{\mathcal{T}} = \mathsf{A}_{\mathcal{T}\mathcal{T}}^{-1}(\mathsf{F}_{\mathcal{T}} - \mathsf{A}_{\mathcal{T}\mathcal{F}} \mathsf{P}_{\mathcal{F}})$. The global transmission problem (4.16) being symmetric positive-definite, it can be solved with a direct solver based on Cholesky's factorization or an iterative solver based on the preconditioned Conjugate Gradient (CG) algorithm.

Remark 4.1. In practice, for extremely large DFNs, it may happen that the Cholesky algorithm fails with a "non positive-definite matrix" error. We suspect an effect of the use of floating-point numbers also linked to the possibly high condition number of the linear system matrix. An alternative which

proved successful in all our test cases run so far is to choose solvers based on LU factorization, like the Intel Pardiso LU solver from Intel MKL library [33].

4.3. **Recovery of equilibrated fluxes**

HHO is a locally conservative method for which it is possible to define equilibrated fluxes at the boundary of every mesh cell $T \in \mathcal{T}$. Let $i = 1, \ldots, N_f$ be the index of the unique fracture to which T belongs. Let $\hat{p}_h \in \hat{V}_{h,D}^k$ solve the discrete problem (4.12). We define the following fluxes associated with $\hat{p}_h$:

$$\phi_{\partial T}(\hat{p}_T) := -\boldsymbol{n}_T \cdot \boldsymbol{\kappa}_i \boldsymbol{\nabla} R_T(\hat{p}_T)_{|\partial T} + \frac{\rho_i}{\ell_T} \Pi_{\partial T}^k (p_{T|\partial T} - p_{\partial T}) \in \mathbb{P}_1^k(\mathcal{F}_T). \tag{4.17}$$

As shown in [16], these fluxes satisfy the following properties:

- Balance with the source term in every mesh cell $T \in \mathcal{T}_i$:

$$(\boldsymbol{\kappa}_i \boldsymbol{\nabla} R_T(\hat{p}_T), \boldsymbol{\nabla} q)_{\boldsymbol{L}^2(T)} + (\phi_{\partial T}(\hat{p}_T), q)_{L^2(\partial T)} = (g_i, q)_{L^2(T)}, \quad \forall q \in \mathbb{P}_2^{k+1}(T). \tag{4.18}$$

- Equilibrium at every mesh interface $F \in \mathcal{F}_i^\circ \cup \mathcal{F}_i^I$: (a) If F does not lie on the intersection between two fractures, then $F = \partial T_- \cap \partial T_+ \in \mathcal{F}_i^\circ$, and we have

$$\phi_{\partial T_-}(\hat{p}_{T_-})_{|F} + \phi_{\partial T_+}(\hat{p}_{T_+})_{|F} = 0. \tag{4.19}$$

  (b) If instead F lies on an intersection I_m for some $m = 1, \ldots, N_I$, then for all $j \in S_m$, denoting by $T_\sigma^j \in \mathcal{T}_j$, $\sigma \in \Sigma_{j,m}$, the mesh cells in the fracture f_j to which the face F belongs, we have

$$\sum_{j \in S_m} \sum_{\sigma \in \Sigma_{j,m}} \phi_{\partial T_\sigma^j}(\hat{p}_{T_\sigma^j})_{|F} = 0. \tag{4.20}$$

- Equilibrium with the Neumann boundary conditions: For all $i = 1, \ldots, N_f$ and all $F \in \mathcal{F}_i^\partial$ with $F = \partial T \cap \Gamma_i$, if $F \subset \Gamma_i \backslash (\Gamma_D \cup \Gamma_N)$, then

$$\phi_{\partial T}(\hat{p}_T)_{|F} = 0, \tag{4.21}$$

  whereas if $F \subset \Gamma_i \cap \Gamma_N$, then

$$\phi_{\partial T}(\hat{p}_T)_{|F} + \Pi_F^k(q_i^N) = 0. \tag{4.22}$$

5. **Study of computational performance**

In this section, we study the computational performance of HHO methods in the context of extremely large DFNs. We address the choice of the basis functions in Section 5.1, and the trade-off between increasing the polynomial order and refining the mesh in Section 5.2. In these two sections, triangular meshes are considered. We refer the reader to [30] for a comparison between HHO and lowest-order Raviart–Thomas methods on triangular meshes; here, we focus on the performance of HHO methods regarding basis functions and polynomial order. Section 5.3 contains numerical experiments in order to evaluate whether computational resources can be saved by means of polygonal cells. In this last section, the mesh in each fracture still matches the various intersections of the fracture, but its size is independent of the mesh of the other fractures, thus creating hanging nodes, and thereby polygonal cells, next to the fracture intersections.

The main numerical experiment to assess the accuracy of a simulation is the computation of the equivalent permeability of a DFN by running a permeameter test case. For example, to estimate the equivalent permeability K_x in the x-direction, we proceed as shown in Figure 5.1, i.e., we apply hydraulic heads p_1^D and p_2^D (with $\Delta p^D := p_1^D - p_2^D > 0$) on the two opposite cube faces along the x-direction and null flux boundary conditions on the other sides of the cube. The computed input flux

is denoted $Q_{\mathrm{in},x}$ (units $\mathrm{m}^3.s^{-1}$) and the computed output flux $Q_{\mathrm{out},x}$ (units $\mathrm{m}^3.s^{-1}$). Recall that the fluxes are computed as discussed in Subsection 4.3 and that, in the absence of round-off errors, we have $Q_{\mathrm{in},x} = Q_{\mathrm{out},x}$ since the HHO method is conservative. Then the equivalent permeability K_x is estimated as $K_x = \frac{Q_{\mathrm{in},x}}{L\,\Delta p^D}$.

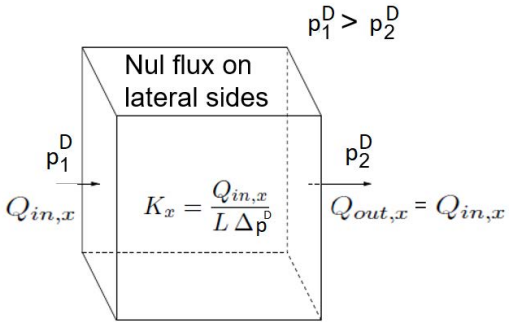

FIGURE 5.1. Permeameter boundary conditions along the x-direction.

Our implementation is made in the Inria C++17 software called `NEF++` [30] that relies on the `Eigen` library, which is a C++ template library for linear algebra [29] and the open-source HHO library `Disk++` [14]. Parametric studies have been carried out using the Python tool `Prune` (Inria). In all the test cases, the linear system is solved with the Intel Pardiso LU solver from Intel MKL library [33]. Except where indicated otherwise, all the simulations are carried out using the Inria cluster CLEPS on a partition with 4 Intel Xeon E7-4860 v2 with 12 cores, 2.6-3.2GHz and 3TB RAM. In all the tables reported in this section, the number of dofs is the one after static condensation.

5.1. **Choice of the HHO basis functions**

In the HHO method, the unknowns are polynomials coefficients, and there are various (classical) possibilities for choosing the basis functions. On every mesh face F, we choose the scaled monomials x_F^α with $\alpha = 0, \ldots, k$ and

$$x_F := 2\,\frac{(x - \bar{x}_F)}{|F|}, \tag{5.1}$$

where $\bar{x}_F$ denotes the midpoint of F and $|F|$ its length. On every mesh cell $T \in \mathcal{T}$, we compare two choices of scaled monomials:

- the basis composed of Cartesian monomials, hereafter called [`Cartesian`], and defined as $x_T^\alpha y_T^\beta$, with $\alpha, \beta = 0, \ldots, l = k+1$, $\alpha + \beta \le l$ and

$$x_T := 2\,\frac{(x - \bar{x}_T)}{h_x^T}, \qquad y_T := 2\frac{(y - \bar{y}_T)}{h_y^T}, \tag{5.2}$$

  where $(\bar{x}_T, \bar{y}_T)$ denote the coordinates of the barycenter of T and h_x^T, h_y^T denote the lengths of the bounding box of T according to the Cartesian axes;

- the basis composed of rotated monomials [1], hereafter called [`Rotated`], and defined as $\xi_T^\alpha \zeta_T^\beta$, with $\alpha, \beta = 0, \ldots, l = k+1$, $\alpha + \beta \le l$ and

$$\xi_T = 2\,\frac{(x - \bar{x}_T, y - \bar{y}_T)\cdot a_\xi^T}{h_\xi^T}, \qquad \zeta_T = 2\,\frac{(x - \bar{x}_T, y - \bar{y}_T)\cdot a_\zeta^T}{h_\zeta^T}, \tag{5.3}$$

Table 5.1. Choice of the cell basis functions and its influence on the relative error on the global mass conservation $|Q_{\mathrm{in},x} - Q_{\mathrm{out},x}|/Q_{\mathrm{in},x}$ on two DFNs

		$\lvert Q_{\mathrm{in},x} - Q_{\mathrm{out},x}\rvert/Q_{\mathrm{in},x}$	
DFN	k	[`Cartesian`]	[`Rotated`]
L100	0	0.00%	0.00 %
	1	0.00%	0.00 %
	2	0.08%	0.00 %
	3	135%	0.00 %
L150	0	0.00%	0.00%
	1	3e-3%	0.00%
	2	183%	0.00%
	3	691%	0.00%

where a_ξ^T, a_ζ^T denote the unit vectors aligned with the inertia axes of T and h_ξ^T, h_ζ^T the lengths of the bounding box of T according to the inertia axes.

Both the Cartesian and rotated monomials are independent of translation (since they use the barycenter of T) and of homothety (since they use the length of a suitable bounding box). Moreover, the rotated monomials are also independent of rotation.

To compare the two choices for the cell basis functions (Cartesian vs. rotated), we report in Table 5.1 the relative error on the global mass conservation indicator $|Q_{\mathrm{in},x} - Q_{\mathrm{out},x}|/Q_{\mathrm{in},x}$ for these two choices on the permeameter test case described above. We observe that both choices of basis functions behave well for both DFNs for $k = 0$ and for the smaller DFN, L100, up to $k = 2$. For L100 with $k > 2$ and for L150 with $k > 0$, global mass conservation (and also local mass conservation, omitted from the table for brevity) is ensured only for the rotated basis functions. The reason of the poor behavior of the Cartesian monomials is the presence of some highly deformed mesh cells which leads to highly ill-conditioned local matrices when computing the reconstruction and stabilization operators, as discussed, e.g., in [20]. In all the following simulations, we employ the rotated monomials.

Remark 5.1. Other choices for the basis functions are possible, such as an L^2-orthonormalised basis, which can help dealing with distorted elements (see [20, §B.1.1]). We have not tested this choice because it is computationally more demanding than the choice of rotated monomials, especially with a HHO library based on on-the-fly computations of the local matrices (as the one we are using).

5.2. **Trade-off between polynomial order and mesh refinement**

We investigate the trade-off between polynomial order and mesh refinement in the context of the permeameter test case described above. As a first step, we perform a flow computation on a fine mesh to obtain a reference value for the equivalent permeability K_x of the DFN. Then, we use this reference value to compute errors on the equivalent permeability obtained using coarser meshes.

In the first step of our study, two fine meshes are built with the software `MODFRAC`: one with 12,890,943 triangles for the DFN L150 and one with 20,522,575 triangles for the DFN L200. Figure 5.2 reports the x-equivalent permeability versus the number of dofs for polynomials degrees $k = 0, 1, 2, 3, 4$, and for the two DFNs L150 and L200. The equivalent permeability reaches a plateau as k increases. The reference values for K_x are taken for $k = 4$ and are equal to $K_x^{\mathrm{ref}} = 3.51 \times 10^{-2}$ $\mathtt{m}^2.\mathtt{s}^{-1}$ for L150 and $K_x^{\mathrm{ref}} = 3.53 \times 10^{-2}$ $\mathtt{m}^2.\mathtt{s}^{-1}$ for L200. Table 5.2 and Table 5.3 report the number of dofs, the number of non-zeros of the matrix of the linear system (nnz), the relative error (in %) on the equivalent permeability defined as $\mathrm{RelErr}(K_x) := |K_x - K_x^{\mathrm{ref}}|/K_x^{\mathrm{ref}}$ and the relative error (in %) on the global mass conservation $|Q_{\mathrm{in},x} - Q_{\mathrm{out},x}|/Q_{\mathrm{in},x}$ for the test cases L150 and L200 respectively. The computational times (total time T_{tot}) for the fine mesh flow simulations and the peak of RAM for

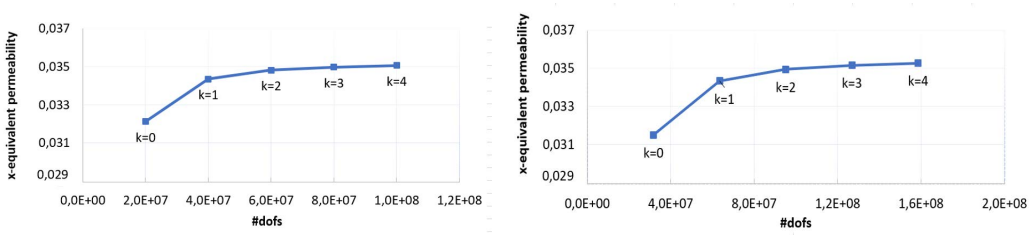

FIGURE 5.2. x-equivalent permeability versus #dofs. Left: L150 (mesh step 1: 12,890,943 triangles); right: L200 (mesh step 1.5: 20,522,575 triangles).

the two test cases L150 and L200 are given in Table 5.4 and 5.5 respectively. The postprocessing encompasses two steps: (1) the mean hydraulic head computations; (2) the flux recovery detailed in Subsection 4.3. These two steps are known to be embarrassingly parallel, i.e. they can be performed independently in each mesh cell, which means a potentially large gain in computational ressources with respect to the present serial implementation. The only difference is that our application is not really 2D since the 2D coordinates change from one fracture to another. Our current version of the code is based on a loop over the fractures. We could save computational time by an efficient parallelization of this loop. Notice also that to save RAM, neither $\mathsf{A}_{\mathcal{T}\mathcal{T}}^{-1}$ to compute $\mathsf{P}_{\mathcal{T}}$ (see Subsection 4.2), nor the local gradient or the stabilization matrix are stored. This is the reason why the postprocessing step has a sizable contribution to the costs.

TABLE 5.2. L150: Flow computation on a fine mesh.

k	#dofs	nnz	RelErr(K_x)	$\lvert Q_{\mathrm{in},x} - Q_{\mathrm{out},x}\rvert / Q_{\mathrm{in},x}$
0	19,999k	97,199k	8.41 %	5.34e-09 %
1	39,998k	388,798k	2.04 %	1.70e-09 %
2	59,997k	874,797k	0.72 %	1.23e-09 %
3	79,996k	1,555,194k	0.24 %	5.51e-08 %
4	99,996k	2,429,992k	0.00 %	1.54e-07 %

TABLE 5.3. L200: Flow computation on a fine mesh.

k	#dofs	nnz	RelErr(K_x)	$\lvert Q_{\mathrm{in},x} - Q_{\mathrm{out},x}\rvert / Q_{\mathrm{in},x}$
0	31,711k	154,704k	10.68 %	3.22e-09 %
1	63,422k	618,818k	2.64 %	9.85e-09 %
2	95,134k	1,392,340k	0.93 %	1.72e-08 %
3	126,845k	2,475,272k	0.30 %	9.28e-08 %
4	158,557k	3,867,613k	0.00 %	7.23e-08 %

TABLE 5.4. L150: Computational resources required for the flow computation on a fine mesh: total time T_{tot} (hh:mm), time of the different steps: Input, Assembly, Solver, Postprocessing (hh:mm and % of T_{tot} in parentheses) and peak of RAM (GiB).

k	#dofs	T_{tot}	Input	Assembly	Solver	Postprocessing	RAM
0	19,999k	00:46	00:06 (13%)	00:08 (18%)	00:04 (9%)	00:27 (58%)	78
1	39,998k	01:47	00:07 (7%)	00:22 (21%)	00:09(9%)	01:06 (62%)	176
2	59,997k	02:29	00:06 (4%)	00:32 (22%)	00:19 (13%)	01:31 (61%)	314
3	79,996k	03:31	00:06 (3%)	00:48 (23%)	00:35 (17%)	02:00 (57%)	508
4	99,996k	04:50	00:06 (2%)	01:10 (24%)	00:54(19%)	02:38 (55%)	738

TABLE 5.5. L200: Computational resources required for the flow computation on a fine mesh: total time T_{tot} (hh:mm), time of the different steps: Input, Assembly, Solver, Postprocessing (hh:mm and % of T_{tot} in parentheses) and peak of RAM (GiB).

k	#dofs	T_{tot}	Input	Assembly	Solver	Postprocessing	RAM
0	31,711k	01:21	00:12 (15%)	00:15 (19%)	00:07 (10%)	00:44(55%)	128
1	63,422k	02:49	00:10 (6%)	00:34 (21%)	00:22 (13%)	01:40 (59%)	297
2	95,134k	04:33	00:11 (4%)	01:00 (22%)	00:46 (17%)	02:34 (57%)	551
3	126,845k	06:08	00:12 (3%)	01:24 (23%)	01:16 (21%)	03:15 (53%)	869
4	158,557k	08:48	00:10 (2%)	01:54 (22%)	02:25 (28%)	04:16 (49%)	1279

Let us now investigate the trade-off between polynomial order and mesh refinement. To do so, simulations are performed on different coarser meshes of the DFN L150 (Table 5.6) and for each mesh, we compute the resulting equivalent permeability for different polynomial degrees $k \in \{0, 1, 2, 3, 4\}$.

TABLE 5.6. L150: different meshes.

DFN mesh step	#Triangles	#edges
2	7,619k	11,732k
1.5	9,720k	15,074k
1	12,890k	19,999k
0.6	28,205k	43,531k
0.5	39,364k	60,555k
0.4	60,536k	92,727k

Figure 5.3 displays the relative error using the reference value obtained above in the first step. Considering first the left panel based on the number of dofs (after static condensation), we observe that $k = 1$ is the most effective choice on the three coarsest meshes if an accuracy between 2% and 4% is desired, whereas for smaller accuracies, going for $k \geq 2$ is preferable. Considering the right panel based on the computational time leads essentially to similar conclusions. Similar conclusions are reached as well if one considers the peak of memory (results not shown for brevity). Actually, the optimal choice depends on the initial mesh as shown on Figure 5.4. It is computationnally less demanding to go for $k = 1$ on a fine mesh and to go for higher order $k \geq 2$ on a coarse mesh. In summary, as expected from the linear nature of the problem and the regularity of the solution, we recommend:

- to use a face polynomial order k at least equal to 1, instead of refining a mesh while keeping $k = 0$,
- to choose $k = 1$ on a fine mesh and $k > 1$ on a coarser mesh.

To provide some further insight on the relation between number of dofs, computational time and peak memory, we report in Table 5.7 the computational time and RAM required for approximately the same number of dofs with polynomial orders $k = 0, 1, 2, 3$. As expected, the time and RAM memory requirements increase with k. This is shown in Figure 5.5 where we plot the computational time and the peak of RAM versus the number of dofs. From Table 5.7 and Figure 5.5, we observe that the extra time and RAM memory for $k \leq 2$ seem negligible in comparison with the time and RAM at order $k = 0$. For $k > 2$, the slopes become somewhat higher. We believe that these increases in time and RAM for $k > 2$ are due to the solver since the matrices have more nonzero elements when k increases.

5.3. **How to take advantage of polygonal cells?**

So far, to get an accurate solution, the numerical simulations were performed on a fine mesh according to the standard procedure described in Algorithm 1 below (where the steps are named "FT" for fully

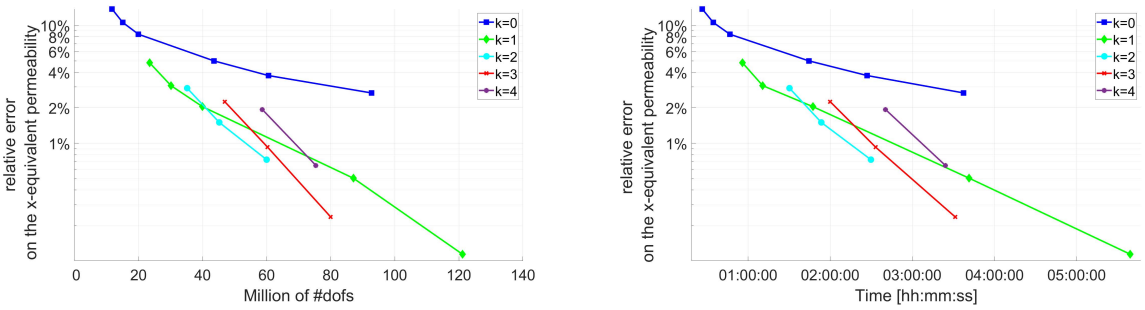

FIGURE 5.3. L150: relative error on the x-equivalent permeability versus #dofs (left) and versus computational time (hh:mm:ss, right).

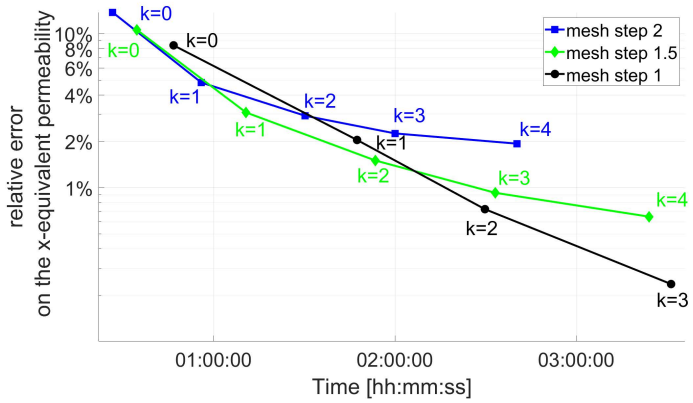

FIGURE 5.4. L150: relative error on the x-equivalent permeability versus computational time (hh:mm:ss).

TABLE 5.7. L150: Computational resources required for approximately a fixed number of dofs and for different polynomial orders $k = 0, 1, 2, 3$.

k	#dofs	Total time (hh:mm:ss)	RAM (GiB)	Relative error RelErr(K_x)	DFN mesh step (m)
0	43,531k	01:51:14	133	4.77 %	0.6
1	39,998k	01:47:26	176	2.04 %	1
2	45,222k	02:15:48	203	1.50 %	1.5
3	46,930k	02:18:27	253	2.25 %	2

triangular). In this section, we study how to change this procedure to take advantage of polygonal cells in order to (possibly) save computational resources.

Algorithm 1 Fully triangular mesh generation and flow computation

1: (FT1) Generate a fine mesh of the DFN with matching cells (in our case triangles) at the intersections
2: (FT2) Perform the corresponding flow simulation

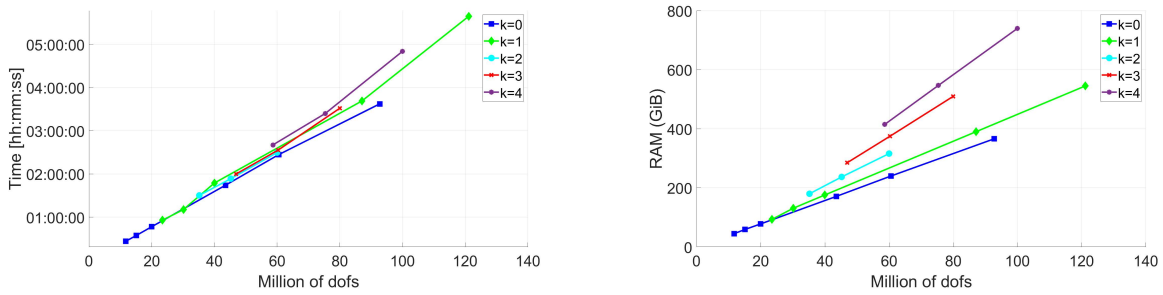

FIGURE 5.5. L150: computational time (left) and RAM requirements (right) versus #dofs.

5.3.1. *A mesh refinement/coarsening procedure*

In DFN, flows are highly channelled, meaning that only a small number of fractures carry most of the flow. A reasonable way to decrease the number of triangles in the mesh of a DFN is, therefore, to use a fine mesh only for those fractures that mostly contribute to the flow and to use a coarse mesh for the other fractures. To do so, one can use the fact that, by choosing a mesh step specific to each fracture, the mesh generator `MODFRAC` allows one to generate triangular meshes that are nonmatching at the intersections between fractures. Then, the resulting nonmatching mesh can be transformed into a polygonal matching mesh which is supported by the HHO method. The detailed steps of the new procedure are described in Algorithm 2. The resulting meshes are called "polygonal-triangular" meshes (since they consist of triangular cells away from the intersections and polygonal cells at the intersection between fractures meshes with different mesh steps). We use therefore the notation "PT" to name the steps of Algorithm 2. The polygonal-triangular mesh can be viewed as an intermediate mesh between the two fully triangular meshes where all the fractures are meshed with either a fine or a coarse triangular mesh. Notice that coarsening at step (PT1) is limited by the geometry and intersections of the fractures (above a given mesh step, no more coarsening is possible owing to geometric constraints).

Algorithm 2 Polygonal-triangular mesh generation and flow computation

1: (PT1) Generate a coarse mesh of the DFN
2: (PT2) Perform the coarse flow simulation
3: (PT3) Decide which fractures to refine according to their flow contribution
4: (PT4) Remesh the DFN according to step (PT3): only the selected fractures are refined and the mesh is therefore nonmatching at the intersections between fractures
5: (PT5) Run a node insertion algorithm to build a (matching) polygonal mesh at intersections: the mesh is therefore composed of triangles and polygons
6: (PT6) Perform the flow computation on the new polygonal mesh obtained at step (PT5)

In view of Algorithm 2, it appears that we need two additional ingredients with respect to the standard procedure of Algorithm 1:

- an algorithm to select the most contributing fractures at step (PT3);
- a node insertion algorithm to create polygons from nonmatching triangles at step (PT5).

These two ingredients are now described.

Selection of the most contributing fractures (step (PT3)). As proposed in [36], we characterize each fracture f_i by a single flow value Q_i defined as the total flow exchanged between f_i and its intersecting fractures:

$$Q_i := \frac{1}{2} \sum_{m=1}^{N_I^i} |Q_{i,m}|, \tag{5.4}$$

where N_I^i is the number of intersections in the fracture f_i, and $Q_{i,m}$ is the flow in f_i exchanged through the intersection m, $m = 1, \ldots, N_I^i$.

To know which fractures to refine/coarsen, we propose a basic idea consisting in two steps:

(1) for each fracture f_i, compute Q_i from the coarse flow simulation performed at step (PT2) and compute the maximum value

$$Q_{\max} := \max_{i=1,\ldots,N_f} (Q_i)\,; \tag{5.5}$$

(2) choose a fine mesh step for the fractures with $Q_i/Q_{\max}$ above a given threshold.

Creation of polygons from nonmatching meshes (step (PT5)). The second ingredient is an algorithm to create a polygonal mesh from a nonmatching triangular mesh. As illustrated in Figure 5.6, choosing a mesh step that can be different from one fracture to another generally leads to nonmatching cells at the intersections between fractures (notice that the two fractures do not lie on the same plane). One way to transform these nonmatching cells into polygonal cells is to use a vertex insertion algorithm (added vertices are displayed in red in Figure 5.6). The vertex insertion algorithm is described in Algorithm 3.

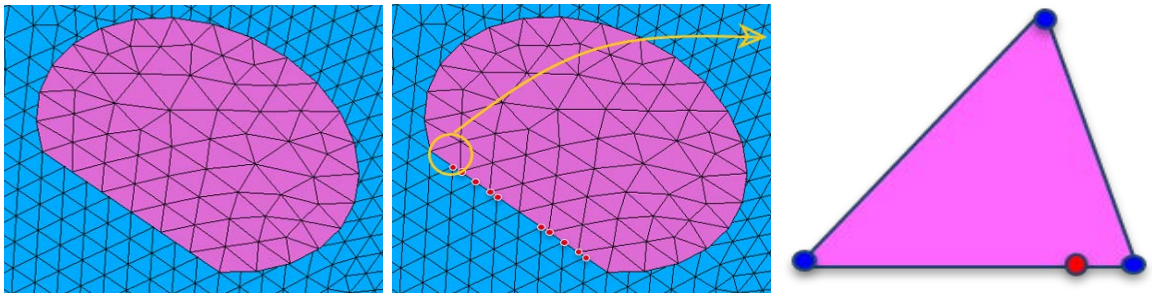

Figure 5.6. (left) Nonmatching triangular mesh; (center) Vertex insertion (red dots) to create a polygonal mesh; (right) Example of a triangle (initial vertices are in blue) transformed into a quadrangle with two aligned sides (the extra vertex is in red). This triangle/quadrangle is a zoom of the encircled triangle in the image at the center.

5.3.2. *Numerical tests*

Let us test the procedure proposed in Algorithm 2 for the DFN L150 of Figure 1.1. As the generated mesh is conforming to the intersections between the fractures and as the DFN L150 and L200 contain many intersections (see Table 1.1), the coarsening with triangles is limited. Therefore the coarse simulation remains quite costly. For this reason, we exploit the fact that the DFN L150 is embedded into the DFN L200 (Figure 1.1) and we reuse, for the test case DFN L200, the set of fractures selected at step (PT3) for the DFN L150. Then, we save steps (PT1-PT3) for the DFN L200 and we start Algorithm 2 at step (PT4). The potential price to pay is that some preferential flow paths

Algorithm 3 Vertex insertion algorithm

1: Load the nonmatching triangular mesh, including the edges and vertices of each intersection on both sides
2: For each intersection, compute the linear coordinates of the vertices on both sides in order to know where to insert the vertices to create a matching discretization
3: Find the unique set of intersection vertices and create a new global numbering of the vertices
4: Add the extra vertices to create the polygons (red dots) and update the connectivity table that defines the polygons in terms of vertices
5: Compute the 2D coordinates of the extra vertices in the fracture to which they now belong
6: Store the polygonal mesh

in the fractures that are present in L200 but not in L150 may be missed. Using this procedure is only considered here to save some computational resources, and is not a limitation of the present algorithms.

Step (PT1): Coarse mesh generation and mesh quality. The first step consists in running a coarse mesh generation with `MODFRAC` for the DFN L150. Let us choose a coarse mesh step of size 10 m. Table 5.8 gives the properties of the coarse mesh and also the mesh generation time.

TABLE 5.8. Matching coarse mesh for the DFN L150.

DFN	#fractures	#Triangles	$\text{Mean}_T(\lvert T\rvert)$	$\text{Min}_T(\lvert T\rvert)$	$\text{Mean}(\delta)$	$\text{Min}(\delta)$	Time (hh:mm:ss)
L150	508k	7,029k	0.47	1.16e-11	0.51	5.5e-05	00:01:53

Step (PT2): Coarse flow simulation. The coarse flow simulation is performed on a Laptop Intel Core i7 4 cores CPU 32GiB RAM with 4 posix-threads. It takes 6 minutes to load the coarse mesh plus 16 minutes to solve the coarse flow problem.

Step (PT3): Selection of the most contributing fractures. From the coarse simulation obtained at Step (PT2), we compute for each fracture f_i the quantity Q_i. Figure 5.7 reports the number of fractures versus $Q_i/Q_{\max}$ (%) (recall that the maximum flux $Q_{\max}$ is defined in (5.5)). Table 5.9 reports the number of selected fractures according to different choices for the threshold parameter.

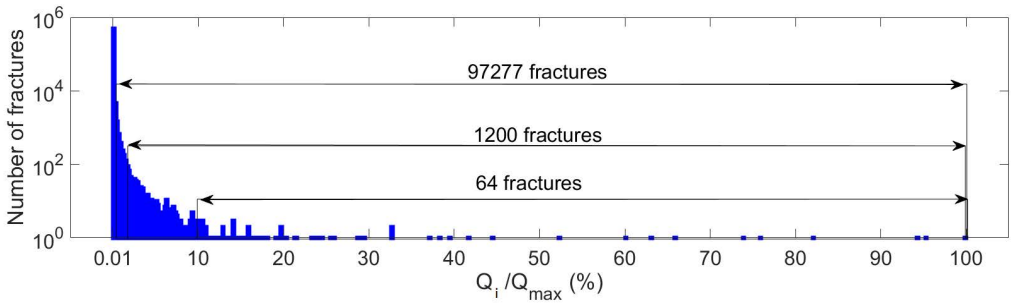

FIGURE 5.7. L150: number of fractures versus their percentage of the maximum output flux.

Table 5.9. L150: Number of selected fractures according to different thresholds.

Threshold	#fractures above the threshold	ratio of selected fractures wrt the DFN
10%	64	0.01%
1%	1,200	0.23%
0.01%	97,277	19%

Step (PT4): Generation of the nonmatching mesh (with triangles). Let us choose the threshold 0.01% in Table 5.9 (all other choices are possible). Then 97,277 fractures are selected as the most contributing fractures at Step (PT3). We use this set of fractures for the test case L150 (19% of the DFN) and also for the test case L200 (8% of the DFN) owing to the embedded feature of the considered DFNs. For the test L150, we set a mesh step of size 1m for the 97,277 selected fractures and we coarsen the other fractures with a mesh step of 2. For the test case L200, we set a mesh step of 1.5m for the 97,277 selected fractures and we coarsen the other fractures with a mesh step of 2m. Table 5.10 reports the properties of the nonmatching meshes and also the mesh generation time. The quality of the mesh is computed according to Subsection 3.3. For L150, we obtain a nonmatching mesh containing 10,461,407 triangles which is 19% triangles less than the fine mesh described in Table 3.1. For L200, the nonmatching mesh contains 18,648,084 triangles, which is 9% less that the fine mesh described in Table 3.1.

Table 5.10. Nonmatching meshes for the test cases L150 and L200.

DFN	#fractures	#Triangles	$\text{Mean}_T(\lvert T\rvert)$	$\text{Min}_T(\lvert T\rvert)$	Mean(δ)	Min(δ)	Time (hh:mm:ss)
L150	508k	10,461k	0.36	1.16e-11	0.69	1e-04	00:02:21
L200	1,176k	18,648k	0.47	1.16e-11	0.59	7.8e-05	00:11:51

Remark 5.2. The mesh generation time is nearly the same for the nonmatching mesh as for the matching mesh (compare with Table 3.1) despite a lower number of triangles. This comes from the fact that the intersections are discretized twice in the nonmatching case versus once in the matching case. Moreover, as the contours of the ellipses are discretized by polygonal lines, some automatic corrections happen more frequently for coarse discretization to ensure that all discrete intersections lie within the polygons [35]. Some code optimization/parallelization is currently under investigation in the nonmatching case to improve the mesh generation time.

Step (PT5): Vertex insertion algorithm to build the polygonal-triangular mesh. The vertex insertion procedure used in Algorithm 3 has been implemented in a Matlab code, named `NEF-flow-polygons` (Inria). It takes 00:17:11 to load the nonmatching mesh and to insert the vertices for the L150 test case and 01:11:47 for the L200 test case. In both cases, about half of the time is dedicated to loading the mesh data. Figure 5.8 shows the polygonal mesh with the red dots representing the inserted vertices. For L200, the red dots are rather inside the cube (since the selected fractures are the ones of the test L150 and therefore rather located inside the embedding cube). Hence, a smaller number of red dots are seen on the cube surface, and these dots belong to the largest fractures, common to the test cases L150 and L200. For L150, 3.5% cells of the polygonal-triangular mesh are polygons, whereas this number is 1% for L200.

Remark 5.3. The code `NEF-flow-polygons` is a Matlab prototype and is not fully optimized. Hence, the computational time to run Algorithm 3 can be improved, especially as there are several loops. A *C* version included in the mesh generator would be much more efficient (also as there would be no need to read the mesh data files).

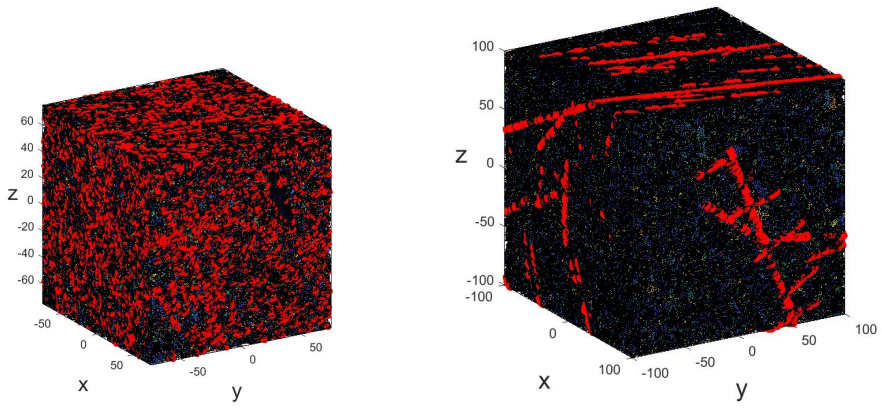

FIGURE 5.8. (left) L150: polygonal mesh; (right) L200 polygonal mesh. In both cases, the red dots represent the inserted vertices.

Step (PT6): Flow computation on the new polygonal-triangular mesh. The last stage is to run the flow simulations on the polygonal-triangular mesh built at step (PT5) using the `NEF++` software. Table 5.11 and Table 5.12 report the number of dofs, the number of non-zeros (nnz) in the linear system matrix, and the relative error on the equivalent permeability $\mathrm{RelErr}(K_x)$ and the relative error (in %) on the global mass conservation $|Q_{\mathrm{in},x} - Q_{\mathrm{out},x}|/Q_{\mathrm{in},x}$ for the test cases L150 and L200, respectively. We observe a slight loss in mass conservation by comparison with Table 5.2 and Table 5.3, respectively. This is due to a lower mesh quality of the nonmatching mesh by comparison with the initial fully triangular matching mesh (see the columns $\mathrm{Mean}(\delta)$ in Tables 3.1 and 5.10).

The computational time for the new polygonal-triangular mesh flow simulations and the peak of RAM for the two test cases L150 and L200 are given in Table 5.13 and 5.14, respectively.

TABLE 5.11. L150: Flow computation on the polygonal-triangular mesh.

k	#dofs	nnz	$\mathrm{RelErr}(K_x)$	$\lvert Q_{\mathrm{in},x} - Q_{\mathrm{out},x}\rvert/Q_{\mathrm{in},x}$
0	16,094k	80,799k	8.71 %	1.94e-09 %
1	32,189k	323,198k	2.27 %	2.62e-07 %
2	48,284k	727,196k	0.94 %	2.30e-06 %
3	64,379k	1,292,792k	0.44 %	1.05e-05 %
4	80,474k	2,019,989k	0.20 %	2.49e-05 %

TABLE 5.12. L200: Flow computation on the polygonal-triangular mesh.

k	#dofs	nnz	$\mathrm{RelErr}(K_x)$	$\lvert Q_{\mathrm{in},x} - Q_{\mathrm{out},x}\rvert/Q_{\mathrm{in},x}$
0	28,685k	141,415k	11.73 %	6.76e-10 %
1	57,370k	565,660k	3.26 %	5.39e-08 %
2	86,055k	1,272,735k	1.46 %	9.79e-07 %
3	114,740k	2,262,640k	0.81 %	5.21e-06 %
4	143,425k	3,535,376k	0.49 %	1.19e-05 %

5.3.3. *Discussion*

Let us compare the results obtained with Algorithm 1 with the ones obtained with Algorithm 2 on the two test cases L150 and L200.

Table 5.13. L150: Computational resources required for the flow computation on the new polygonal-triangular mesh: total time T_{tot} (hh:mm), time of the different steps: Input, Assembly, Solver, Postprocessing (hh:mm and % of T_{tot} in parentheses) and peak of RAM (GiB).

k	#dofs	T_{tot}	Input	Assembly	Solver	Postprocessing	RAM
0	16,094k	00:38	00:05 (14%)	00:06(18%)	00:03 (10%)	00:22 (58%)	65
1	32,189k	01:20	00:05 (6%)	00:17(22%)	00:09(12%)	00:47 (59%)	147
2	48,284k	02:07	00:05 (4%)	00:29 (23%)	00:17 (14%)	01:15 (59%)	273
3	64,379k	02:42	00:05(3%)	00:36 (23%)	00:29 (18%)	01:30 (56%)	434
4	80,474k	03:59	00:05 (2%)	00:55 (23%)	00:49 (21%)	02:08(53%)	641

Table 5.14. L200: Computational resources required for the flow computation on the new polygonal-triangular mesh: total time T_{tot} (hh:mm), time of the different steps: Input, Assembly, Solver, Postprocessing (hh:mm and % of T_{tot} in parentheses) and peak of RAM (GiB).

k	#dofs	T_{tot}	Input	Assembly	Solver	Postprocessing	RAM
0	28,685k	01:11	00:09 (14%)	00:11(17%)	00:07 (11%)	00:40(57%)	119
1	57,370k	02:28	00:09 (7%)	00:24 (17%)	00:19 (13%)	01:33 (63%)	269
2	86,055k	03:39	00:10 (5%)	00:50(23%)	00:39 (18%)	01:58 (54%)	498
3	114,740k	05:20	00:10 (3%)	01:10(22%)	01:13 (23%)	02:45 (52%)	807
4	143,425k	07:41	00:09 (2%)	01:38(21%)	02:00 (26%)	03:52 (50%)	1172

Figure 5.9 displays the relative error on the x-equivalent permeability with respect to the number of dofs (#dofs) for the test case L150 and for the three types of mesh: fully triangular coarse, polygonal-triangular and fully triangular fine. We observe that the intermediate polygonal-triangular mesh produces an approximation of the x-equivalent permeability as accurate as the one obtained on the fully triangular fine mesh, while saving $(k+1)*3{,}904{,}298$ dofs. This roughly represents 15% to 25% time saved and 13% to 16% RAM saved for a given polynomial degree k (compare Table 5.4 with Table 5.13).

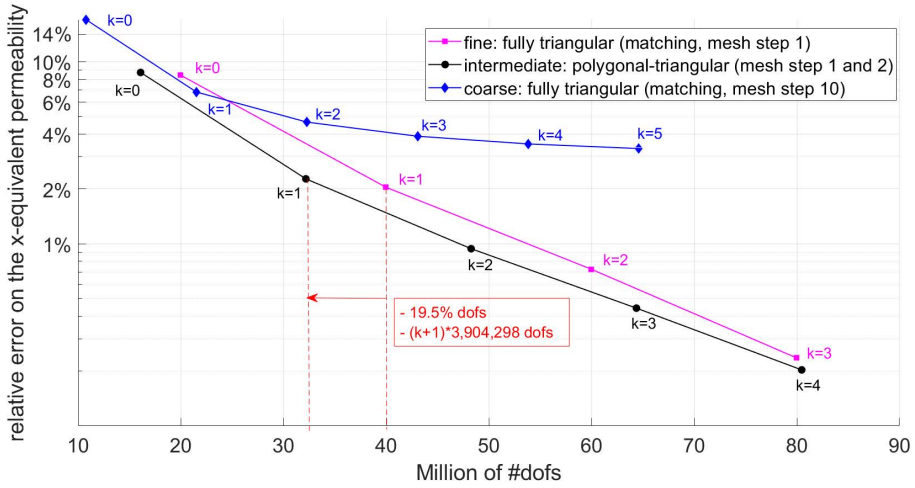

Figure 5.9. L150: relative error on the x-equivalent permeability versus #dofs. Example of gain obtained with a polygonal discretization.

For the test case L200, as shown in Figure 5.10, we observe again that the intermediate polygonal-triangular mesh produces an approximation of the x-equivalent permeability as accurate as the one obtained on the fully triangular fine mesh, while saving $(k+1)*3{,}026{,}232$ dofs. This represents roughly 12% to 20% time saved and 7% to 10% RAM saved for a given degree k (see Table 5.5 versus Table 5.14).

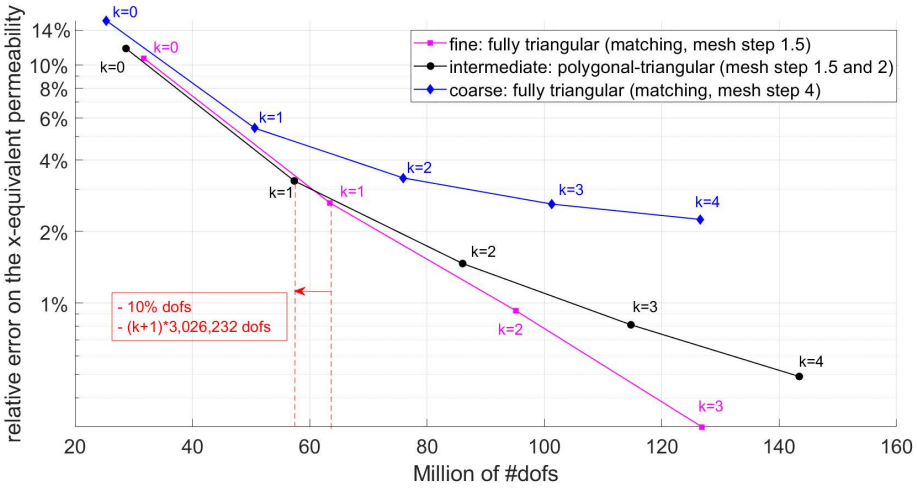

FIGURE 5.10. L200: relative error on the x-equivalent permeability versus #dofs. Example of gain obtained with a polygonal discretization.

These time savings deserve some further comments since a fair comparison requires to compare the overall time spent on steps (PT1)-(PT6) to the overall time spent on steps (FT1)-(FT2). In that case, the polygonal strategy is not the most relevant in terms of computational time, but the gain remains in terms of RAM memory requirements. Thus, the present results still provide a proof of concept that the polygonal feature of HHO helps in decreasing the number of dofs. Moreover, this study emphasizes that staying with triangles limits the coarsening possibilities (and therefore the computational gain) due to the geometry of the DFN and intersections. To further improve the results with HHO, general polygonal cells can be generated also away from the intersections between fractures, using a triangle agglomeration strategy for example. Also, a more advanced procedure based on local error estimators as proposed for example in [6] for VEM can be quite helpful. Furthermore, alternatives to reduce the computational cost can be considered. For example, steps (PT1-PT2-PT3) could be replaced by a single step that does not require any flow computation and based on recent applications of the graph theory for DFNs to know which fractures to coarsen and to refine [23, 34]. Also, step (PT5) could be included in the mesh generator (avoiding a costly mesh reload as it is the case now).

6. Conclusion

We have successfully tested the HHO method on large scale DFNs (more than 1 million of fractures). The implementation of the method is locally and globally conservative (whatever the polynomial degree). The use of basis functions based on hierarchical scaled rotated monomials is important to avoid roundoff errors due to ill-conditioning. Moreover, the use of high order polynomials is an advantage to compute a more accurate flow without much extra time compared to a low-order method, but the RAM memory demand increases with the polynomial order for the same number of dofs. We have also shown that the use of nonmatching meshes and the polygonal feature of the HHO method allow one to exploit the channelling effect of DFN flows by reducing the number of dofs, and therefore

the RAM requirements. Regarding the computational time, the strategy based on a mesh/coarsening procedure involving an additional coarse flow simulation is still costly. Other promising methods will be tested in a near future, especially the ones based on graph theory for DFN [23, 34] or a posteriori error estimates [6, 42]. Moreover, further reduction of the number of dofs by using the unfitted HHO method [11] and the possibility to merge the triangles in (possibly non convex) polygons can be considered. Finally, we are also interested in studying iterative solvers to reduce the RAM memory requirements [39].

Acknowledgements

The authors warmly thank Patrick Laug for his fruitful collaborative work regarding mesh generation. The authors are also grateful to the LabCom *fractory* (CNRS, Université de Rennes 1 and Itasca Consultants) for providing the geometry and transmissivity data for all the DFN considered in this paper. The present simulations presented in this paper were carried out using the Inria CLEPS experimental testbed (`https://paris-cluster-2019.gitlabpages.inria.fr/`). The authors warmly thank Simon Legrand (Inria, Paris) for his technical help with CLEPS.

References

[1] F. Bassi, L. Botti, A. Colombo, D. A. Di Pietro, and P. Tesini. On the flexibility of agglomeration based physical space discontinuous Galerkin discretizations. *J. Comput. Phys.*, 231(1):45–65, 2012.

[2] Lourenço Beirão da Veiga, Franco Brezzi, Luisa Donatella Marini, and Alessandro Russo. Mixed virtual element methods for general second order elliptic problems on polygonal meshes. *ESAIM, Math. Model. Numer. Anal.*, 50(3):727–747, 2016.

[3] Matías Fernando Benedetto, Stefano Berrone, Sandra Pieraccini, and Stefano Scialò. The virtual element method for discrete fracture network simulations. *Comput. Methods Appl. Mech. Eng.*, 280:135–156, 2014.

[4] Matías Fernando Benedetto, Stefano Berrone, and Stefano Scialò. A Globally Conforming Method for Solving Flow in Discrete Fracture Networks Using the Virtual Element Method. *Finite Elem. Anal. Des.*, 109(C):23–36, 2016.

[5] S. Berrone, S. Scialò, and F. Vicini. Parallel meshing, discretization, and computation of flow in massive discrete fracture networks. *SIAM J. Sci. Comput.*, 41(4):c317–c338, 2019.

[6] Stefano Berrone, Andrea Borio, and Alessandro D'Auria. Refinement strategies for polygonal meshes applied to adaptive VEM discretization. *Finite Elem. Anal. Des.*, 186, 2021.

[7] Stefano Berrone, Sandra Pieraccini, and Stefano Scialò. An optimization approach for large scale simulations of discrete fracture network flows. *J. Comput. Phys.*, 256:838–853, 2014.

[8] E. Bonnet, O. Bour, N. E. Odling, P. Davy, I. Main, P. Cowie, and B. Berkowitz. Scaling of fracture systems in geological media. *Reviews of Geophysics*, 39(3):347–383, 2001.

[9] Houman Borouchaki and Paul-Louis George. Quality mesh generation. *C. R. Acad. Sci., Paris, Sér. II, Fasc. b, Méc.*, 328(6):505–518, 2000.

[10] Houman Borouchaki, Patrick Laug, and Paul-Louis George. Parametric surface meshing using a combined advancing-front generalized Delaunay approach. *Int. J. Numer. Methods Eng.*, 49(1-2):233–259, 2000.

[11] Erik Burman and Alexandre Ern. An unfitted hybrid high-order method for elliptic interface problems. *SIAM J. Numer. Anal.*, 56(3):1525–1546, 2018.

[12] M. C. Cacas, E. Ledoux, G. de Marsily, B. Tillie, A. Barbreau, E. Durand, B. Feuga, and P. Peaudecerf. Modeling fracture flow with a stochastic discrete fracture network: calibration and validation: 1. The flow model. *Water Resources Research*, 26(3):479–489, 1990.

[13] Florent Chave, Daniele A. Di Pietro, and Luca Formaggia. A hybrid high-order method for passive transport in fractured porous media. *GEM. Int. J. Geomath.*, 10(1), 2019.

[14] M. Cicuttin, D. A. Di Pietro, and A. Ern. Implementation of discontinuous skeletal methods on arbitrary-dimensional, polytopal meshes using generic programming. *J. Comput. Appl. Math.*, 344:852–874, 2018.

[15] Mattheo Cicuttin, Alexandre Ern, and Nicolas Pignet. *Hybrid high-order methods. A primer with application to solid mechanics*. SpringerBriefs in Mathematics. Springer, 2021.

[16] Bernardo Cockburn, Daniele A. Di Pietro, and Alexandre Ern. Bridging the hybrid high-order and hybridizable discontinuous Galerkin methods. *ESAIM, Math. Model. Numer. Anal.*, 50(3):635–650, 2016.

[17] Philippe Davy, Romain Le Goc, and Caroline Darcel. A model of fracture nucleation, growth and arrest, and consequences for fracture density and scaling. *J. Geophys. Res. Solid Earth*, 118(4):1393–1407, 2013.

[18] Philippe Davy, Romain Le Goc, Caroline Darcel, Olivier Bour, Jean-Raynald de Dreuzy, and R. Munier. A likely universal model of fracture scaling and its consequence for crustal hydromechanics. *J. Geophys. Res. Solid Earth*, 115(B10), 2010.

[19] Jean-Raynald de Dreuzy, Géraldine Pichot, Baptiste Poirriez, and Jocelyne Erhel. Synthetic Benchmark for Modeling Flow in 3D Fractured Media. *Comput. Geosci.*, 50:59–71, 2013.

[20] Daniele A. Di Pietro and Jérôme Droniou. *The hybrid high-order method for polytopal meshes. Design, analysis, and applications*, volume 19. Springer, 2020.

[21] Daniele A. Di Pietro and Alexandre Ern. A hybrid high-order locking-free method for linear elasticity on general meshes. *Comput. Methods Appl. Mech. Eng.*, 283:1–21, 2015.

[22] Daniele A. Di Pietro, Alexandre Ern, and Simon Lemaire. An arbitrary-order and compact-stencil discretization of diffusion on general meshes based on local reconstruction operators. *Comput. Methods Appl. Math.*, 14(4):461–472, 2014.

[23] D. Doolaeghe, P. Davy, J. D. Hyman, and C. Darcel. Graph-based flow modeling approach adapted to multiscale discrete-fracture-network models. *Phys. Rev. E*, 102(5), 2020.

[24] Jocelyne Erhel, Jean-Raynald de Dreuzy, and Baptiste Poirriez. Flow simulation in three-dimensional discrete fracture networks. *SIAM J. Sci. Comput.*, 31(4):2688–2705, 2009.

[25] Luca Formaggia, Alessio Fumagalli, Anna Scotti, and Paolo Ruffo. A reduced model for Darcy's problem in networks of fractures. *ESAIM, Math. Model. Numer. Anal.*, 48(4):1089–1116, 2014.

[26] André Fourno, Tri-Dat Ngo, Benoit Noetinger, and Christian La Borderie. FraC: A new conforming mesh method for discrete fracture networks. *J. Comput. Phys.*, 376:713–732, 2019.

[27] Alessio Fumagalli and Eirik Keilegavlen. Dual virtual element method for discrete fractures networks. *SIAM J. Sci. Comput.*, 40(1):b228–b258, 2018.

[28] Alessio Fumagalli, Eirik Keilegavlen, and Stefano Scialò. Conforming, non-conforming and non-matching discretization couplings in discrete fracture network simulations. *J. Comput. Phys.*, 376:694–712, 2019.

[29] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. `http://eigen.tuxfamily.org`, 2010.

[30] Florent Hédin, Géraldine Pichot, and Alexandre Ern. Hho-DFN: a Hybrid High Order (HHO) method for the simulation of flow in large tridimensional Discrete Fracture Networks. In *Numerical Mathematics and Advanced Applications ENUMATH 2019*. Springer, 2019.

[31] Jeffrey D. Hyman, Carl W. Gable, Scott L. Painter, and Nataliia Makedonska. Conforming Delaunay triangulation of stochastically generated three dimensional discrete fracture networks: a feature rejection algorithm for meshing strategy. *SIAM J. Sci. Comput.*, 36(4):a1871–a1894, 2014.

[32] Jeffrey D. Hyman, Satish Karra, Nataliia Makedonska, Carl W. Gable, Scott L. Painter, and Hari S. Viswanathan. dfnWorks: A discrete fracture network framework for modeling subsurface flow and transport. *Comput. Geosci.*, 84:10–19, 2015.

[33] Intel. Mkl. `https://software.intel.com/en-us/mkl`, 2019.

[34] S. Karra, D. O'Malley, J. D. Hyman, H. S. Viswanathan, and G. Srinivasan. Modeling flow and transport in fracture networks using graphs. *Phys. Rev. E*, 97(3), 2018.

[35] Patrick Laug and Géraldine Pichot. Mesh Generation and Flow Simulation in Large Tridimensional Fracture Networks. In *MASCOT2018 - 15th Meeting on Applied Scientific Computing and Tools*, volume 22 of *IMACS Series in Computational and Applied Mathematics*, 2019.

[36] J. Maillot, P. Davy, R. Le Goc, C. Darcel, and J.-R. de Dreuzy. Connectivity, permeability, and channeling in randomly distributed and kinematically defined discrete fracture network models. *Water Resources Research*, 52(11):8526–8545, 2016.

[37] Vincent Martin, Jérôme Jaffré, and Jean E. Roberts. Modeling fractures and barriers as interfaces for flow in porous media. *SIAM J. Sci. Comput.*, 26(5):1667–1691, 2005.

[38] Jiří Maryška, Otto Severýn, and Martin Vohralík. Numerical simulation of fracture flow with a mixed-hybrid FEM stochastic discrete fracture network model. *Comput. Geosci.*, 8(3):217–234, 2004.

[39] Ani Miraçi, Jan Papež, and Martin Vohralík. A multilevel algebraic error estimator and the corresponding iterative solver with p-robust behavior. *SIAM J. Numer. Anal.*, 58(5):2856–2884, 2020.

[40] Tri-Dat Ngo, André Fourno, and Benoit Noetinger. Modeling of transport processes through large-scale discrete fracture networks using conforming meshes and open-source software. *Journal of Hydrology*, 554:66–79, 2017.

[41] G. Pichot, J. Erhel, and 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.

[42] Martin Vohralík. A posteriori error estimates for lowest-order mixed finite element discretizations of convection-diffusion-reaction equations. *SIAM J. Numer. Anal.*, 45(4):1570–1599, 2007.