

9

Embedded Discrete Fracture Models

DANIEL WONG, FLORIAN DOSTER, AND SEBASTIAN GEIGER

Abstract

Fractures are often implicitly represented in models used to simulate flow in fractured porous media. This simplification results in smaller models that are computationally tractable. As computational power continues to increase, there has been growing interest in simulation methods that explicitly represent fractures. The embedded discrete fracture model (EDFM) is one such method. In EDFM, fracture and matrix grids are constructed independently. The grids are then coupled to each other via source/sink relations. This modeling approach makes EDFM versatile and easy to use. EDFM has been shown to be able to handle complex fracture networks. The grid construction process is also straightforward and requires minimal fine-tuning. Within academia and industry, EDFM has been used to study geothermal energy production, unconventional gas production, multiphase flow in fractured reservoirs, and enhanced oil recovery processes. In this chapter, the mathematical formulation of EDFM is introduced. We then demonstrate the usage of EDFM via three examples. The first example involves a simple fracture network containing only three fractures. The second involves upscaling a stochastically generated fracture network. Finally, a well test will be simulated in a publicly available data set sourced from the Jandaira carbonate formation in Brazil.

9.1 Introduction

Naturally fractured reservoirs (NFRs), defined as reservoirs that contain naturally occurring fractures that can or are expected to play a significant role in fluid flow [31] are often encountered in the exploitation of hydrocarbon accumulations

and geothermal energy, as well as in the management of groundwater resources. In terms of hydrocarbon reserves, a significant amount of hydrocarbons is contained in NFRs. At the turn of the millennium, it was estimated that over 20% of the world's oil reserves are contained in NFRs [14]. In a study of 56 oil reservoirs, Jack and Sun [19] showed that recovery averaged at 26% but could range from a low of less than 10% to a high of over 60%. These figures suggest that there is huge incentive to properly manage NFRs in order to produce hydrocarbons sustainably, securely, and economically.

As with conventional reservoir management, reservoir simulations form an important component in making predictions about how NFRs may behave when produced. Such predictions enable us to evaluate and compare different field development and production strategies. This comparison then allows us to evolve and optimize the management of a field, maximizing the hydrocarbon recovery. Many mathematical methods have been developed to simulate flow in NFRs. One of the central features in these mathematical methods is how fractures are represented, implicitly or explicitly [4]. In methods that employ implicit fracture representation, fractures are converted into equivalent porous media via upscaling. These methods are often referred to as continuum methods. The models created through these methods are referred to as continuum models. Continuum models are computationally inexpensive to solve and can be used for full-field simulation studies. An example of a continuum model is the dual-porosity model, which represents fractures as a secondary porous medium that interacts with the matrix via transfer functions [2, 47].

Despite the advantages of continuum methods in terms of computational efficiency, their reliability is increasingly coming into question. Due to the multi-scale nature of fracture networks, which are often self-similar, the representation of fracture networks as a continuum may not be possible due to the lack of a representative elementary volume [3, 7]. In line with these findings, Elfeel et al. [13] demonstrated that, for multiscale fracture networks, upscaled equivalent permeability fields are dependent on chosen grid sizes. This discrepancy ultimately led to different outcomes in terms of production forecasting and history matching. Egeya et al. [11] showed that, contrary to what dual-porosity models might suggest, NFRs do not necessarily exhibit the dual-porosity signal in well tests. Moreover, various transfer functions have been proposed for the dual-porosity model. Choosing the appropriate one requires an understanding of the physical processes governing flow in NFRs. This understanding is often not available a priori [1, 26]. In view of these pitfalls, more accurate and robust simulation methods are required.

The response to the need for accurate simulation of flow in NFRs is the development of mathematical methods that explicitly represent a network of fractures in a

porous rock matrix. These methods are collectively known as discrete fracture and matrix (DFM) methods. The main feature of DFM methods is that they make minimal simplifications to the fractures being modeled and seek to stay as close to reality as possible. With the increase in computational power, as well as improvement of data acquisition methods, there has been a growth in applications and studies that make use of DFM methods. For example, Geiger and Matthäi [16] reviewed and assessed how DFM methods can be used to critically assess continuum methods to identify areas of improvement. Panfili and Cominelli [32] simulated miscible gas injection into an NFR with DFM to assess the risks that the presence of fractures pose. Bisdom et al. [6] used DFM methods for numerical upscaling of fracture networks to determine the impact of various aperture prediction methods. Egya et al. [12] studied well-test responses from various fracture networks and found that the infamous “dual-porosity dip” is not always observable. Hardebol et al. [17] used DFM simulations to study the impact of fracture network characterization on fluid flow. Vo et al. [45] compared DFM and dual-permeability models and identified a range of deficiencies in terms of the dual-permeability method’s ability to account for viscous displacement and spontaneous imbibition. Hui et al. [18] used a DFM method to simulate waterflooding in a full-field model in which the fracture network is simplified via a method known as edge collapse. In this simplification, short fracture segments with hanging nodes are removed to reduce the degrees of freedom involved in the simulations.

Various DFM methods exist because of different simplification choices [15]. The classical DFM method makes use of an unstructured grid that conforms to the fracture network geometry; fractures are then represented as subdimensional objects that are located at cell boundaries [21]. On the other hand, recent developments of DFM methods have focused on using grids that do not have to conform to the fracture network geometry. One such method is the extended finite-element method (XFEM), which enriches the finite-element solution space with basis functions that capture the effects of fractures [40]. Closely related to XFEM is the Lagrange multiplier method, which does not enrich the finite-element basis functions but instead accounts for matrix–fracture coupling using Lagrange multipliers [24, 39]. Alternatively, the embedded discrete fracture model (EDFM) relaxes the grid conformance requirement by treating fractures as line sources or sinks. EDFM uses a finite-volume discretization, thus making it compatible with conventional reservoir simulators. The decoupling of matrix and fracture grids in XFEM, Lagrange multiplier method, and EDFM permits the usage of structured grids, which are the de facto grid type used in geological modeling [25, 30].

The focus of this chapter is on EDFM, which is available in the MATLAB Reservoir Simulation Toolbox (MRST). EDFM was first introduced by Lee et al. [25]

and took inspiration from the representation of wells as sinks or sources; however, they only considered a few sparse and unconnected fractures within the simulation domain. Li and Lee [27] later extended the EDFM approach to account for a network of connected vertical fractures. Improvements were then made by Moinfar et al. [30] to account for inclined fractures. Flemisch et al. [15] benchmarked EDFM against a full-dimensional fractured model for single-phase flow and found a good match between the two. However, to our knowledge, no such validation has been performed for more complex flow phenomena that involve capillarity or gravity. Nevertheless, EDFM has been used in many studies, including that of Panfili and Cominelli [32] to simulate miscible sour gas injection into a fractured carbonate field; Siripatrachai et al. [43] and Zhang et al. [51] to understand the impact of capillary effects on hydrocarbon production in tight hydraulically fractured formations; Karvounis and Jenny [22] for simulating flow in enhanced geothermal systems; and Shakiba et al. [42] to analyze production uncertainties in hydraulic fracture networks characterized by microseismic data.

The strengths of EDFM over its other DFM counterparts is twofold. Firstly, it does not require conforming grids, thus allowing us to avoid meshing difficulties by using structured grids. Although preprocessing is required to establish the fracture–matrix and fracture–fracture connectivity relations, it is a much easier procedure. Any subsequent modifications such as changes in fracture apertures can be incorporated by applying multipliers on the transmissibility and porosity values. Secondly, EDFM uses the finite-volume discretization and two-point flux approximation (TPFA), making it naturally compatible with conventional reservoir simulators. Though TPFA is a popular formulation in many DFM methods, the combination of TPFA and nonconforming grids make the use of EDFM appealing. The main disadvantage of EDFM is that it is only suitable for conductive fractures. A study by [44] has shown that matrix pressure and saturation are continuous across fractures in EDFM simulations. Pressure and saturation continuity is not a major issue for fractures that are highly conductive relative to the matrix but introduces significant errors when fractures are sealed and form barriers to flow.

This chapter aims to introduce the reader to theory behind EDFM as well as the practical aspects of running EDFM simulations. In the following section, we introduce the concept of fracture permeability. Then, we present the mathematical formulation of EDFM. Following that, we present three examples of EDFM simulations along with detailed explanations. The first example involves two-phase flow in a porous medium containing three fractures. The second example shows how EDFM can be used to upscale a stochastically generated fracture network. Finally, the third example shows how EDFM can be used to simulate a well-test response in a fracture network mapped on an outcrop.

9.2 Fracture Permeability

To model flow in fractured porous media, one possible approach would be to define fractures as spaces in which flow would be modeled using the Navier–Stokes equation. However, it is a very challenging process to couple different mathematical models. Instead, we can represent the spaces as equivalent porous material with equivalent fracture permeabilities to simplify flow modeling within fractures. In the case of a conductive fracture, we can determine its equivalent permeability using its aperture,

$$k_f = \frac{a^2}{12f_c}, \quad (9.1)$$

where k_f refers to fracture permeability and a refers to fracture aperture. The correction factor f_c equals to 1 for laminar flow between parallel and smooth planes and is larger than 1 if the flow regime, fracture geometry, and surface roughness deviate from these ideal conditions. According to Darcy's law, volumetric flow rate is proportional to the product of k_f and the cross-sectional area of flow. The latter is proportional to a . As such, the volumetric flow rate is proportional to a^3 . As a result of this relationship, (9.1) is often called the cubic law. In a study performed by Witherspoon et al. [48], the cubic law was validated by performing flow tests on different rocks under various stress conditions. The study showed that the cubic law is valid regardless of the stress history undergone by a rock. The correction factor f_c was found to vary between 1.04 and 1.65 because of the surface roughness of the fractures. In actual practice, f_c is difficult to determine for subsurface rocks because of uncertainties surrounding fracture geometry and surface roughness. Because values for f_c are typically close to 1, f_c is usually assumed to be equal to 1 in the literature [8, 23, 29, 34, 37, 52].

The possibility of representing the transmissibilities of fractures by fracture permeabilities allows fracture networks to be incorporated within the framework of porous media flow. This is an important simplification that enables the use of DFM methods.

9.3 Mathematical Formulation

In this section, the mathematical formulation of EDFM will be reviewed. The mathematical formulation is based on the works of Lee et al. [25], Li and Lee [27], and Moïnfar et al. [30]. EDFM very much resembles the dual-porosity formulation, except that fractures are represented as subdimensional objects instead of a continuum and the transfer functions no longer assume a sugar cube geometry for matrix blocks. Mathematically, for a 3D system with N_f fractures, flow in the

fractures is formulated using N_f mass-conservation equations, with each fracture represented by an index i . For all $i \in [1, N_f]$,

$$\frac{\partial(\phi^i \rho_\alpha S_\alpha^i)}{\partial t} + \nabla \cdot (\rho_\alpha \vec{u}_\alpha^i) = \frac{\rho_\alpha}{a_i} \left[q_\alpha^i - q_\alpha^{i,0} - \sum_{j=1, j \neq i}^{N_f} q_\alpha^{i,j} \right], \quad \Omega_i \subset \mathbb{R}^2. \quad (9.2)$$

Flow in the matrix is modeled using a mass conservation equation as well. For convenience, the matrix is represented using an index $i = 0$. For $i = 0$,

$$\frac{\partial(\phi^0 \rho_\alpha S_\alpha^0)}{\partial t} + \nabla \cdot (\rho_\alpha \vec{u}_\alpha^0) = \rho_\alpha \left[q_\alpha^0 - \sum_{i=1}^{N_f} q_\alpha^{0,i} \right], \quad \Omega_0 \subset \mathbb{R}^3. \quad (9.3)$$

In (9.2) and (9.3), ϕ^i refers to the porosity of medium i . For fractures ($i \in [1, N_f]$), ϕ^i can be used to account for effects such as mineralization that reduce a fracture’s total aperture open to flow. For the matrix ($i = 0$), the pore spaces that are open to flow is ϕ^0 . The density of fluid phase α is ρ_α . The aperture of fracture i is a_i . The saturation of fluid phase α in medium i is S_α^i . The transfer function $q_\alpha^{i,0}$ (with a unit of $(\text{m}^3/\text{s})/\text{m}^2$) determines fluid flow from fracture i to the matrix, whereas $q_\alpha^{0,i}$ (with a unit of $(\text{m}^3/\text{s})/\text{m}^3$) flows in the opposite direction. Similarly, $q_\alpha^{i,j}$ (with a unit of $(\text{m}^3/\text{s})/\text{m}^2$) is the transfer function for flow from fracture i to fracture j . The source/sink terms in the matrix and fractures, q_α^0 and q_α^i , have units $(\text{m}^3/\text{s})/\text{m}^3$ and $(\text{m}^3/\text{s})/\text{m}^2$, respectively.

All of the fracture equations (9.2) are coupled to the matrix equation through $q_\alpha^{i,0}$. The fracture equations are coupled with each other through $q_\alpha^{i,j}$; however, because fractures do not always intersect each other, $q_\alpha^{i,j} = 0$ for some fracture pairs. Finally, \vec{u}_α^i is the velocity of fluid phase α in medium i and is modeled using Darcy’s law,

$$\vec{u}_\alpha^i = -\frac{k_{r\alpha}^i}{\mu_\alpha} \mathbf{K}^i \nabla(p_\alpha^i - \rho_\alpha^i \vec{g}), \quad (9.4)$$

for all $i \in [0, N_f]$. Here, $k_{r\alpha}^i$ and p_α^i refer, respectively, to the relative permeability and pressure of fluid phase α in medium i ; μ_α is the viscosity of fluid phase α ; \mathbf{K}^i is the permeability of medium i ; and \vec{g} is the acceleration due to gravity.

In the EDFM literature, $q_\alpha^{i,0}$ and $q_\alpha^{i,j}$ are not defined in continuous space; instead, they are constructed within the context of the finite-volume discretization [27, 30].

Given a matrix grid and fracture network (Figure 9.1a), the fractures are also discretized such that each fracture cell corresponds to one matrix cell (Figure 9.1b).

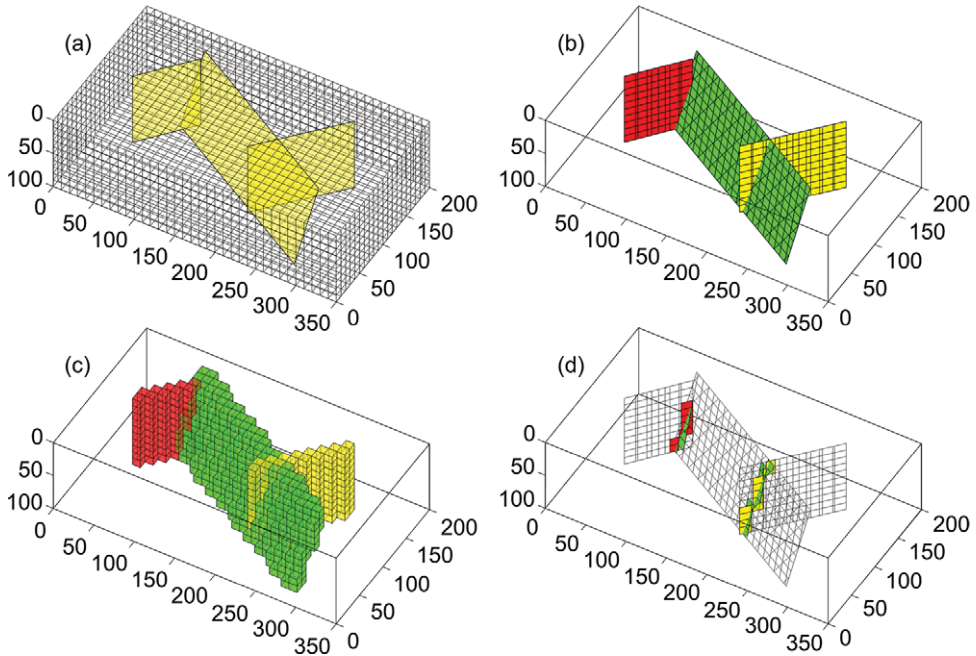


Figure 9.1 Construction of an EDFM. (a) Structured matrix grid and fracture network. (b) Fracture grid. (c) Matrix cells color-coded according to fractures they are connected to. (d) Fracture cells that are connected to each other.

For each matrix–fracture cell pair (m, f) , the movements of the fluid phases are modeled via TPFA as follows:

$$Q_{\alpha}^{m,f} = \int_{V_m} q_{\alpha}^{m,f} dV = T^{mf} \frac{k_{r\alpha}}{\mu_{\alpha}} (p_{\alpha}^m - p_{\alpha}^f), \tag{9.5}$$

$$Q_{\alpha}^{f,m} = \int_{A_f} q_{\alpha}^{f,m} dA = T^{mf} \frac{k_{r\alpha}}{\mu_{\alpha}} (p_{\alpha}^f - p_{\alpha}^m), \tag{9.6}$$

where $Q_{\alpha}^{m,f}$ is the volumetric flow rate from the matrix cell to the fracture cell, whereas $Q_{\alpha}^{f,m}$ is the volumetric flow rate from the fracture cell to the matrix cell. The two are related by $Q_{\alpha}^{m,f} = -Q_{\alpha}^{f,m}$. The matrix and fracture cell domains are respectively V_m and A_f . The mobility term $k_{r\alpha}/\mu_{\alpha}$ is upwind weighted. The transmissibility T^{mf} is unique to each matrix–fracture cell pair and is defined as

$$T^{mf} = \frac{k^{mf} A^{mf}}{\langle d \rangle^{mf}}, \tag{9.7}$$

where k^{mf} is the pore volume weighted harmonic average of fracture and matrix permeabilities, whereas A^{mf} is the fluid exchange area between the matrix–fracture cell pair. The average normal distance $\langle d \rangle^{mf}$ between points in the matrix and the fracture cell can be calculated as

$$\langle d \rangle^{mf} = \frac{\int_{V_m} |(\vec{x} - \vec{x}_{ref}) \cdot \vec{n}_f| \, dV}{\int_{V_m} dV}, \quad (9.8)$$

where $\vec{x} \in V_m$ are points in the matrix cell, \vec{x}_{ref} is any reference point on the fracture plane, and \vec{n}_f is the unit normal vector of the fracture plane. $\langle d \rangle^{mf}$ can be numerically approximated by subgridding a matrix cell and calculating the weighted average of normal distances from each matrix subcell's centroid to the fracture cell. The weights are the volumes of each matrix subcell. Note that it is possible for multiple fracture cells from different fractures to be within the same matrix cell. If fractures intersect with each other, there will be at least two fracture cells within one matrix cell. If two fractures are not intersecting but located close to each other, there may also be multiple fracture cells within each matrix cell. The latter case should be avoided by reducing matrix cell size, particularly for multiphase flow simulations where numerical dispersion can be severe. Also noteworthy is that at fracture tips, fracture cells may not be through-going within matrix cells. In such a case, the fluid exchange area A^{mf} will be reduced according to the size of the fracture cell.

For every two fracture cells ($f1, f2$) that intersect with each other, the fluid exchange is modeled by

$$Q_\alpha^{f1, f2} = \int_{A_{f1}} q_\alpha^{f1, f2} \, dA = T^{ff} \frac{k_{r\alpha}}{\mu_\alpha} (p_\alpha^{f1} - p_\alpha^{f2}), \quad (9.9)$$

$$Q_\alpha^{f2, f1} = \int_{A_{f2}} q_\alpha^{f2, f1} \, dA = T^{ff} \frac{k_{r\alpha}}{\mu_\alpha} (p_\alpha^{f2} - p_\alpha^{f1}), \quad (9.10)$$

where the transmissibility T^{ff} is zero for nonintersecting fracture cell pairs and positive otherwise. To determine T^{ff} , the half-transmissibilities of each fracture cell in the pair ($f1, f2$) have to be calculated:

$$T_{1/2}^{fi} = \frac{k^{fi} a^{fi} L_\cap^{f1, f2}}{d_\cap^{fi}}, \quad \forall i \in \{1, 2\}. \quad (9.11)$$

Here, k and a are the permeability and aperture of the fracture cell, L_\cap is the length of the intersection line between the two cells, and d_\cap is the average normal distance from the centers of the subsegments to the intersection line (Figure 9.2).

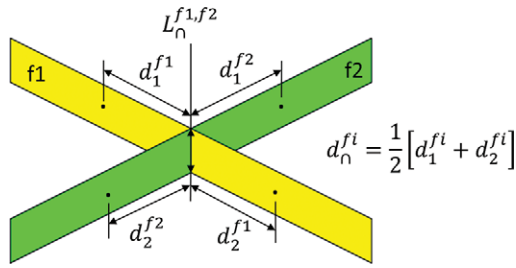


Figure 9.2 Geometrical input required for the calculation of fracture half-transmissibilities.

Each subsegment of a fracture cell lies on one side of the intersection line. If the intersection line does not extend fully across a fracture cell, the subsegments of the fracture cell can be established by extrapolating the intersection line to the edge of the fracture cell. The calculated half-transmissibilities then enable us to determine T^{ff} as the harmonic average of the two:

$$T^{ff} = \frac{T_{1/2}^{f1} T_{1/2}^{f2}}{T_{1/2}^{f1} + T_{1/2}^{f2}} \tag{9.12}$$

Note that EDFM only caters to the intersection of two fractures. For intersections involving more than two fractures, connectivity among the fractures can still be established by assigning transmissibility values for fracture cell pairs. However, to our knowledge, this scenario has not been addressed in the literature. In our work, we will only be considering intersections between two fractures.

In summary, the EDFM method involves three main steps:

1. Construction of a fracture grid based on a preexisting matrix grid (Figure 9.1b).
2. Connect intersecting matrix–fracture cell pairs using (9.7) (Figure 9.1c).
3. Connect intersecting fracture–fracture cell pairs using (9.12) (Figure 9.1d).

9.4 Hierarchical Fracture Model Module

EDFM has been implemented in the open-source MATLAB Reservoir Simulation Toolbox (MRST) under the `hfm` module. The initial implementation was by Shah et al. [41] in release 2016b and was mainly developed for 2D simulations. In release 2017b, we upgraded the `hfm` module to include full 3D capabilities based on the method developed by Li and Lee [27] and Moïnfar et al. [30]. The `hfm` module in MRST serves as a preprocessor that creates a global grid along with a list of

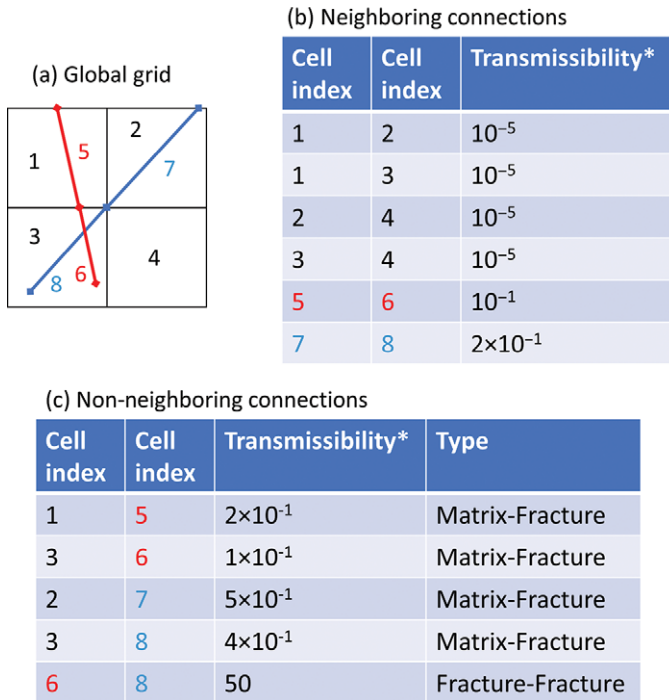


Figure 9.3 Difference between neighboring and non-neighboring connections. (a) A simple global grid consisting of four matrix and four fracture cells, (b) a list of neighboring connections and the associated transmissibilities, and (c) a list of non-neighboring connections that account for matrix–fracture and fracture–fracture fluid exchanges. Note that the transmissibility values are entirely fictional and for illustrative purposes only.

non-neighboring connections (NNCs) that are compatible with MRST's black-oil simulator framework from `ad-core` and `ad-blackoil`. The construction of the global grid is performed through the `EDFMgrid` function. The calculation~of the NNCs is performed using the `fracturematrixNNC3D` and `fracturefractureNNCs3D` functions. The NNCs refer to the matrix–fracture and fracture–fracture connections; although the interacting cells are physically next to each other, in the simulation, they are not treated as physical neighbors and, as such, require NNCs to be connected. A simple example is shown in Figure 9.3, where NNCs have been computed and tabulated; such NNC lists are commonly used by reservoir simulators.

The current 3D implementation of EDFM in the `hfm` module is capable of handling conductive fractures that are represented using convex 2D polygons in any orientation. However, the matrix grid is limited to orthogonal grids in a cuboid domain. Such a matrix grid can be constructed using the `cartGrid` and

tensorGrid functions in MRST. These functions have been extensively covered in section 3.1 of the MRST textbook [28]. For readers keen to implement EDFM on corner-point grids, we refer the reader to Xu and Sepehrnoori [49]. In the hfm module, some aspects of the algorithm have been parallelized. In particular, fracture grid construction is implemented such that each fracture within a network can be handled by one parallel processor. The matrix–fracture NNC calculations are also treated the same way. For the fracture–fracture NNCs, the calculations are separated by fracture pairs for parallelization. The module can also be used in serial mode if the MATLAB Parallel Processing Toolbox is not available. In the next section, the workflow for setting up an EDFM simulation is described.

9.5 Two-Phase Flow through a Simple Fracture Network

In this section, we demonstrate how the 3D EDFM implementation in MRST works. The MATLAB script for this section, titled `Example_1_manualEDFM.m`, is available in the hfm module. This example involves simulating two-phase flow through a domain containing three intersecting fractures using the hfm and ad-blackoil modules in MRST. We refer readers unfamiliar with the ad-blackoil module to chapters 11 and 12 in the MRST textbook [28].

In our workflow, the hfm module can be thought of as a preprocessor for the ad-blackoil module. The preprocessor takes as inputs an orthogonal matrix grid and a database of fractures. The orthogonal matrix grid can be created using the cartGrid function. Rock properties can, for instance, be prescribed using the makeRock function:

```

%% SET UP A STRUCTURED MATRIX GRID
physdim = [350, 200, 100]; % 350m x 200m x 100m domain
celldim = [35, 20, 10]; % 10m x 10m x 10m grid cell sizes
G = cartGrid(celldim, physdim);
G = computeGeometry(G);
G.rock = makeRock(G,100*milli*darcy,0.3); % km=100mD, matrix porosity = 0.3

```

The database of fractures is contained in a struct array; each fracture is characterized by a set of vertices, a permeability value, aperture, and porosity value (usually set to 1). In this example, three fractures are created. All three fractures span the full vertical depth of the matrix grid. The first and third fractures are vertical, whereas the second fracture is inclined. Once the setup in Listing 9.1 has been executed, the following code produces the plot in Figure 9.1a:

```

plotfracongrid(G,fracplanes);
view(30,45); axis equal tight

```

Listing 9.1 Setup of model with three intersecting fractures.


```

%% SET UP FRACTURE 1 (RED)
fracplanes(1).points = [40 100 0;
                       90 160 0;
                       90 160 100;
                       40 100 100]; % Vertices
fracplanes(1).aperture = 1/25;
fracplanes(1).poro = 0.8;
fracplanes(1).perm = 10000*darcy;

%% SET UP FRACTURE 2 (GREEN)
points = [80 160 0;
          290 40 0;
          290 40 100;
          80 160 100]; % Vertices
f2normal = getnormal(points);
points([1,2], :) = points([1,2], :) - f2normal * 15; % displace top points
points([3,4], :) = points([3,4], :) + f2normal * 15; % displace bottom points
fracplanes(2).points = points;
fracplanes(2).aperture = 1/25;
fracplanes(2).poro = 0.8;
fracplanes(2).perm = 10000*darcy;

%% SET UP FRACTURE 3 (YELLOW)
fracplanes(3).points = [200 70 0;
                       280 160 0;
                       280 160 100;
                       200 70 100]; % Vertices
fracplanes(3).aperture = 1/25;
fracplanes(3).poro = 0.8;
fracplanes(3).perm = 10000*darcy;

```



Now that the matrix grid and fracture database have been created, the fracture grid can be constructed. This is usually done by passing these two objects to the `EDFMgrid` function:

```
[G, fracplanes] = EDFMgrid(G, fracplanes, 'Tolerance', 1e-6);
```

which creates a global grid containing both matrix and fracture grid cells. However, in this example, we will describe in detail all of the lower-level commands necessary to construct the grid manually for pedagogical reasons. The code in Listing 9.2 generates three fracture grids contained in `Fgrid`. Each fracture grid corresponds to one fracture plane defined in `fracplanes`. The main function in the code snippet is `pebiAABBintersect`, which takes as inputs the vertex coordinates of a 3D polygon (in this case the vertex coordinates of a fracture plane) and the vertex coordinates of a cube (in this case the nodal coordinates of a matrix cell).

Listing 9.2 *Generate fracture grid.*

```

tol = 1e-5;
Nm = G.cells.num;
for i=1:length(fracplanes)
    points = fracplanes(i).points;
    aperture = fracplanes(i).aperture;

    % Calculate plane unit normal
    diffp = diff(points,1);
    planenormal = cross(diffp(1,:), diffp(2,:));
    planenormal = planenormal/norm(planenormal);

    % Instantiate data types to hold fracture grid information
    fraccellpoints = cell(Nm,1); % vertices of each fracture grid cell
    area = -1*ones(Nm,1); % area of each fracture grid cell

    % Calculate intersection of fracture with each matrix grid cell
    for j = 1:Nm
        [cn,cpos] = gridCellNodes(G,j);
        [~,area(j),~,~,fraccellpoints{j}] = ...
            pebiAABBintersect(points,G.nodes.coords(cn,:),tol);
    end

    % Consolidate intersection data
    intersected = ~cellfun('isempty',fraccellpoints);
    fraccellpoints = fraccellpoints(intersected);
    mcells = find(intersected); area = area(intersected);

    % Generate grid using vertex coordinates (V) and indirection map (C)
    V = vertcat(fraccellpoints{:});
    C = cellfun(@(c) 1:size(c,1),fraccellpoints,'UniformOutput',false);
    for j = 2:size(C,1)
        addTo = C{j-1}(end);
        C{j} = C{j}+addTo;
    end
    Fgrid(i).grid = fractureplanegeneralgrid(V,C,points,...
        planenormal,aperture,tol);
    Fgrid(i).matrix_connection.cells = mcells;
    Fgrid(i).matrix_connection.area = area;
end

```

It then determines whether the fracture plane intersects a given matrix cell. If so, the area and vertex coordinates of the intersection are calculated. The function `pebiAABBintersect` is used in a double `for` loop to determine the intersection of every fracture plane and matrix cell combination.

The generated intersection data are then used as input for a function called `fractureplanegeneralgrid`, which generates a 3D grid for each fracture plane. Each 3D grid will have a thickness equal to the aperture of the corresponding fracture plane. The list of intersected matrix cells and the respective intersection areas

are also saved to facilitate the calculation of fracture–matrix NNCs. Figure 9.1b shows the grids constructed with the code from Listing 9.2. The red and yellow fractures are vertical but angled with respect to the horizontal axes, which results in grid cells that have uneven lengths along the fractures. The green fracture is inclined relative to the vertical axis and, as a result, has an unstructured grid pattern. The figure can be generated as follows:

```
colors = ['r','g','y'];
for i = 1:3
    plotGrid(Fgrid(i).grid,'FaceColor',colors(i));
end
axis equal tight; view(30,45);
xlim([0 physdim(1)]); ylim([0 physdim(2)]);
```

Figure 9.1c shows the intersected matrix grids and can be generated as follows:

```
for i = 1:3
    plotGrid(G,Fgrid(i).matrix_connection.cells,...
            'FaceAlpha', 0.5, 'FaceColor', colors(i));
end
```

The generated fracture grids are then saved in the previous grid object G under a new field `FracGrid`, as shown in Listing 9.3. The grid is also extended into a global grid by using the `assembleGlobalGrid` function to append the fracture grids in $G.FracGrid$ to the matrix grid. The grid cell indices for the fracture grids will continue from the matrix or the last appended grid. In the case of Figure 9.1, the global grid cell index will run from 1 to $(N_m + N_{red} + N_{green} + N_{yellow})$, with the first N_m indices representing matrix grids, the next N_{red} indices representing the red fracture cells, the next N_{green} indices representing the green fracture cells, and the last N_{yellow} indices representing the yellow fracture cells. An empty NNC list is also created in preparation for NNC calculations. Note that whenever possible, the resulting fracture grids should be inspected. Occasionally, small grid cells may be omitted by `EDFMgrid`. In such cases, a tighter floating point tolerance can be specified as an optional input.

Next, the empty NNC list needs to be populated to connect cells. We first consider neighboring cell connections. Because MRST's grid generator automatically determines neighboring connections, the matrix cells are already connected to each other; similarly, fracture cells are also connected to neighboring fracture cells that are within the same fracture. These connections need not be added to the NNC list.

For matrix–fracture connections, because intersections have already been determined in the fracture grid construction step, we already know which matrix cell

Listing 9.3 Assemble global grid.

```

% Initiate starting indices for cells, faces and nodes
cstart = G.cells.num+1;
fstart = G.faces.num+1;
nstart = G.nodes.num+1;

% Append Fgrid to G
for i = 1:length(fracplanes)
    fieldname = ['Frac',num2str(i)];

    % Add fracture grid to G.FracGrid
    G.FracGrid.(fieldname) = Fgrid(i).grid;

    % Save global starting indices and compute next one
    G.FracGrid.(fieldname).cells.start = cstart;
    G.FracGrid.(fieldname).faces.start = fstart;
    G.FracGrid.(fieldname).nodes.start = nstart;
    cstart = cstart + G.FracGrid.(fieldname).cells.num;
    fstart = fstart + G.FracGrid.(fieldname).faces.num;
    nstart = nstart + G.FracGrid.(fieldname).nodes.num;

    % Append poroperm data
    G.FracGrid.(fieldname).rock.perm = ...
        ones(G.FracGrid.(fieldname).cells.num,1)*fracplanes(i).perm;
    G.FracGrid.(fieldname).rock.poro = ...
        ones(G.FracGrid.(fieldname).cells.num,1)*fracplanes(i).poro;
end
G.nnc = []; % Instantiate an empty list of NNCs
G = assembleGlobalGrid(G); % Create a global grid

```

each fracture cell should be connected with. Figure 9.1 shows matrix cells that are connected to fracture cells; the matrix cells are color-coded to match the fractures they should be connected to. For each matrix–fracture cell pair, the cell-to-cell transmissibility, T^{mf} , is calculated using (9.7); the two cell indices and associated transmissibility T^{mf} are then appended to the NNC list. This procedure is achieved with the function:

```
G = fracturematrixNNC3D(G, 1e-5);
```

Listing 9.4 presents details from the underlying code. Here, `G.nnc.cells` is a matrix in which every row contains the indices of two connected cells: one matrix and one fracture. The corresponding row in `G.nnc.T` contains the transmissibility between the cells. The transmissibility is calculated using the `calfracmatCI` function, which uses the formula in (9.7). The average normal distance (9.8) between points in a matrix cell and a fracture cell is calculated using the `calcdavg` function.

Listing 9.4 *Generate non-neighboring connections between fracture and matrix.*

```

G.nnc.cells = []; G.nnc.T = [];
G.nnc.area = []; G.nnc.type = [];
tol = 1e-6;

for i=1:length(fracplanes)
    points = fracplanes(i).points;
    diffp = diff(points,1);
    planenormal = cross(diffp(1,:), diffp(2,:));
    planenormal = planenormal/norm(planenormal);
    fieldname = ['Frac',num2str(i)];

    % Generate list of NNC pairs
    fcellstart = G.FracGrid(fieldname).cells.start;
    Nf = Fgrid(i).grid.cells.num;
    mcells = Fgrid(i).matrix_connection.cells;
    N_nnc = length(mcells);
    nncpairs = [mcells, repmat((1:Nf)+fcellstart-1', N_nnc/Nf, 1)];

    % Calculate average distance and CI
    CI = ones(N_nnc,1); % instantiate list of CI
    for j=1:N_nnc
        [cn,cpos] = gridCellNodes(G,Fgrid(i).matrix_connection.cells(j));
        cellnodecoords = G.nodes.coords(cn,:);
        davg = calcdavg(cellnodecoords,planenormal,points(1,:),tol);
        CI(i) = calcfrcmatCI(cellnodecoords,...
            Fgrid(i).matrix_connection.area(j),...
            planenormal,points(1,:),davg,tol);
    end

    % Calculate cell to cell transmissibility
    pv = poreVolume(G,G.rock);
    w1 = pv(nncpairs(:,1))./G.rock.perm(nncpairs(:,1));
    w2 = pv(nncpairs(:,2))./G.rock.perm(nncpairs(:,2));
    wt = pv(nncpairs(:,1))+pv(nncpairs(:,2));
    Trans = 2*CI.*(wt./(w1+w2));

    % Append NNC data
    G.nnc.cells = [G.nnc.cells;nncpairs];
    G.nnc.T = [G.nnc.T;Trans];
    G.nnc.area = [G.nnc.area;Fgrid(i).matrix_connection.area];
end

```

Finally, connections between intersecting fractures need to be established:

```
[G,fracplanes] = fracturefractureNNCs3D(G, fracplanes, 1e-5);
```

In the underlying code, shown in Listing 9.5, we exploit the fact that if two fracture cells intersect, they must also intersect the same matrix cell. As such, we utilize the existing fracture–matrix NNC data to narrow down the fracture cell pairs that

have to be evaluated. For each fracture pair, the vertex coordinates of every fracture cell pair are provided to `PEBIPEBIintersection3D` to determine intersections. As was done for matrix–fracture NNCs, the indices of the intersecting cells along with the associated transmissibility T^{ff} are appended to the NNC list. Figure 9.1d shows an example in which the intersecting fracture cell pairs are highlighted with red and green and green and yellow colors, respectively. The figure was generated with the following code:

```
allnncells=nnc_pairs(:);
for i=1:3
    fieldname = ['Frac',num2str(i)];
    cstart    = G.FracGrid.(fieldname).cells.start;
    cend      = cstart-1+G.FracGrid.(fieldname).cells.num;

    plotGrid(G, cstart:cend, 'FaceColor', 'none');
    plotGrid(G, allnncells(ismember(allnncells',cstart:cend)),...
            'FaceColor', colors(i));
end
```

The next steps in this example follow the standard procedure for running a black-oil simulation using the `ad-blackoil` module. Whereas this example involves only two phases, a three-phase simulation model will be set up so that readers can easily make necessary modifications for their applications. The compressible three-phase fluid can be set up as follows:

```
% Define a three-phase fluid model without capillarity. Properties are
% listed in the order 'Water-Oil-Gas'.
pRef = 100*barsa;
fluid = initSimpleADIFluid('phases', 'WOG', ...
    'mu', [ 1, 5, 0.2] * centi*poise, ...
    'rho', [1000, 700, 250] * kilogram/meter^3, ...
    'c', [1e-8, 1e-5, 1e-3] / barsa, ...
    'n', [ 2, 2, 2], ...
    'pRef', pRef);
```

The fluid object and global grid can then be used to set up a three-phase black-oil model. In this model, the dissolved gas and vaporized oil options are turned off. The effects of gravity are also disregarded. Additionally, the operators within the model are overwritten to ensure that the NNCs are incorporated in the model:

```
gravity off
model = ThreePhaseBlackOilModel(G,G.rock,fluid,'disgas',false,'vapoil',false);
model.operators = setupEDFMOperatorsTPFA(G, G.rock, tol);
```

Listing 9.5 *Generate non-neighboring connections among fractures.*

```

tol=1e-6;
nnc_pairs = []; nnc_T = [];

% Indices of matrix cells which contain fracture cells
mcells = unique(G.nnc.cells(:,1));
N_nnc_max = length(mcells);

for i=1:N_nnc_max
    mcelli = mcells(i);
    ind = ismember(G.nnc.cells(:,1),mcelli);

    if sum(ind)==2
        fraccells = G.nnc.cells(ind,2)';
        fraccellareas = G.nnc.area(ind);
        cellnodes1 = G.nodes.coords(gridCellNodes(G,fraccells(1)),:);
        cellnodes2 = G.nodes.coords(gridCellNodes(G,fraccells(2)),:);

        % Intersection calculation
        [intersected,~,xlength,df] = ...
        PEBIPEBIintersection3D(cellnodes1,cellnodes2,tol);

        if intersected
            nnc_pairs = [nnc_pairs; sort(fraccells)];

            % Calculate transmissibility
            aperture1 = G.cells.volumes(fraccells(1))/fraccellareas(1);
            aperture2 = G.cells.volumes(fraccells(2))/fraccellareas(2);
            Trans1 = G.rock.perm(fraccells(1))*aperture1*xlength/df(1);
            Trans2 = G.rock.perm(fraccells(2))*aperture2*xlength/df(2);
            Trans = (Trans1+Trans2)/(Trans1+Trans2);
            nnc_T = [nnc_T; Trans];
        end
    end
end

G.nnc.cells = [G.nnc.cells;nnc_pairs];
G.nnc.T = [G.nnc.T;nnc_T];

```

Wells are specified with one well injecting one pore volume over two years:

```

%% ADD INJECTOR
totTime = 2*year;
tpv = sum(model.operators.pv);
wellRadius = 0.1;
[nx, ny, nz] = deal(G.cartDims(1), G.cartDims(2), G.cartDims(3));
cellinj = 1:nx*ny:(1+(nz-1)*nx*ny);
W = addWell([], G, G.rock, cellinj, 'Type', 'rate', ...
            'Val', tpv/totTime, 'Radius', wellRadius, ...
            'Comp_i', [1, 0, 0], 'Name', 'Injector');

```

and another well producing at a fixed pressure:

```

%% ADD PRODUCER
cellprod      = nx*ny : nx*ny : nz*nx*ny;
W             = addWell(W, G, G.rock, cellprod, 'Type', 'bhp', ...
                       'Val', 50*barsa, 'Radius', wellRadius, ...
                       'Comp_i', [1, 1, 0], 'Name', 'Producer');
    
```

The model is then initialized to be fully saturated with oil at constant pressure:

```

s0 = [0, 1, 0];
state = initResSol(G, pRef, s0);
    
```

Finally, a timestepping scheme is set up before the simulation is launched. The timestep is set to 30 days, with an initial ramp-up to the designated step length:

```

dt      = rampupTimesteps(totTime, 30*day, 10);
schedule = simpleSchedule(dt, 'W', W);

%% LAUNCH SIMULATION
[ws, states, report] = simulateScheduleAD(state, model, schedule);
    
```

The results of the simulation are shown in Figure 9.4, where the progression of the water saturation front is visibly affected by the three fractures. At 90 days, the front just reaches the fracture network. At 150 days, water has short-circuited through

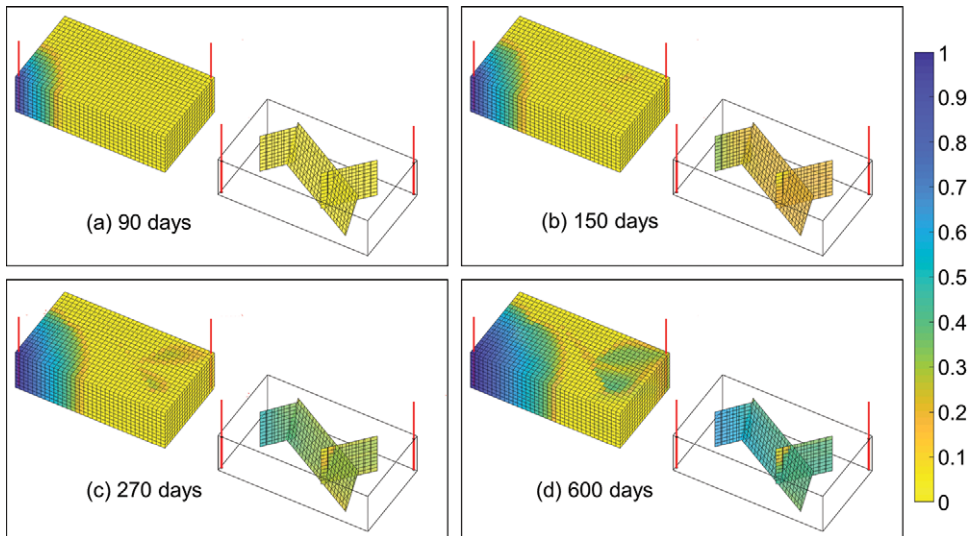


Figure 9.4 Results for a waterflood simulation into the model in Figure 9.1. Water saturation maps for the matrix and fractures are shown for four different timesteps.

the fracture network. Water breaks through at the well at 270 days. After a further 330 days, most of the domain remains unswept by water. The plots are produced with MRST's plotting tool: `plotToolBar(G, states)`.

9.6 Upscaling a Stochastically Generated Fracture Network

Instead of manually creating a fracture network like in the previous example, the statistical properties of a fracture network can be used to stochastically generate discrete fracture networks (DFN) realizations, which enable the study of flow behaviors within NFRs. Figure 9.5d is an example of a stochastically generated DFN. In this example, a DFN will be generated and processed using the `hfm` module. Then, the resulting EDFM will be used for numerical upscaling.

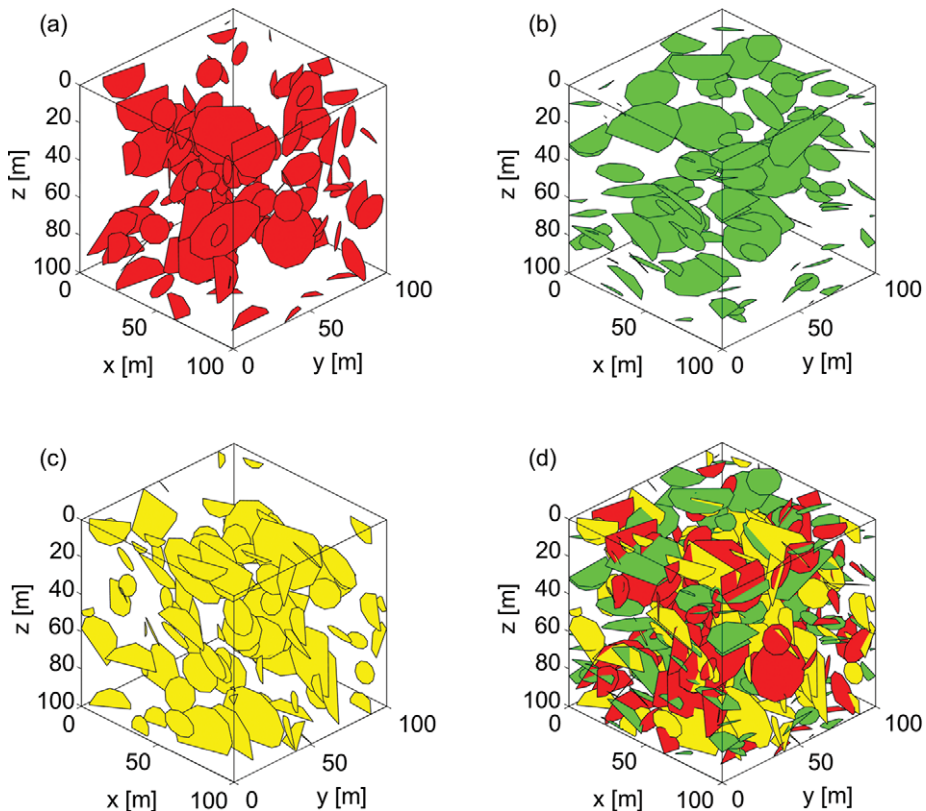


Figure 9.5 Stochastically generated DFN. (a) Vertical fracture set (red) oriented along the y -axis, (b) horizontal fracture set (green), (c) vertical fracture set (yellow) oriented along the x -axis, and (d) full fracture network. Note that this is not the fracture network used in Figure 9.6.

To generate a DFN, we first need random generators that output fracture properties; e.g., fracture size, orientation, aperture, and position. These random generators are derived from the statistical distributions of the properties in question. For example, the cumulative distribution function for fracture sizes that follow a power law distribution is

$$C(s) = \frac{s^{1-K_{PL}} - s_{min}^{1-K_{PL}}}{s_{max}^{1-K_{PL}} - s_{min}^{1-K_{PL}}}, \tag{9.13}$$

where s refers to fracture size and K_{PL} is a power-law exponent. The distribution function can be inverted to express s in terms of $C(s)$. Noting that $0 \leq C(s) \leq 1$, we replace $C(s)$ with a random number generator R_U that returns values between 0 and 1, drawn from a standard uniform distribution. The resulting random fracture size generator R_{PL} (subscript referring to power law) is

$$R_{PL} = \left(s_{min}^{1-K_{PL}} + R_U \left(s_{max}^{1-K_{PL}} - s_{min}^{1-K_{PL}} \right) \right)^{\frac{1}{1-K_{PL}}}. \tag{9.14}$$

The same approach can be used to derive random generators for other fracture properties. In this example, the steps for creating a DFN are based on the approach by Priest [33]:

1. Start with the first fracture orientation set.
2. Randomly generate the parameters for a new fracture: orientation, aperture, size, location.
3. Check that the new fracture does not intersect an exclusion zone around existing fractures within the same set.
 - Remove new fracture if intersection is detected.
 - Update fracture density if no intersection is detected.
4. If target fracture density is not met, repeat from Step 2.
5. Repeat Steps 2–4 for all other fracture orientation sets.

Step 3 in this procedure is a modification based on the observation of stress shadow zones around preexisting fractures. The shapes of exclusion zones, for practical reasons, are often chosen to be simple geometries that enclose fractures [20, 36, 46].

Note that the DFN generation procedure outlined can be modified to account for more features. For example, we have assumed that aperture and size are not correlated. The algorithm can also be modified to specify aperture as a function of fracture size. Another possible modification is the inclusion of fracture abutment probabilities, which can be used to create a random decision maker (equivalent to an unfair coin toss); new fractures may then either intersect with or truncate at an older fracture based on the outcome of the decision maker [17].

This procedure has been implemented in MATLAB and is part of the `hfm` module. The DFN generator takes as inputs the domain in which fractures will be generated, fracture network properties, and a parameter that controls the size of exclusion zones. In the following code, three orthogonal fracture sets are defined. The fracture orientations are allowed to vary around corresponding mean orientations based on the Fisher distribution. The fracture sizes are also allowed to vary according to a power-law distribution. Fracture apertures are constant. Fractures are represented as 10-sided polygons randomly distributed in space based on the Poisson process:

```

%% FRACTURE NETWORK STATISTICAL PARAMETERS
physdim=[100 100 100]; % domain size

% Fracture sets 1 (red), 2(green), 3(yellow)
fracinput1 = struct('circle',true,'vertices',10,'poro',1,'perm',1e4*darcy,...
                  'P32',0.025*(1/meter),'aperture',1e-3*meter,...
                  'normal',struct('direction',[1 0 0],'K',10),...
                  'size',struct('minsize',5*meter,...
                               'maxsize',20*meter,'exponent',1.5));
fracinput2 = fracinput1; fracinput2.normal.direction=[0 0 1];
fracinput3 = fracinput1; fracinput3.normal.direction=[0 1 0];

% Exclusion zone is cylindrical with radius being (1+exclzonemult) times
% the fracture radius; height is exclzonemult times the fracture radius.
exclzonemult = 0.01;

```

The parameters are passed to the DFN generator in the `hfm` module. Periodicity is enforced at the domain boundaries, such that truncated fractures reappear at opposite boundaries:

```

%% DFN GENERATION
tol = 10^-5; % tolerance for comparison of doubles
fracplanes = DFNgenerator([],fracinput1,physdim,exclzonemult,tol);
fracplanes = DFNgenerator(fracplanes,fracinput2,physdim,exclzonemult,tol);
fracplanes = DFNgenerator(fracplanes,fracinput3,physdim,exclzonemult,tol);

```

Note that the generated DFN shown in Figure 9.5 is simply one of a limitless number of possible realizations that honor the fracture network parameters specified. In actuality, a proper risk assessment should adopt a Monte Carlo approach by generating hundreds to thousands of DFN realizations in order to quantify the uncertainty in flow behavior resulting from our limited knowledge of the fracture network. Figure 9.5 can be produced using the following code:

```

%% PLOT DFN
colourchoice=['r','g','y'];
figure;
hold on;
plotGrid(cartGrid([1 1 1],physdim),'facealpha',0);
axis equal tight
view(45,30);
for i=1:3
    index = find(vertcat(fracplanes.SetID)==i);
    C = colourchoice(i);
    for j=index'
        X = fracplanes(j).points(:,1);
        Y = fracplanes(j).points(:,2);
        Z = fracplanes(j).points(:,3);
        fill3(X,Y,Z,C);
    end
    xlabel('x','Interpreter','latex')
    ylabel('y','Interpreter','latex')
    zlabel('z','Interpreter','latex')
end
end

```

The output from the DFN generator is a database of fractures that can directly be used as an input to the `hfm` module. In the next part of this example, we show how the generated DFN can be numerically upscaled using the method introduced by Durlofsky [10]. Numerical upscaling involves running a single-phase incompressible simulation in which the simulation domain is subjected to a pressure differential. The calculated flow rate can then be used to back-calculate an equivalent permeability using Darcy's law. For an in-depth discussion of upscaling, we refer the reader to chapter 15 in the MRST textbook [28]. In this example, the EDFM grid and NNCs are set up automatically using the functions `EDFMgrid`, `fracturematrixNNC3D`, and `fracturefractureNNCs3D`. These functions perform the same EDFM preprocessing tasks as those shown in the previous example. Take note that due to the number of fractures involved, the EDFM preprocessing phase will take a while. For the reader's convenience, an already preprocessed stochastically generated DFN is provided (`SampleDFN_preprocessed.mat`) so that the reader may proceed straight to the numerical upscaling part of this example:

```

%% SETUP GRID
celldim = [25 25 25];
G       = cartGrid(celldim, physdim);
G       = computeGeometry(G);
km      = 10*milli*darcy;
G.rock = makeRock(G, km, 0.25);

```

```

%% EDFM PRE-PROCESSING
tol = 1e-6;
[G,fracplanes] = EDFMgrid(G,fracplanes,'Tolerance',tol);
G = fracturematrixNNC3D(G,tol);
[G,fracplanes] = fracturefractureNNCs3D(G,fracplanes,tol);

```

Because numerical upscaling only requires a single flowing phase, a fluid model is set up with properties of water. Then, the EDFM grid and fluid model are used to instantiate a water model. Additionally, because the flowing fluid is incompressible, the `stepFunctionIsLinear` property in the water model is set to `true`:

```

pRef = 100*barsa;
fluid = initSimpleADIFluid('phases','W','mu',1*centi*poise,'pRef',pRef,...
                           'rho',1000*kilogram/meter^3,'c',0/barsa);

%% SETUP WATER MODEL
gravity reset off
model = WaterModel(G,G.rock,fluid);
model.operators = setupEDFMOperatorsTPFA(G,G.rock,tol);
model.stepFunctionIsLinear = true;

```

In the next step, initial and boundary conditions are set. The initial condition is such that the entire domain is at constant pressure. A pressure differential is applied to the western and eastern boundaries of the domain. The fracture cell faces at these boundaries are determined using the `findfracboundaryfaces` function that checks for intersections between domain boundaries and fractures:

```

% Initial condition
state = initResSol(G,pRef);

% Find fracture cell faces at domain boundary
boundfaces = findfracboundaryfaces(G,tol);
% Set pressure differential on opposing boundaries in the x-direction
deltaP = 50*barsa;
bc = pside([],G.Matrix,'East',pRef);
matwestfaces = bc.face;
bc = pside(bc,G.Matrix,'West',pRef + deltaP);
bc = addBC(bc,boundfaces.East,'pressure',pRef);
bc = addBC(bc,boundfaces.West,'pressure',pRef + deltaP);
bc.sat = ones(size(bc.face));

```

A simple schedule with a single 1 second timestep is set up and the simulation is launched:

```

schedule = simpleSchedule(1,'bc',bc);
[~,states,~] = simulateScheduleAD(state,model,schedule);

```

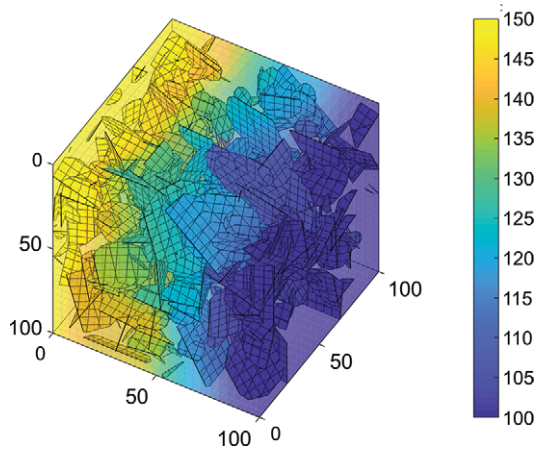



Figure 9.6 Pressure field under a pressure differential imposed in the x -direction from the West to the East. Pressure is shown in bars.

Figure 9.6 shows the pressure field established in the model given the applied pressure differential. Pressures are high on the western boundary and low on the eastern boundary. The isopressure surfaces are uneven due to the influence of highly conductive fractures within the simulation domain. This figure can be reproduced with the code:

```
plotCellData(G,states{1}.pressure,1:G.Matrix.cells.num,...
    'FaceAlpha',0.5,'EdgeAlpha',0.1);
plotCellData(G,states{1}.pressure,G.Matrix.cells.num+1:G.cells.num);
view(30,45);
caxis([100 150]*barsa); colorbar('EastOutside');
axis equal tight; box on
```

The simulation results can be easily processed to back-calculate an equivalent permeability for the whole domain. To do this, the overall flux through the domain in the x -direction has to be determined. Because the fluid involved is incompressible, fluid flux is equal at both the western and eastern boundaries. We will calculate the flux at the western boundary. Once the flux is calculated, Darcy's law can be used to determine the equivalent permeability:

```
% Determine flux through western boundary
westfaces = [matwestfaces;boundfaces.West'];
waterflux = sum(abs(states{end}.flux(westfaces,1)));

% Inverse Darcy's law for permeability
k_eq      = waterflux*fluid.muW(1)*physdim(1)/(physdim(1)*physdim(2)*deltaP);
```

In this example, only the x -direction of the equivalent permeability has been calculated. The boundary conditions can easily be modified such that the pressure differential is prescribed in other directions. By doing so, the y - and z -directions of the equivalent permeability can be determined. For the DFN in Figure 9.6, the equivalent permeability in the x -direction is 33.0 md. In comparison, the matrix permeability is 10.0 md. Due to the high conductivity of the fractures, the flow capacity of the overall volume in the x -direction has been increased threefold. Note that in the method proposed by Durlofsky [10], periodic boundary conditions are applied to boundaries perpendicular to the applied pressure differential. However, because the fracture network that is being upscaled in this example is largely isotropic, crossflow is not expected to be significant. Hence, we assume that no crossflow occurs and impose no-flow conditions on side boundaries. The example can also be repeated to obtain a distribution of possible equivalent permeabilities, given the same fracture network parameters.

9.7 Simulation of Well Test Response in an Outcrop-Based Fracture Network

In the final example of this chapter, we show how a well test in a real fracture network can be simulated. We use a publicly available database provided by Bisdom et al. [5] that contains fracture networks mapped on outcrops of the Jandaira carbonate formation in the Potiguar basin, Brazil. The data are contained in two formats: kml and shapefiles. The former format can be read into Google Earth to visualize the fracture network. Shapefiles, on the other hand, can easily be read into MATLAB using the `shaperead` function, which is part of the Mapping Toolbox. For the reader's convenience, we have parsed the shapefile for the Apodi 2 fracture network and saved the data as `Apodi2.mat` (Figure 9.7). This file is available in the `hfm` module and contains two variables: `physdim` and `fracplanes`. The former contains information about the size of the domain of interest. The latter is a `struct` array that contains fracture segments.

Once `Apodi2.mat` is loaded, similar to the previous examples, an orthogonal grid can be set up. The grid and `fracplanes` can then be passed through the `EDFMgrid`, `fracturematrixNNC3D`, and `fracturefractureNNC3D` functions to create an EDFM. Note that in this example, the EDFM only has one grid cell in the z -direction and is actually a 2D model. However, the example still utilizes the 3D version of the `hfm` module so that readers can easily modify the example such that there are more layers in the z -direction. Such a modification will allow for the study of gravitational effects on flow. Due to the number of fractures involved, the EDFM preprocessing step will take a while to complete. For readers who are keen to go straight into the flow simulation, we have provided `Apodi2preprocessed.mat`, which contains the preprocessed grid `G`:

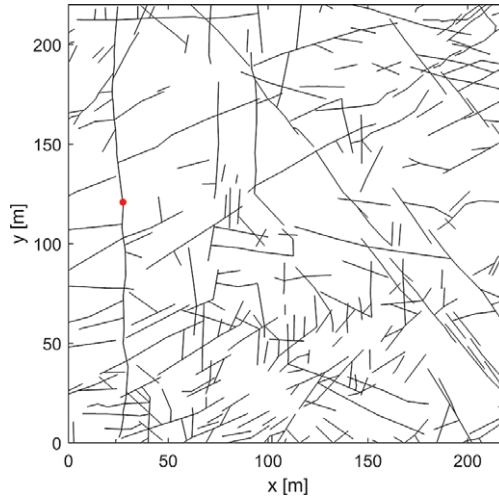


Figure 9.7 The Apodi 2 fracture network from Bisdom et al. [5] mapped on outcrops of the Jandaira carbonate formation in the Potiguar basin, Brazil. The red point is the location of a well.

```

%% SETUP GRID
G      = cartGrid([220 220 1], physdim);
G      = computeGeometry(G);
G.rock = makeRock(G,1*milli*darcy,0.25);
    
```

```

%% EDFM PRE-PROCESSING
tol      = 1e-6;
[G,fracplanes] = EDFMgrid(G,fracplanes,'Tolerance',tol);
G        = fracturematrixNNC3D(G,tol);
[G,fracplanes] = fracturefractureNNCs3D(G,fracplanes,tol);
    
```

For this well test, we assume that only a single-phase fluid is flowing. As such, we set up a fluid object with typical oil properties. The grid and fluid are used to instantiate a `WaterModel`. Note that although the model used here is named after water, it can in fact be used with any fluid as long as only a single phase is involved:

```

%% SETUP FLUID MODEL WITH OIL PROPERTIES
pRef = 100*barsa;
fluid = initSimpleADIFluid('phases','W','mu',8*centi*poise,'pRef',pRef,...
    'rho',700* kilogram/meter^{3},'c',1e-5/barsa);

%% SETUP WATER MODEL
gravity reset off
model      = WaterModel(G, G.rock, fluid);
model.operators = setupEDFMOperatorsTPFA(G, G.rock, tol);
    
```

In the next step, we specify a point sink that acts as a well that is being produced at constant rate. The point sink will be positioned within a fracture cell to emulate the process of producing from a fracture. To do this, we first plot the EDFM grid using `plotEDFMgrid`. Using the plot as a reference, the approximate coordinates of a point on a fracture are chosen and used to determine the index of a fracture cell in that location. The fracture cell index can then be used as an input to the `addSource` function to specify a point sink. This procedure is implemented in the following code, which will produce Figure 9.7:

```

%% PLOT GRID TO HELP DETERMINE APPROXIMATE WELL LOCATION
[hm,hf]=plotEDFMgrid(G); hm.EdgeAlpha=0; hold on
scatter(27.16, 121, 10, 'r', 'filled'); view(0,90); box on

%% LOCATE NEAREST FRACTURE CELL
fraccellcent = [27.16,121,0.5]; % Approximate location of well
fraccell      = find(all(abs(G.cells.centroids-fraccellcent)<0.5,2));
fraccell      = fraccell(fraccell>G.Matrix.cells.num);

%% SET UP SINK
src           = addSource([],fraccell,-1*(meter^3)/day);
src.sat       = ones(size(src.cell));

```

The model is then initialized and a schedule with log spaced timesteps is set up. The log spaced timestepping is required to capture early time flow dynamics that are mainly concentrated near the well. Finally, the simulation can be launched:

```

state         = initResSol(G, pRef);
dt            = diff(10.^(-8.5+(1:60)*0.25)); % log spaced timesteps
schedule      = simpleSchedule(dt,'src',src);
[~, states] = simulateScheduleAD(state, model, schedule);

```

The results of well tests are often visualized on log–log plots of pressure derivatives with respect to log time [9]. The next code excerpt shows how the simulation results can be postprocessed to generate such a plot (Figure 9.8):

```

pvalues = cellfun(@(x) x.pressure(src.cell), states);
tottime = cumsum(dt);
dp_dlnt = -diff(pvalues)./diff(log(tottime));
midtime = 0.5*(tottime(1:end-1)+tottime(2:end));
loglog(midtime,dp_dlnt);

```

The diagnostic plot (Figure 9.8) shows that there are two flow regimes separated by a brief drop in the pressure derivative. The first flow regime corresponds to fracture-dominated flow. In this phase, fluid flow occurs only in the fracture network. The second flow regime corresponds to matrix-dominated flow. In this

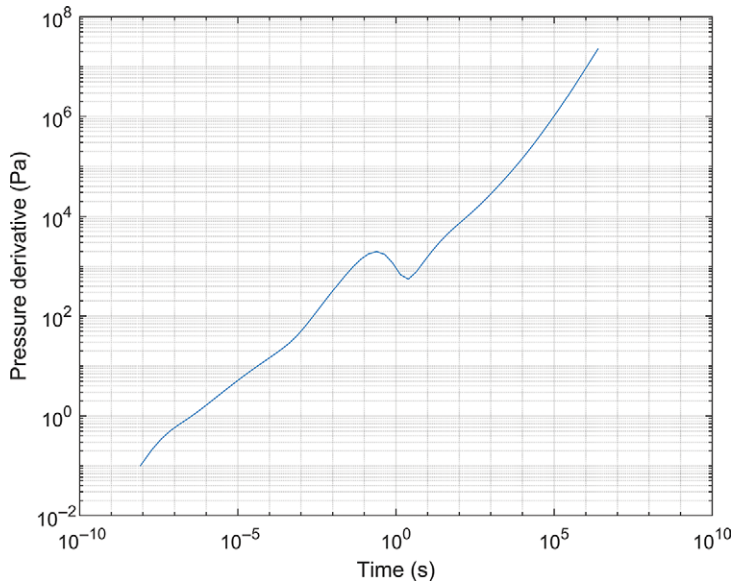


Figure 9.8 Diagnostic plot for a well test simulated using EDFM in a real fracture network.

flow regime, fractures serve as the conduit for fluid flow from the matrix to the well. At the transition between the two flow regimes, the pressure front in the fracture network reaches the domain boundaries while the pressure in the matrix remains undisturbed (Figure 9.9a). At late time (beyond the transition), pressure in the matrix will deplete as fluid is produced from the well through the fractures (Figure 9.9b).

9.8 Concluding Remarks

In this chapter, EDFM has been presented as a way to perform high-resolution flow simulations in a fractured porous medium. In contrast to the popular dual-porosity model, EDFM represents fractures explicitly in the simulation model as line sinks or sources. Due to the explicit fracture representation, EDFM is capable of capturing the geometrical complexity of fracture networks. The `hfm` module provides the basic building blocks of EDFM. Various research efforts are still ongoing to improve the capabilities of EDFM. For example, Tene et al. [44] proposed the projection-based EDFM, which addresses the issue of pressure and saturation continuity across fractures in EDFM; their work allows sealing fractures to be modeled using EDFM. This method is explained in more detail and used to model

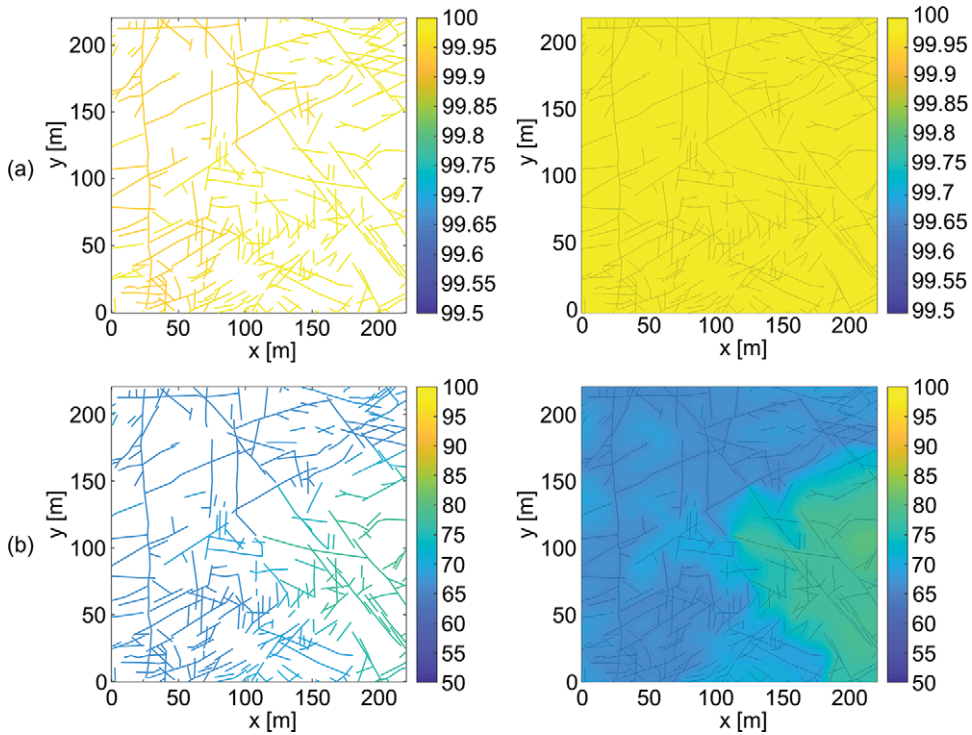


Figure 9.9 Pressure distribution for the Apodi 2 well test simulation. Row (a) contains the pressure fields at 1 second. Row (b) contains the pressure fields at 3.66 days. Column 1 shows the pressures in the fracture cells. Column 2 shows the pressures in the matrix cells. Pressures are in bars.

flow in unconventional oil and gas reservoirs in Chapter 10. Hui et al. [18] have further simplified and optimized the EDFM preprocessing algorithm, thus allowing users to handle a much larger number of fractures. Additionally, various authors have coupled EDFM with geomechanical simulations for fully coupled simulations [35, 38, 50]. We highly recommend that keen readers delve into these works to continue exploring EDFM beyond what has been covered in this chapter. Lastly, we encourage readers to modify and run the examples presented in this chapter to discover for themselves the ease of running EDFM simulations.

Acknowledgement. To close this chapter, we express our utmost gratitude to Knut-Andreas Lie and Olav Møyner for inviting us to contribute to this book. We also thank Energi Simulation for funding Sebastian Geiger's Chair in Carbonates Research. Our thanks also go to Arjan Kamp, François Gouth, and Eddy Francot at Total, who supported Daniel Wong during his doctoral studies.

References

- [1] A. S. Abushaikha and O. R. Gosselin. Matrix-fracture transfer function in dual-media flow simulation: review, comparison and validation. In *Europec/EAGE Conference and Exhibition*, 2013. doi: 10.2118/113890-MS.
- [2] G. Barenblatt, I. Zheltov, and I. Kochina. Basic concepts in the theory of seepage of homogeneous liquids in fissured rocks [strata]. *Journal of Applied Mathematics and Mechanics*, 24(5):1286–1303, 1960. doi: 10.1016/0021-8928(60)90107-6.
- [3] B. Berkowitz. Characterizing flow and transport in fractured geological media: a review. *Advances in Water Resources*, 25(8–12):861–884, 2002. doi: 10.1016/S0309-1708(02)00042-8.
- [4] I. Berre, F. Doster, and E. Keilegavlen. Flow in fractured porous media: a review of conceptual models and discretization approaches. *Transport in Porous Media*, 130:215–236, 2019. doi: 10.1007/s11242-018-1171-6.
- [5] K. Bisdorn, G. Bertotti, H. Bezerra, M. Van Eijk, E. Van Der Voet, and J. Reijmer. Deterministic fracture network models from the Potiguar basin, Brazil, 2017. doi: 10.4121/uuid:988152da-3ac3-44cb-9d87-c7365e3707b6.
- [6] K. Bisdorn, G. Bertotti, and H. Nick. The impact of different aperture distribution models and critical stress criteria on equivalent permeability in fractured rocks. *Journal of Geophysical Research: Solid Earth*, 121(5), 4045–4063, 2016. doi: 10.1002/2015JB012657.
- [7] 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. doi: 10.1029/1999RG000074.
- [8] F. E. Botros, A. E. Hassan, D. M. Reeves, and G. Pohll. On mapping fracture networks onto continuum. *Water Resources Research*, 44(8):1–17, 2008. doi: 10.1029/2007WR006092.
- [9] D. Bourdet, J. Ayoub, and Y. Pirard. Use of pressure derivative in well test interpretation. *SPE Formation Evaluation*, 4(02):293–302, 1989. doi: 10.2118/12777-PA.
- [10] L. J. Durlofsky. Numerical calculation of equivalent grid block permeability tensors for heterogeneous porous media. *Water Resources Research*, 27(5):699–708, 1991. doi: 10.1029/91WR00107.
- [11] D. Egya, S. Geiger, and P. W. M. Corbett. Pressure-transient responses of fractures with variable conductivity and asymmetric well location. *SPE Reservoir Evaluation & Engineering*, 22(2):745–755, 2019. doi: 10.2118/190884-pa.
- [12] D. O. Egya, S. Geiger, P. W. M. Corbett, R. March, K. Bisdorn, G. Bertotti, and F. H. Bezerra. Analysing the limitations of the dual-porosity response during well tests in naturally fractured reservoirs. *Petroleum Geoscience*, 25(1):30–49, 2019. doi: 10.1144/petgeo2017-053.
- [13] M. Elfeel, S. Jamal, C. Enemanna, D. Arnold, and S. Geiger. Effect of DFN upscaling on history matching and prediction of naturally fractured reservoirs. In *75th European Association of Geoscientists and Engineers Conference and Exhibition 2013 Incorporating SPE EUROPEC 2013: Changing Frontiers*, 2013, pp. 10–13. Society of Petroleum Engineers, 2013. doi: 10.2118/164838-MS.
- [14] A. Firoozabadi. Recovery mechanisms in fractured reservoirs and field performance. *Journal of Canadian Petroleum Technology*, 39(11):13–17, 2000. doi: 10.2118/00-11-DAS.
- [15] B. Flemisch, I. Berre, W. Boon, A. Fumagalli, N. Schwenck, A. Scotti, I. Stefansson, and A. Tatomir. Benchmarks for single-phase flow in fractured porous media.

- Advances in Water Resources*, 111:239–258, 2017. doi: 10.1016/j.advwatres.2017.10.036.
- [16] S. Geiger and S. Matthäi. What can we learn from high-resolution numerical simulations of single- and multi-phase fluid flow in fractured outcrop analogues? In *Advances in the Study of Fractured Reservoirs*, pp. 125–144, Geological Society, London, 2014. doi: 10.1144/SP374.8.
- [17] N. J. Hardebol, C. Maier, H. Nick, S. Geiger, G. Bertotti, and H. Boro. Multiscale fracture network characterization and impact on flow: a case study on the Latemar carbonate platform. *Journal of Geophysical Research: Solid Earth*, 120(12):8197–8222, 2015. doi: 10.1002/2015JB011879.
- [18] M.-H. Hui, G. Dufour, S. Vitel, P. Muron, R. Tavakoli, M. Rousset, A. Rey, and B. Mallison. A robust embedded discrete fracture modeling workflow for simulating complex processes in field-scale fractured reservoirs. In *SPE Reservoir Simulation Conference, 10–11 April, Galveston, Texas, USA, 2019*. doi: 10.2118/193827-MS.
- [19] A. Jack and S. Sun. Controls on recovery factor in fractured reservoirs: lessons learned from 100 fractured fields. *Proceedings of SPE Annual Technical Conference and Exhibition*, 2003. doi: 10.2523/84590-MS.
- [20] J.-Y. Josnin, H. Jourde, P. Fénart, and P. Bidaux. A three-dimensional model to simulate joint networks in layered rocks. *Canadian Journal of Earth Sciences*, 39(10):1443–1455, 2002. doi: 10.1139/e02-043.
- [21] M. Karimi-Fard, L. Durlofsky, and K. Aziz. An efficient discrete fracture model applicable for general purpose reservoir simulators. *SPE Journal*, 9(2):227–236, 2004. doi: 10.2118/79699-MS.
- [22] D. C. Karvounis and P. Jenny. Adaptive hierarchical fracture model for enhanced geothermal systems. *Multiscale Modeling & Simulation*, 14(1):207–231, 2016. doi: 10.1137/140983987.
- [23] A. Kirkby, G. Heinson, and L. Krieger. Relating permeability and electrical resistivity in fractures using random resistor network models. *Journal of Geophysical Research*, 121(3):1546–1564, 2016. doi: 10.1002/2015JB012541.
- [24] M. Köppel, V. Martin, J. Jaffré, and J. E. Roberts. A Lagrange multiplier method for a discrete fracture model for flow in porous media. *Computational Geosciences*, 23(2):239–253, 2019. doi: 10.1007/s10596-018-9779-8.
- [25] S. H. Lee, M. F. Lough, and C. L. Jensen. Hierarchical modeling of flow in naturally fractured formations with multiple length scales. *Water Resources Research*, 37(3):443–455, 2001. doi: 10.1029/2000WR900340.
- [26] P. Lemonnier and B. Bourbiaux. Simulation of naturally fractured reservoirs. State of the art. *Oil & Gas Science and Technology – Revue de l’Institut Français du Pétrole*, 65(2):263–286, 2010. doi: 10.2516/ogst/2009067.
- [27] L. Li and S. H. Lee. Efficient field-scale simulation of black oil in a naturally fractured reservoir through discrete fracture networks and homogenized media. *SPE Reservoir Evaluation & Engineering*, 11(4):750–758, 2008. doi: 10.2118/103901-PA.
- [28] K.-A. Lie. *An Introduction to Reservoir Simulation Using MATLAB/GNU Octave: User Guide for the MATLAB Reservoir Simulation Toolbox (MRST)*. Cambridge University Press, Cambridge, UK, 2019. doi: 10.1017/9781108591416.
- [29] S. K. Matthai and M. Belayneh. Fluid flow partitioning between fractures and a permeable rock matrix. *Geophysical Research Letters*, 31(7):1–5, 2004. doi: 10.1029/2003GL019027.
- [30] A. Moïnfar, A. Varavei, K. Sepehrnoori, and R. T. Johns. Development of a novel and computationally-efficient discrete-fracture model to study IOR processes in naturally

- fractured reservoirs. In *SPE Improved Oil Recovery Symposium*, pp. 1–17. Society of Petroleum Engineers, 2012. doi: 10.2118/154246-MS.
- [31] R. A. Nelson. *Geologic Analysis of Naturally Fractured Reservoirs*. Gulf Professional Publishing, Boston, 2001.
- [32] P. Panfili and A. Cominelli. Simulation of miscible gas injection in a fractured carbonate reservoir using an embedded discrete fracture model. In *Abu Dhabi International Petroleum Exhibition and Conference*, 10–13 November, Abu Dhabi, UAE, 2014. doi: 10.2118/171830-MS.
- [33] S. D. Priest. *Discontinuity Analysis for Rock Engineering*. Springer, Dordrecht, the Netherlands, 1993. doi: 10.1007/978-94-011-1498-1.
- [34] V. Reichenberger, H. Jakobs, P. Bastian, and R. Helmig. A mixed-dimensional finite volume method for two-phase flow in fractured porous media. *Advances in Water Resources*, 29(7):1020–1036, 2006. doi: 10.1016/j.advwatres.2005.09.001.
- [35] G. Ren, J. Jiang, and R. M. Younis. A fully coupled XFEM-EDFM model for multiphase flow and geomechanics in fractured tight gas reservoirs. *Procedia Computer Science*, 80:1404–1415, 2016. doi: 10.1016/j.procs.2016.05.449.
- [36] C. E. Renshaw and D. D. Pollard. Numerical simulation of fracture set formation: a fracture mechanics model consistent with experimental observations. *Journal of Geophysical Research: Solid Earth*, 99(B5):9359–9372, 1994. doi: 10.1029/94JB00139.
- [37] P. N. Saevik, M. Jakobsen, M. Lien, and I. Berre. Anisotropic effective conductivity in fractured rocks by explicit effective medium methods. *Geophysical Prospecting*, 62(6):1297–1314, 2014. doi: 10.1111/1365-2478.12173.
- [38] A. Sangnimmuan, J. Li, and K. Wu. Development of efficiently coupled fluid-flow/geomechanics model to predict stress evolution in unconventional reservoirs with complex-fracture geometry. *SPE Journal*, 23, 2018. doi: 10.2118/189452-PA.
- [39] P. Schädle, P. Zulian, D. Vogler, S. R. Bhopalam, M. G. Nestola, A. Ebigbo, R. Krause, and M. O. Saar. 3D non-conforming mesh model for flow in fractured porous media using Lagrange multipliers. *Computers & Geosciences*, 132:42–55, 2019. doi: 10.1016/j.cageo.2019.06.014.
- [40] N. Schwenck, B. Flemisch, R. Helmig, and B. I. Wohlmuth. Dimensionally reduced flow models in fractured porous media: crossings and boundaries. *Computational Geosciences*, 19(6):1219–1230, 2015. doi: 10.1007/s10596-015-9536-1.
- [41] S. Shah, O. Møyner, M. Tene, K.-A. Lie, and H. Hajibeygi. The multiscale restriction smoothed basis method for fractured porous media (F-MsRSB). *Journal of Computational Physics*, 318:1–22, 2016. doi: 10.1016/j.jcp.2016.05.001.
- [42] M. Shakiba, J. S. d. A. Cavalcante Filho, and K. Sepehrnoori. Using embedded discrete fracture model (EDFM) in numerical simulation of complex hydraulic fracture networks calibrated by microseismic monitoring data. *Journal of Natural Gas Science and Engineering*, 55:495–507, 2018. doi: 10.1016/j.jngse.2018.04.019.
- [43] N. Siripatrchai, T. Ertekin, R. Johns, et al. Compositional simulation of discrete fractures incorporating the effect of capillary pressure on phase behavior. In *SPE Improved Oil Recovery Conference, 11–13 April, Tulsa, Oklahoma, USA*. Society of Petroleum Engineers, 2016. doi: 10.2118/179660-MS.
- [44] M. Tene, S. B. Bosma, M. S. Al Kobaisi, and H. Hajibeygi. Projection-based embedded discrete fracture model (pEDFM). *Advances in Water Resources*, 105: 205–216, 2017. doi: 10.1016/j.advwatres.2017.05.009.
- [45] H. Vo, J. Kamath, and R. Hui. High fidelity simulation of recovery mechanisms in complex natural fracture systems. In *SPE Reservoir Simulation Conference*, 2019. doi: 10.2118/193864-MS.

- [46] X. Wang. Fluid flow in multi-scale fractured networks: from field observation to numerical modelling. PhD thesis, Imperial College London, 2016.
- [47] J. Warren and P. Root. The behavior of naturally fractured reservoirs. *Society of Petroleum Engineers Journal*, 3(3):245–255, 1963. doi: 10.2118/426-PA.
- [48] P. A. Witherspoon, J. S. Y. Wang, K. Iwai, and J. E. Gale. Validity of cubic law for fluid flow in a deformable rock fracture. *Water Resources Research*, 16(6): 1016–1024, 1980. doi: 10.1029/WR016i006p01016.
- [49] Y. Xu and K. Sepehrnoori. Development of an embedded discrete fracture model for field-scale reservoir simulation with complex corner-point grids. *SPE Journal*, 24(4):1552–1575, 2019. doi: 10.2118/195572-PA.
- [50] Q.-D. Zeng, J. Yao, and J. Shao. Study of hydraulic fracturing in an anisotropic poroelastic medium via a hybrid EDFM-XFEM approach. *Computers and Geotechnics*, 105:51–68, 2019. doi: 10.1016/j.compgeo.2018.09.010.
- [51] Y. Zhang, W. Yu, K. Sepehrnoori, and Y. Di. A comprehensive numerical model for simulating fluid transport in nanopores. *Scientific Reports*, 1–11, 2016. doi: 10.1038/srep40507.
- [52] R. Zimmerman and G. Bodvarsson. Effective transmissivity of two-dimensional fracture networks. *International Journal of Rock Mechanics and Mining*, 33(4): 433–438, 1996. doi: 10.1016/0148-9062(95)00067-4.