MOX–Aprile 2014

FVCode3D: a user's guide

RESPONSABILI DI PROGETTO

Luca Formaggia, Anna Scotti

AUTORE:
Stefano Zonca

MOX, Dipartimento di Matematica "F. Brioschi"
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox@mate.polimi.it http://mox.polimi.it

Questo documento si riferisce al contratto di ricerca nr. 2500008934 del 20/09/2013 prot. 05/13 Eni, stipulato tra il Politecnico di Milano (Dipartimento di Matematica F. Brioschi) ed Eni. S.p.A. (Divisione Exploration and Production), Odl 4310067601. Titolo work order: Gridding for discrete fracture. Task b: sviluppo solutore steady state/pseudo steady state

Contents

1 Introduction

This document describes the functionality of the solver to simulate single-phase flow in fractured porous media by means of the Darcy equation in a 3D framework.

A Finite Volume (FV) method is employed to solve both the steady and pseudo-steady problem. The implemented code is written in `C++`. The code is able to operate on polyhedral grids, with the only limitation that the polyhedra should be convex. The fractures are discretized as a subset of the faces of the polyhedral mesh.

After a brief introduction to the underlining mathematical model, we present the numerical method that has been adopted, which is based on a two-point flux approximation (TPFA) scheme. We give some details on the treatment of the boundary conditions and the formulas used to compute the transmissibility. Moreover, we discuss the representation of the fractures both from a geometrical and from a modeling point of view. Some examples of numerical simulations are shown. Finally, we illustrate the structure of the implemented `C++` code, including practical guidelines for its usage and on the generation of the reference manual.

2 The mathematical model

We consider the Darcy's law coupled with the mass conservation equation:

$$\begin{cases} \mathbf{u} = -\dfrac{1}{\mu} K \nabla p \, , in \; \Omega \\ \\ \nabla \cdot \mathbf{u} = f \, , in \; \Omega \end{cases} \tag{1}$$

where $\mathbf{u}$ is the flow velocity, p the pressure, K is the symmetric positive definite permeability tensor, μ the dynamic viscosity and f is the volumetric source/sink term.
We have assumed that the fluid and the medium are incompressible and we also have neglected the gravitational effects.

Following a common practice, we combine the two equations into one and we get a second order equation for just the pressure:

$$-\nabla \cdot \left(\frac{1}{\mu} K \nabla p \right) = f \tag{2}$$

The model we have adopted to describe a single-phase steady-state flow

in a porous medium is thus the following:

$$\begin{cases} -\lambda\nabla\cdot(K\nabla p) = f\,, & in\ \Omega \\ p = g\,, & on\ \Gamma_D \\ -\lambda K\dfrac{\partial p}{\partial \mathbf{n}} = h\,, & on\ \Gamma_N \end{cases} \tag{3}$$

where $\lambda = \dfrac{1}{\mu}$ is the fluid mobility, $K = K(\mathbf{x})$, $\mathbf{n}$ is the outward normal, g is the prescribed pressure on the Dirichlet boundary and h is the prescribed flow on the Neumann boundary.

2.1 Time dependent problem

The code is able to treat also the following time-dependent problem:

$$c\phi\frac{\partial p}{\partial t} - \lambda\nabla\cdot(K\nabla p) = f\,, \quad in\Omega\,, \tag{4}$$

completed with suitable boundary and initial conditions, where c is the compressibility of the fluid and ϕ the porosity of the medium.

3 The numerical scheme

To solve equations (3) and (4), we employ a Finite Volume (FV) (see, e.g., [2]), method in which the flux is approximated with a two-point scheme.

3.1 Finite volume method

In this section, we recall the FV scheme both in the case of the steady and unsteady problem. Let us consider a discretization of the domain Ω by a finite set of polyhedra, also called control volumes, $\{\Omega_i\}_1^N$ such that $\Omega_i \subset \Omega$, $\cup_i\Omega_i = \Omega$ and $\Omega_i \cap \Omega_j = \varnothing$ for $i \neq j$, see Figure 1. The degrees of freedom are cell centered, so for each control volume we have one degree of freedom for the pressure, placed at the centroid.

We approximate properties such as permeability and porosity as piecewise constant functions, more precisely, they are constant on each cell.

We integrate the Darcy equation

$$-\nabla\cdot(\lambda K\nabla p) = f, \quad \forall\mathbf{x}\in\Omega\,,$$

on each control volume Ω_i:

$$-\int_{\Omega_i}\nabla\cdot(\lambda K\nabla p) = \int_{\Omega_i} f, \quad i = 1,\ldots,N.$$

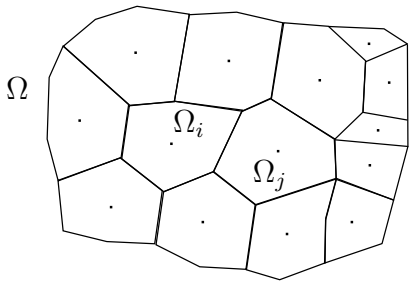

Figure 1: Partition of the domain Ω.

Then, we apply the divergence theorem on the left side, obtaining

$$-\int_{\Omega_i} \nabla\cdot(\lambda K\nabla p) = -\int_{\partial\Omega_i} \lambda K\nabla p\cdot\mathbf{n} = -\sum_{j\in\mathcal{N}_i}\int_{A_j} \lambda K\nabla p\cdot\boldsymbol{n}_{ij}, \quad i=1,\dots,N$$

where $\mathcal{N}_i$ is the set of neighboring control volumes of Ω_i, A_j is the interface polygon between Ω_i and Ω_j and $\boldsymbol{n}_{ij}$ is the outward normal from Ω_i to Ω_j.

We get

$$-\sum_{j\in\mathcal{N}_i}\int_{A_j} \lambda K\nabla p\cdot\boldsymbol{n}_{ij} = \int_{\Omega_i} f, \quad i=1,\dots,N.$$

Finally, we approximate the integral at the left hand side as

$$\int_{A_j} \lambda K\nabla p\cdot\boldsymbol{n}_{ij} \approx H_{ij}\left(p_i,p_j\right), \tag{5}$$

where H_{ij} is the numerical flux between control volumes Ω_i and Ω_j. It has to satisfy the following property

$$H_{ij}(p_i,p_j) = -H_{ji}(p_j,p_i).$$

The right hand side is approximated as

$$\int_{\Omega_i} f \approx f_i\,, \tag{6}$$

by a numerical quadrature rule.

The resulting scheme reads:

$$-\sum_{j\in\mathcal{N}_i} H_{ij}\left(p_i,p_j\right) = f_i \quad i=1,\dots,N. \tag{7}$$

In the case of imposition of Dirichlet boundary condition (imposed pressure) the set is extended to the faces on the Dirichlet boundary through the

technique of the "ghost cell" that will be presented later on. In the case on Neumann boundary conditions (imposed flux), the right hand side is suitably modified.

There are several ways to define the numerical flux H. The one we use in `FVCode3D` is presented in the next section.

Let us now consider the time dependent equation (4).
Let Δt the time step, and let us perform, for example, an implicit Euler discretization

$$\int_{\Omega_i} \frac{c\phi}{\Delta t}\left(p^{n+1}-p^n\right) - \sum_{j\in\mathcal{N}_i}\int_{A_j} \lambda K \nabla p^{n+1}\cdot \boldsymbol{n}_{ij} = \int_{\Omega_i} f^{n+1}\,, \quad i=1,\dots,N\,,$$

that yields

$$\frac{c\phi}{\Delta t}\int_{\Omega_i} p^{n+1} - \sum_{j\in\mathcal{C}_i} H_{ij}\left(p_i^{n+1}, p_j^{n+1}\right) = f_i^{n+1} + \frac{c\phi}{\Delta t}\int_{\Omega_i} p^n\,, \quad i=1,\dots,N.$$

Other possible time schemes are the Backward Difference Formulas (BDF), which is an implicit multistep method.
The BDF method applied to the unsteady Darcy equation (4) gives

$$\int_{\Omega_i} c\phi\left(\frac{\alpha_0}{\Delta t}p^{n+1} - \sum_{s=1}^{r}\frac{\alpha_s}{\Delta t}p^{n+1-s}\right) - \sum_{j\in\mathcal{N}_i}\int_{A_j}\lambda K\nabla p^{n+1}\cdot\boldsymbol{n}_{ij} = \int_{\Omega_i} f^{n+1}\,, \quad i=1,\dots,N\,,$$

where r is the order of the method and the coefficients α_s are determined such that:

$$\frac{\partial p}{\partial t}|_{t=t^{n+1}} = \frac{\alpha_0}{\Delta t}p^{n+1} - \sum_{s=1}^{r}\frac{\alpha_s}{\Delta t}p^{n+1-s} + \mathcal{O}\left(\Delta t^r\right).$$

For example, the BDF2 ($r=2$) method has $\alpha_0=\frac{3}{2}$, $\alpha_1=2$ and $\alpha_2=-\frac{1}{2}$. When $r=1$, the coefficients are $\alpha_0=\alpha_1=1$, i.e., the implicit Euler method.

`FVCode3D` implements both the implicit Euler and the BDF2 methods.

3.2 Two point flux approximation scheme

The flux is approximated by means of a two-point scheme. Therefore, the numerical flux introduced in (5) is defined as follows

$$H_{ij}(p_i,p_j) = \lambda T_{ij}\left(p_j - p_i\right) \tag{8}$$

where T_{ij} is the transmissibility between cells Ω_i and Ω_j.

The numerical scheme (7) becomes

$$-\sum_{j\in\mathcal{N}_i} \lambda T_{ij}\,(p_j - p_i) = f_i, \quad i = 1,\dots,N.$$

The corresponding algebraic formulation can be written as

$$Tp = \tilde{f} \tag{9}$$

where T is the stiffness matrix and $\tilde{f} = f + b$ is the source/sink term with the addition of the terms due to boundary conditions.
In the case of the time-dependent equation with a generic BDF time scheme of order r, the algebraic formulation is the following

$$\frac{\alpha_0}{\Delta t} M p^{n+1} + T p^{n+1} = \tilde{f}^{n+1} + \sum_{s=1}^{r} \frac{\alpha_s}{\Delta t} M p^{n+1-s} \tag{10}$$

where M is a diagonal mass matrix with $M_{ii} = c\phi|\Omega_i|$.

3.3 Treatment of the boundary conditions

The boundary conditions considered are of Dirichlet and Neumann type. Here, we describe the numerical treatment of these types of boundary conditions.
In the case of Dirichlet boundary conditions (see Figure 2):

1. we create a ghost cell beside the boundary cell;

2. we consider the degree of freedom associated with the ghost cell, $\mathbf{s_g}$, as placed at the same distance of $\mathbf{s_j}$ from the boundary and such that the line that connects $\mathbf{s_j}$ to the centroid of the boundary facet is orthogonal to the boundary;

3. the numerical flux is $H_{jg} = \lambda T_{jg}\,(p_g - p_j)$.

In the case of a Neumann boundary condition, we simply add to the right hand side at position j the associated flux:

$$b_j = h_g \cdot A_j$$

where h_g is the value of the boundary condition and A_j is the boundary facet area.

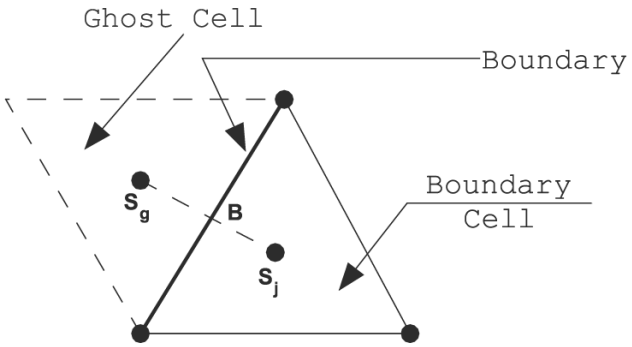

Figure 2: Sketch of the configuration on the boundary ($\mathbf{B}$). $\mathbf{s}_g$ is the centroid of the ghost cell, while $\mathbf{s}_j$ is the centroid of the cell adjacent to the boundary.

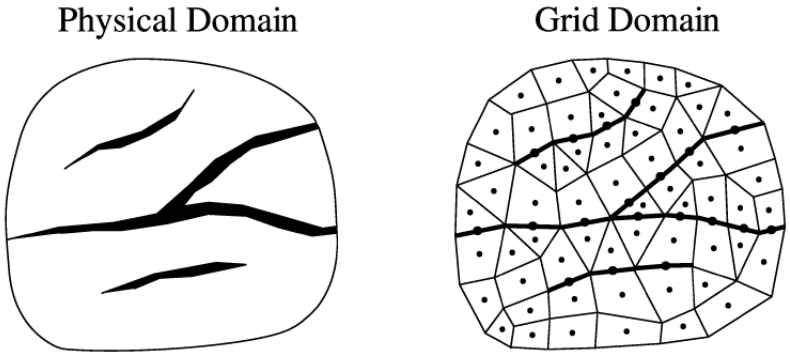

Figure 3: (Left) Representation of the fractures in the physical domain. (Right) Representation of the fractures in a geometrical point of view.

3.4 Representation of the fractures

For the fractures, we employ the same model used for the porous medium:

$$\begin{cases} -\lambda \nabla \cdot (K \nabla p) = f \,, & in\ \Omega_f \\ \\ \lambda K \dfrac{\partial p}{\partial \mathbf{n}} = 0 \,, & on\ \partial\Omega_f \end{cases} \tag{11}$$

where Ω_f indicates the domain associated with the fractures.
From a geometric point of view, the fractures are modeled as 2D surfaces (a set of planar facets), and are represented by a subset of facets of the mesh. This means that the fractures are matching with the mesh, see Figure 3.

However, we need to account that fractures have a thickness (also called aperture), so from the physical point of view they are modeled as 3D entities,

see Figure 4 and Figure 5.

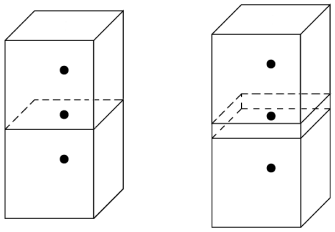

Figure 4: (Left) A fracture facet between two control volumes in a geometrical point of view. (Right) The fracture is thickened.

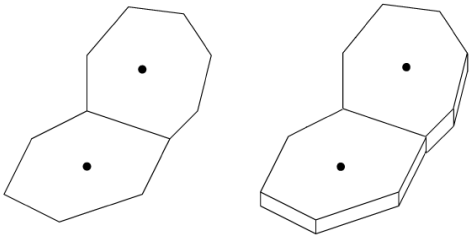

Figure 5: (Left) Two adjacent fractures in a geometrical point of view. (Right) The fractures are considered as 3D entities.

3.5 Transmissibility

We have seen that the numerical flux H_{ij} is defined as in equation (8), where T_{ij} indicates the transmissibility between cells i and j.

According to [1], the transmissibility between two control volumes is defined as

$$T_{12} = \frac{\beta_1 \beta_2}{\beta_1 + \beta_2}\,, \tag{12}$$

with

$$\beta_i = \frac{A_i K_i}{D_i} \boldsymbol{n_i} \cdot \boldsymbol{f_i}\,, \quad i = 1, 2\,, \tag{13}$$

where A_i is the interface area between the control volumes considered, D_i is the distance between the facet centroid and the cell centroid of the cell i, K_i is the permeability inside the cell i, $\mathbf{n}_i$ the inward normal and $\mathbf{f}_i$ the unit vector that ties the facet centroid to the cell centroid, see Figure 6.

In the case of a fractured porous medium we have three types of transmissibility: matrix-matrix, matrix-fracture and fracture-fracture transmissibility. The matrix-fracture and fracture-fracture transmissibility is computed in the same way as in the case of matrix-matrix transmissibility.

The only difference is that the interface area, A_i and A_j, between two adjacent fractures, i and j, is computed as the interface line multiplied by the aperture of the fractures.

A special treatment is required when three or more fractures intersect in one line, see Figure 7. In this case the definition given in equation (12) needs to be generalized. The idea is to apply an analogy between flow of a fluid through porous media and flow of electric charge through a network of resistors and use the "star-delta" transformation. Hence, the transmissibility between two of these fractures is computed as follow:

$$T_{ij} = \frac{\beta_i \beta_j}{\sum_{k=1}^{N_f} \beta_k} \tag{14}$$

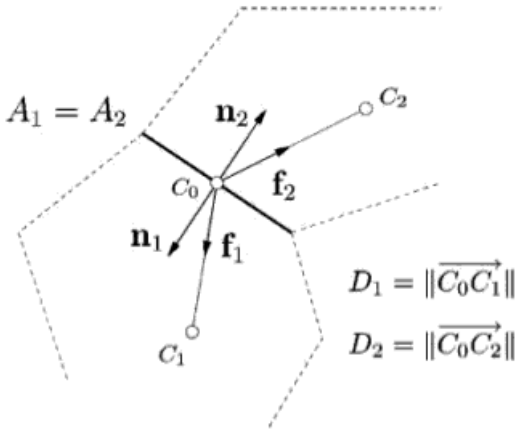

Figure 6: The quantities involved in the computation on the transmissibility between control volumes: the centroids C_1 , C_2, the normal vectors of the facets $\mathbf{n}_1$, $\mathbf{n}_2$ and the interface area $A_1 = A_2$.

where N_f is the number of fractures that intersected in the same line.

4 Examples

In this sections we show two examples of numerical simulations obtained with `FVCode3D`. In the first example, we consider a steady-state problem on a hexahedral mesh, while in the second one we consider an unsteady problem with a fixed value of pressure inside the fractures.

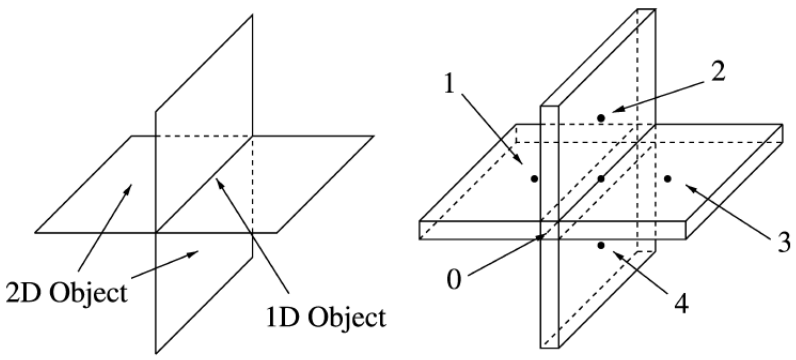

Figure 7: Intersection of four fracture facets.

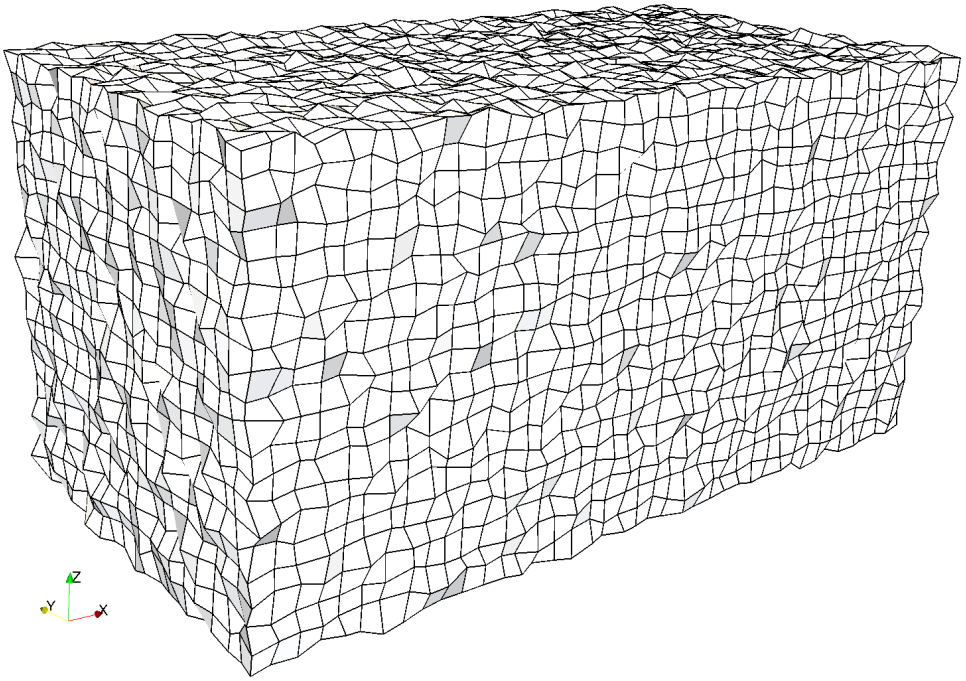

Figure 8: Hexahedral grid used to solve the steady-state problem.

4.1 A steady-state problem

Let us consider a steady-state problem on the domain shown in Figure 8 and 9 with the pressure fixed equal to 1 to the left side, equal to 0 to the right side of the domain and a no-flux condition on the remaining part of the boundary domain.

The formulation is the following:

$$\begin{cases} -\lambda \nabla \cdot (K \nabla p) = f\,, & in\ \Omega \\ p = 1\,, & on\ \Gamma_{left} \\ p = 0\,, & on\ \Gamma_{right} \\ \lambda K \dfrac{\partial p}{\partial \mathbf{n}} = 0\,, & on\ \Gamma_N \end{cases}$$

where $\partial\Omega = \Gamma_{left} \cup \Gamma_{right} \cup 1, \Gamma_N$, $c = 1$, $\lambda = 1$, the porous medium permeability $K_m = K|_{\Omega_m} = 1$, the fractures permeability $K_f = K|_{\Omega_f} = 10^6$, $\phi_m = \phi|_{\Omega_m} = 0.25$, $\phi_f = \phi|_{\Omega_f} = 1$, the fractures aperture $a = 10^-2$.

The solution is shown in Figure 10 and 11.

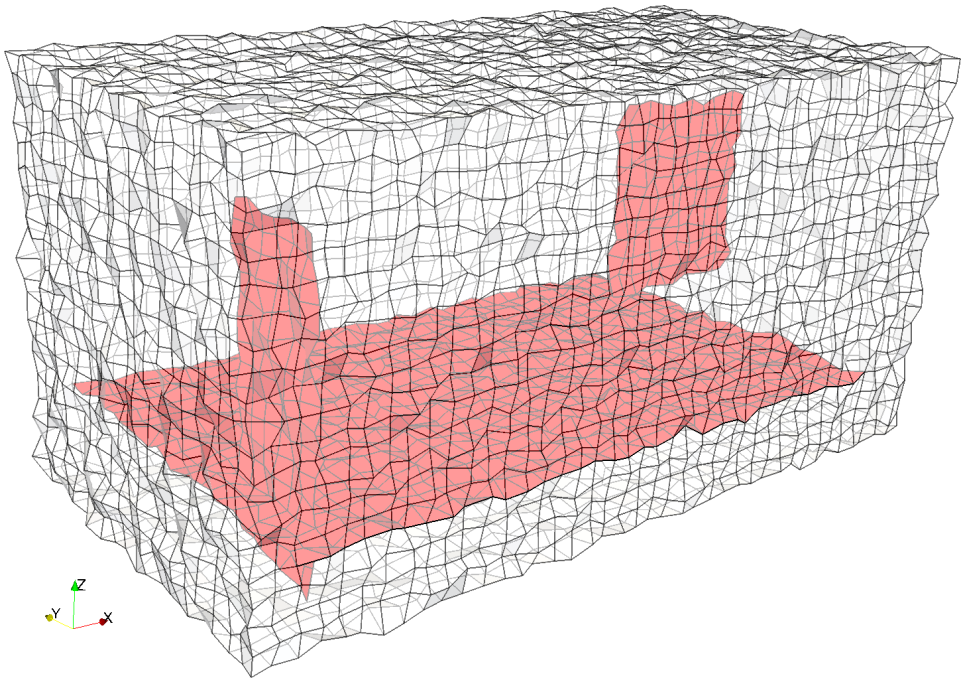

Figure 9: Fractures considered for the steady-state problem.

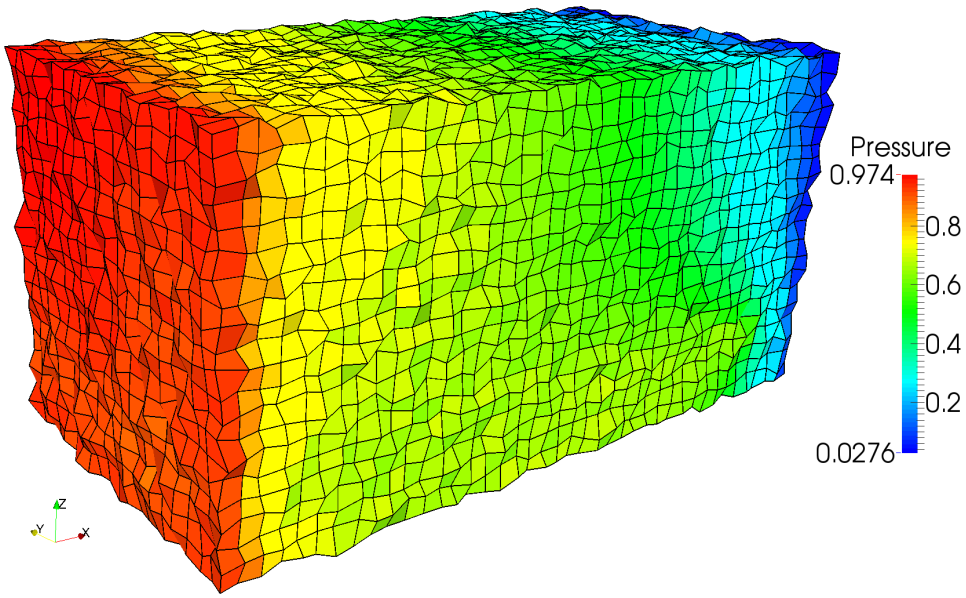

Figure 10: Solution of the steady-state problem on the porous medium.

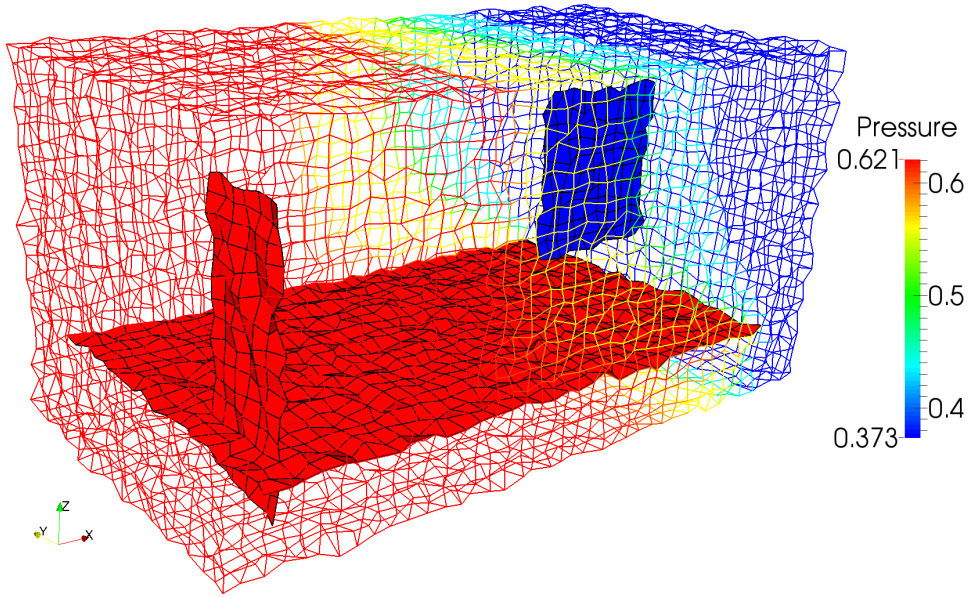

Figure 11: Solution of the steady-state problem on the fractures.

4.2 A time dependent problem

Let us consider the following unsteady problem:

$$\begin{cases} c\phi \dfrac{\partial p}{\partial t} - \lambda \nabla \cdot (K \nabla p) = f\,, & in\ \Omega \times [0,T] \\ \lambda K \dfrac{\partial p}{\partial \mathbf{n}} = 0\,, & on\ \partial\Omega \\ p = 1\,, & in\ \Omega_f \end{cases}$$

where $\Omega = \Omega_m \cup \Omega_f$ is the union of the domain occupied by the porous medium and the domain occupied by the fractures, $c = 1$, $\lambda = 1$, the porous medium permeability $K_m = K|_{\Omega_m} = 1$, the fractures permeability $K_f = K|_{\Omega_f} = 10^6$, $\phi_m = \phi|_{\Omega_m} = 0.25$, $\phi_f = \phi|_{\Omega_f} = 1$, the fractures aperture $a = 10^-2$, $T = 2 \cdot 10^6$ and $\Delta t = 2 \cdot 10^4$.

The domain considered, shown in Figure 12, is composed of three layers of cells along z-direction. The cells are triangular and quadrilateral prism. The solution at different time step is shown in Figure 13.

5 Structure of the code

The code is written in C++. As input, the program requires a mesh as '.fvg' format file. The output is given as '.vtu' file format, which can be viewed

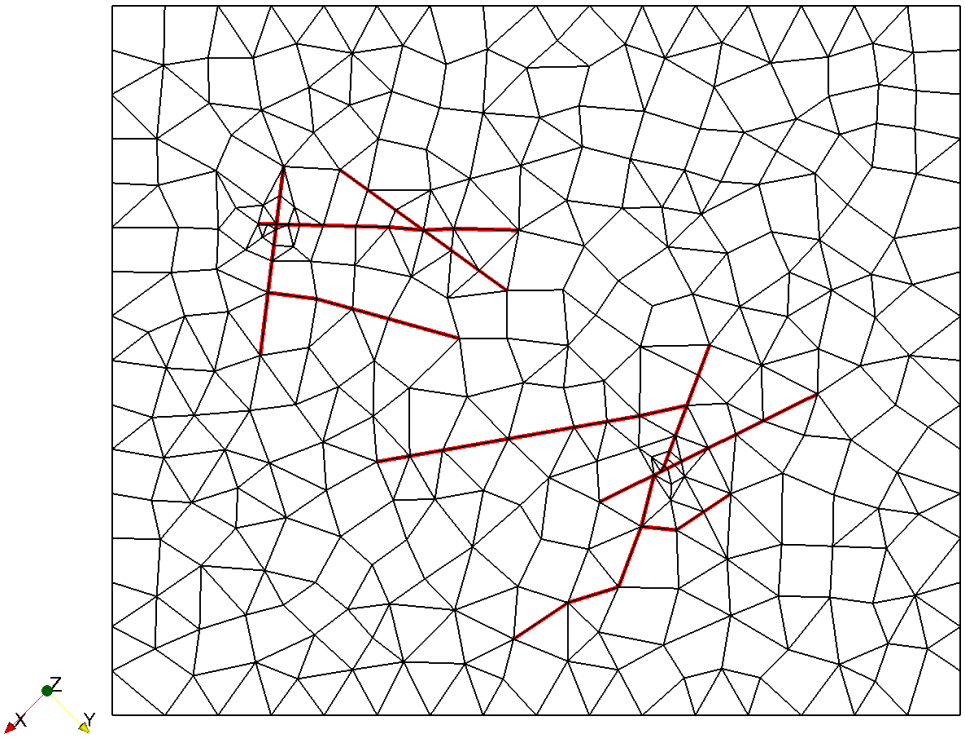

Figure 12: Mesh of the domain and fractures used to solve the time-dependent problem.

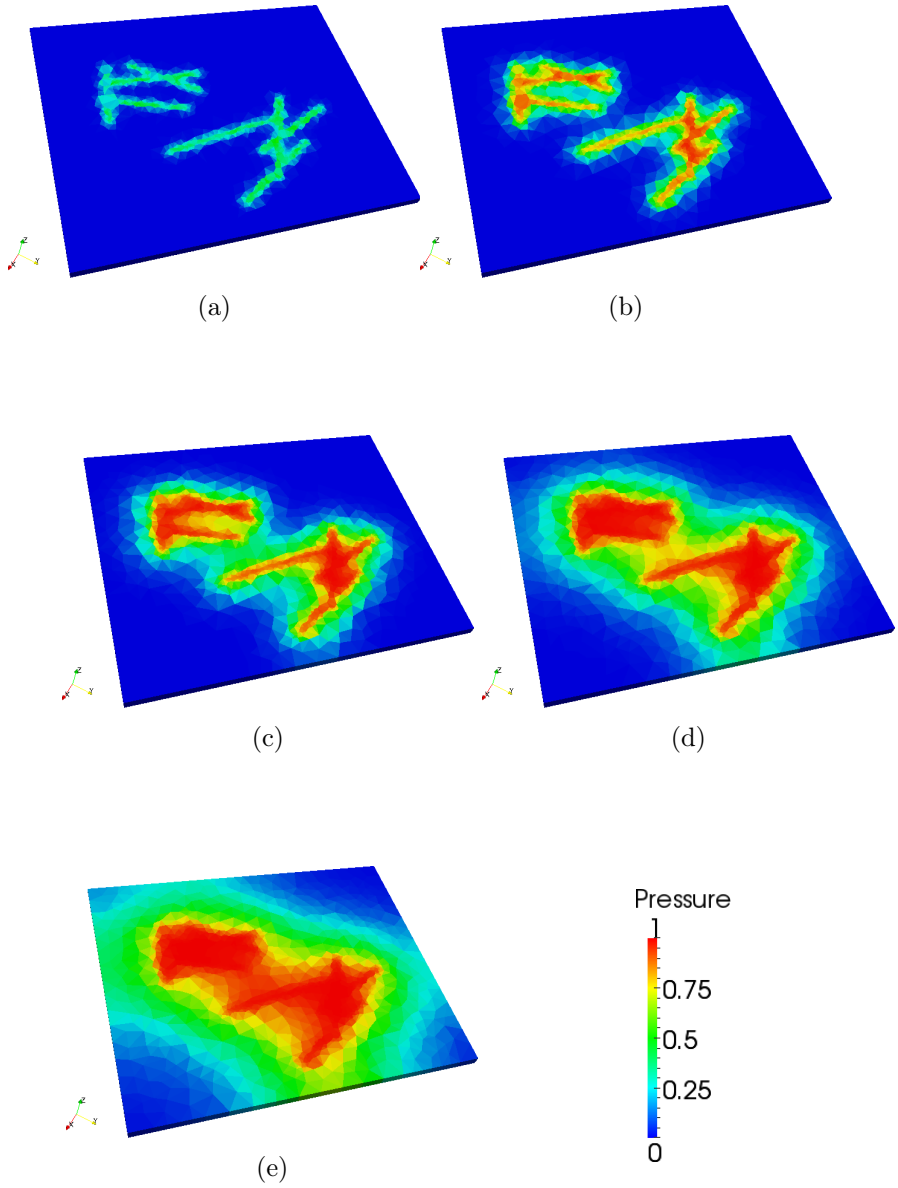

Figure 13: Solution of a time dependent problem of a fractured porous medium at different time step.(a): $t = 0$. (b): $t = 10^5$. (c): $t = 3 \cdot 10^5$. (d): $t = 8 \cdot 10^5$. (e): $t = 2 \cdot 10^6$.

with Paraview.

5.1 Installation

To install the program simply enter in the source directory and execute the install.sh script:

```
cd FVCode3D-source
./install.sh --prefix=../FVCode3D-install
```

5.2 How to execute the program

To run the program just execute

```
./fvcode3d.exe
```

By default, the parameters need are read from data.txt file.
With the option '-f ', it is possible to specify the data file.
Ex.

```
./fvcode3d.exe -f data.txt
```

5.3 Input parameters

The program requires:

- an input mesh in '.fvg' format
- the data file
- the definition of the source/sink function and the functions of the boundary condition

The file format '.fvg' consists of four parts:

- list of POINTS, each of them described by its coordinates x, y, z;
- list of FACETS, each of them described by a list of ids of POINTS, plus some properties if the facet represents a fracture;
- list of CELLS, each of them described by a list of ids of FACETS, plus some properties;
- list of FRACTURE NETWORKS, each of them described by a list of FACETS that belong to the same network.

The properties related to the facets that represent a fracture are the permeability, the porosity and the aperture.
The properties related to the cells (porous medium) are the permeability and the porosity.

The data file consists of six sections:

mesh the input file parameters such as mesh location and filename;

output the output directory and filename;

problem parameters related to the problem, such as the problem type (steady or unsteady) or if apply or not the source/sink term;

fluid mobility and compressibility of the fluid;

bc the rotation to apply around the z axis, needed to correctly apply the boundary conditions;

solver the solver to use and some parameters such as the tolerance and the number of iterations.

Below, we give a brief description for each parameter:

Parameters of the datafile

```
[mesh]
mesh_dir    = ./data/       // mesh directory
mesh_file   = cartBC1.fvg   // mesh filename
mesh_type   = .fvg

[output]
output_dir  = ./results/    // output directory
output_file = sol           // output prefix of the files

[problem]
type        = steady        // steady or pseudoSteady
fracturesOn = 1             // 1 enable fractures, 0 disable fractures
sourceOn    = none          // where the source is applied: all, matrix,
    fractures, none
fracPressOn = 0             // 1 set pressure inside fracture, 0 otherwise
fracPress   = 1.            // if 'fracPressOn = 1', this set the value of
    the pressure
initial_time= 0.            // if 'type = pseudoSteady', this set the
    initial time
end_time    = 2.e6          // if 'type = pseudoSteady', this set the final
    time
time_step   = 1.e5          // if 'type = pseudoSteady', this set the time
    step

[fluid]
mobility    = 1.            // mobility of the fluid
compressibility = 1.        // compressibility of the fluid

[bc]
theta       = 0.            // rotation to apply to the grid around the z
    axis

[solver]
type    = EigenUmfPack      // solver:  EigenCholesky
                            //          EigenLU
```

```
                            //          EigenUmfPack
                            //          EigenCG
                            //          EigenBiCGSTAB

    [./iterative]
    maxIt       = 1000      // max iterations of the iterative solver
    tolerance   = 1e-5      // tolerance of the iterative solver
```

Finally, the functions used to set the source/sink term and the boundary conditions are defined in `functions.hpp`.

For example, the following line

Definition of a function that depends on the spatial coordinates

```
Func SourceDomain = [](Point3D p){return 5. * (p.x()*p.x() + p.y()*p.y() + p
    .z()*p.z()) < 1;};
```

defines a function equal to 5 inside the sphere of radius 1 centered in the origin, zero elsewhere.

Instead, these lines define the functions $f(\mathbf{x}) = 0$, $f(\mathbf{x}) = 1$ and $f(\mathbf{x}) = -1$.

Some definitions of constant functions

```
Func fZero = [](Point3D){ return 0.; };
Func fOne = [](Point3D){ return 1.; };
Func fMinusOne = [](Point3D){ return -1.; };
```

5.4 Main Classes

The main classes involved are the following:

Data collects all the parameters used by the program that can be set through the datafile;

Importer reads the input file mesh;

PropertiesMap collects the properties of the fractures and porous medium;

Mesh3D stores the geometrical mesh as Point3D, Facet3D and Cell3D;

Rigid_Mesh converts a Mesh3D in a rigid format, suitable for assemble the problem;

BoundaryConditions defines the boundary conditions to assign to the problem; BCs can be Dirichlet or Neumann;

Problem defines the problem to solve: it can be a steady problem (DarcySteady) or a time-dependent problem (DarcyPseudoSteady); it assembles the matrix and the right hand side;

Solver solves the Problem; the code implements direct solvers as well as iterative solvers;

Exporter exports as '.vtu' files the mesh, the properties, the solution and more.

5.5 Tutorial

Here, we briefly describe how to use the code.
First of all, read the necessary parameters from the datafile:

Read data from datafile

```
// we use the GetPot utility to access the datafile
GetPot command_line(argc,argv);
const std::string dataFileName = command_line.follow("data.txt", 2, "-f",
  "--file");
// Read Data from the datafile
```

Then we define the Mesh3D, the PropertiesMap so that we can import the mesh:

Import mesh grid and properties

```
//Define Mesh and Properties
Mesh3D mesh;
PropertiesMap propMap(dataPtr->getMobility(), dataPtr->getCompressibility
  ());
//Create Importer
Importer * importer = 0;
importer = new ImporterForSolver(dataPtr->getMeshDir() + dataPtr->
  getMeshFile(), mesh, propMap);
//Import grid file
importer->import(dataPtr->fractureOn());
```

After the reading of the mesh, it is necessary to perform some operations to process the mesh:

Process the grid

```
//Compute the cells that separate each facet
mesh.updateFacetsWithCells();
//Compute the neighboring cells of each cell
mesh.updateCellsWithNeighbors();
//Set labels on boundary (necessary for BCs)
importer->extractBC(dataPtr->getTheta());
//Compute facet ids of the fractures (creates fracture networks)
mesh.updateFacetsWithFractures();
```

We can now create the boundary conditions:

Create boundary conditions

```
//Assign to the border marked as 'Left' a Dirichlet condition equal to 1
BoundaryConditions::BorderBC leftBC (BorderLabel::Left, Dirichlet, fOne );
```

```
//And so on...
BoundaryConditions::BorderBC rightBC(BorderLabel::Right, Dirichlet, fZero
  );
...

//We create a vector that contains all the BorderBC...
std::vector<BoundaryConditions::BorderBC> borders;
//...and we insert into it the BCs previously created
borders.push_back( backBC );
borders.push_back( frontBC );
...

//Finally we create the BoundaryConditions
BoundaryConditions BC(borders);
```

Then we create a Rigid_Mesh:

Create the Rigid_Mesh

```
//The Rigid_Mesh requires the Mesh3D and the PropertesMap
Rigid_Mesh myrmesh(mesh, propMap);
```

We can proceed by building the problem:

Build the problem

```
Problem<CentroidQuadrature, CentroidQuadrature> * darcy(nullptr);
typedef DarcySteady<CentroidQuadrature, CentroidQuadrature> DarcyPb;
typedef DarcyPseudoSteady<CentroidQuadrature, CentroidQuadrature, SpMat,
  TimeScheme::BDF2> PseudoDarcyPb;

//If the problem is steady
if(dataPtr->getProblemType() == Data::ProblemType::steady)
{
    darcy = new DarcyPb(dataPtr->getSolverType(), myrmesh, BC, SS, dataPtr
  );
}
//If the problem is unsteady
else if(dataPtr->getProblemType() == Data::ProblemType::pseudoSteady)
{
    darcy = new PseudoDarcyPb(dataPtr->getSolverType(), myrmesh, BC, SS,
  dataPtr);
}

//If the solver used is an iterative one, set tolerance and the maximum
  number of iterations
if(dynamic_cast<IterativeSolver*>(darcy->getSolverPtr()))
{
    dynamic_cast<IterativeSolver*>(darcy->getSolverPtr())->setMaxIter(
  dataPtr->getIterativeSolverMaxIter());
    dynamic_cast<IterativeSolver*>(darcy->getSolverPtr())->setTolerance(
  dataPtr->getIterativeSolverTolerance());
}
```

Now, we can solve the problem:

Solve the problem

```
//In the case of a steady problem, we assemble and the solve the problem
if(dataPtr->getProblemType() == Data::ProblemType::steady)
{
    darcy->assemble();
    if(dataPtr->pressuresInFractures())
    {
        FixPressureDofs<DarcyPb> fpd(dynamic_cast<DarcyPb *>(darcy));
        fpd.apply(dataPtr->getPressuresInFractures());
    }
    darcy->solve();
}
//In the case of a unsteady problem, we initialize the problem and for
   each time step, we assemble and solve the
problem
else if(dataPtr->getProblemType() == Data::ProblemType::pseudoSteady)
{
    dynamic_cast<PseudoDarcyPb *>(darcy)->initialize();
    for(Real t = dataPtr->getInitialTime() + dataPtr->getTimeStep() ; t <=
    dataPtr->getEndTime(); t+=dataPtr->getTimeStep())
    {
        darcy->assemble();
        if(dataPtr->pressuresInFractures())
        {
            FixPressureDofs<PseudoDarcyPb> fpd(dynamic_cast<PseudoDarcyPb
   *>(darcy));
            fpd.apply(dataPtr->getPressuresInFractures());
        }
        darcy->solve();
    }
}
```

The solution can be exported:

Export the solution

```
//Define Exporter
ExporterVTU exporter;
//Export solution on matrix and on the fractures
exporter.exportSolution(myrmesh, dataPtr->getOutputDir() + dataPtr->
   getOutputFile() + "_solution.vtu", darcy->getSolver().getSolution());
exporter.exportSolutionOnFractures(myrmesh, dataPtr->getOutputDir() +
   dataPtr->getOutputFile() + "_solution_f.vtu", darcy->getSolver().
   getSolution());
```

Remember to delete the Problem and the Importer instance:

Delete instances

```
delete darcy;
delete importer;
```

5.6 Generation of the reference manual

The installation procedure described in Section 5.1 also produces a more detailed technical documentation in pdf and html format. To install the documentation, the Doxygen package is required. The LaTeX typesetting system is required if one needs a pdf file.

Doxygen can be downloaded at `http://www.stack.nl/~dimitri/doxygen/index.html`.

A configuration file (in the following indicated as doxyfile), called `DoxyFile.in` is contained in the `doc` directory and it allows to personalize the documentation. The doxyfile may be changed if one wants the documentation in a different format (for instance rtf) or change some of the output layout. To this aim, one may use the `doxywizard` command (normally installed with the doxygen package, sometimes with a different package called `doxygen-gui`). It provides a nice graphical interface.

The reference manual may contain diagrams illustrating the relations among the different classes. To generate them you may use the inbuilt diagram generator or use the tools provided by the `graphviz` package (`http://www.graphviz.org`), which gives a nicer output.

The documentation, available both in html and latex format, can be found in `doc/html` and in `doc/latex`. To look at the html documentation you should load the file `index.html` in the `doc/html` directory. To produce a pdf file from the LaTeX sources you should run `make` from the `doc/latex` directory (the installation tools does it automatically).

References

[1] K. Aziz M. Karimi-Fard, L.J. Durlofsky, *An efficient discrete-fracture model applicable for general-purpose reservoir simulation*, SPE Journal **9** (2004), no. 2, 227–236.

[2] R. Herbin R. Eymard, T. Gallouët, *The finite volume method*, Handbook of Numerical Analysis (Ph. Ciarlet and J.L. Lions, eds.), vol. 7, North Holland, 2000, pp. 713–1018.