

RESEARCH FIELD OPTION

CASE STUDY 5

Comparative study of numerical tools to solve closure problems in porous media

Authors

Thomas GAUTHEY, Clément CÔTE, Emmanuel CASTIEL,
Yingjie ZHAO, Wassim BOURBIA

Supervisors

Morgan CHABANON, Bich-Lien DOAN, Cristina MANIU

Contents

1	Introduction and context	1
1.1	Description of the problem	1
1.2	Objectives of the study	2
2	Methods	3
2.1	Equation	3
2.2	Protocol	3
2.3	Metrics	3
2.4	How time execution of the programs is computed ?	4
3	Software programs	6
3.1	OpenFOAM	6
3.2	NGS-Py Finite Element Tool	9
3.3	FEniCS project	12
4	Results	14
4.1	Mésocentre	14
4.2	Comparison	14
5	Conclusion	15
5.1	Which one should you choose?	15

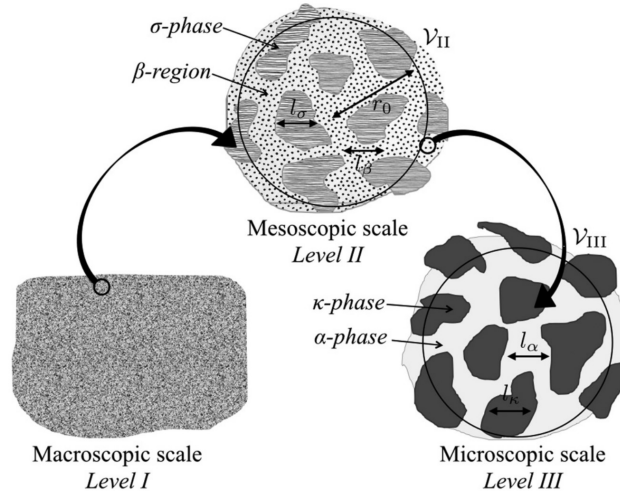
1 Introduction and context

1.1 Description of the problem

Porous media are spaces occupied by multi phase materials. It contains relatively small spaces, so-called pores or voids, free of solids, embedded in the solid or semisolid matrix, and it is permeable to a variety of fluids, which means fluids should be able to penetrate through one face of a septum made of the material and emerge on the other side[1].

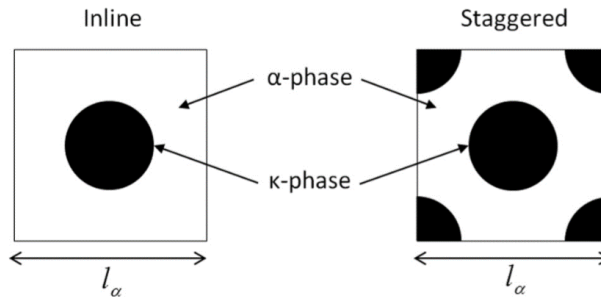
Porous materials are ubiquitous in engineering processes regarding energy and health, from petroleum industry to bio engineering. In such systems, studying the viscous flow mechanism is of great interest. However, the structure is too complex for direct analysis of transport equations within the pores at the macroscopic scale. We can solve transport equations in a pore with numerical tools at the microscopic scale, but deploying such a method at bigger scaler would cost lots of computing resources and time. So the homogenization is performed to handle this difficulty. It is based on volume-averaging method which derive the averaged physical properties in a representative volume and extends them to a larger scale.

Figure 1: From microscopic to macroscopic[2]



The reason to use homogenization is to rigorously derive continuum equations for multi phase systems[7]. And the key issue is to use local volume averaged equations which are valid everywhere to simplify the calculation. The chosen scale of the average volume must be representative enough to be applied to the whole media. To simplify this process, we take the elementary volume as below, as well as performing the homogenization on it. After volume averaging, Darcy's law can be applied to such a scale by solving the closure problem[7].

Figure 2: Simplified elementary volume[2]



The radius of the κ -phase is noted as r_1 , and the radius of the quarter circle in the corner is noted as r_2 .

1.2 Objectives of the study

The objective is to carry out a comparative study of different numerical tools for solving the closure problem. It consists of 3 tasks as below:

- *Get started with different finite elements software programs :* The first objective is to get familiar with OpenFOAM, but we also propose some other tools to explore, such as NGSolve and FEniCS.
- *Solve the closure problem :* We apply proper solvers in different pieces of software to solve the same closure problem.
- *Compare their performance :* We establish metrics to evaluate these programs among different dimensions, and rank them.

2 Methods

2.1 Equation

The velocity field is determined with the continuity equation by incompressible, steady Stokes equations, and non-slip boundary[7].

$$\begin{cases} \nabla \cdot v_\alpha = 0 \\ 0 = -\nabla p_\alpha + \mu_\alpha \nabla^2 v_\alpha \\ v_\alpha = 0 \end{cases} \quad \text{at } \mathcal{A}_{\alpha\kappa} \quad (1)$$

The v_α , p_α and μ_α are the velocity, pressure and viscosity in α -phase. And $\mathcal{A}_{\alpha\kappa}$ is the interface between solid and fluid.

The homogenization based on volume-averaging is then performed as below[7],

$$\langle \psi \rangle = \frac{1}{V_\alpha + V_\kappa} \int_{V_\alpha + V_\kappa} \psi_\alpha dV \quad (2)$$

where ψ is the arbitrary physical quantity which can be v and p . V_α and V_κ are the volume of corresponding phase.

Then we can apply Darcy's law to this volume,

$$\langle v \rangle = -\frac{\underline{\underline{K}}}{\epsilon \mu_\alpha} \nabla \langle p \rangle \quad (3)$$

where $\langle v \rangle$ is the averaging velocity and $\langle p \rangle$ the averaging pressure. ϵ is the porosity defined as

$$\epsilon = \frac{V_\alpha}{V_\alpha + V_\kappa} \quad (4)$$

$\underline{\underline{K}}$ is the permeability tensor, the value of which is uniquely determined by the pore geometry and is independent of the properties of the penetrating fluid[1]. It's the function of closure variables, which lead to the closure problem.

We then characterize the permeability with these two configurations by solving the closure problem with equations[7] in α -phase below,

$$\begin{cases} 0 = -\nabla \underline{b} + \nabla^2 \underline{\underline{B}} + \underline{\underline{I}} \\ \nabla \cdot \underline{\underline{B}} = 0 \end{cases} \quad (5)$$

where \underline{b} and $\underline{\underline{B}}$ are closure variables. The boundary conditions are defined as below.

$$\begin{cases} \underline{\underline{B}} = 0 \text{ at } \partial\Omega_\alpha \cap \partial\Omega_\kappa \\ \text{periodic at } \partial\Omega_\alpha \setminus \partial\Omega_\kappa \end{cases} \quad (6)$$

The permeability tensor can be then derived by the equation below,

$$\langle B \rangle^\alpha = \epsilon^{-1} \underline{\underline{K}} \quad (7)$$

where $\langle B \rangle^\alpha$ is the averaging closure variable.

2.2 Protocol

The way we use to compare these tools is to begin by observing the values of $C = \frac{2}{9} \frac{r^2}{1-\epsilon} \frac{1}{K_{xx}}$

2.3 Metrics

Sur quoi nous allons les comparer ?

2.4 How time execution of the programs is computed ?

All the commands used to compute a single iteration of the program are put into a bash script. For instance, for FEniCS, the script `python_exec.sh` is the following:

```
#!/bin/bash
python3 OLI9_maillage3D.py > /dev/null 2> /dev/null
python3 OLI9_perm3D.py > /dev/null 2> /dev/null
```

The `> /dev/null 2> /dev/null` are there to avoid the log messages from the program. As we cannot see the logs, we must be sure that the program terminates.

This bash script is then called 50 times with the `time` command, to compute the execution time of a single bash script. For instance, for FEniCS, the slurm script is the following:

```
#!/bin/bash
#SBATCH --job-name=testFenics
#SBATCH --output=%x.o%j
#SBATCH --ntasks=1
#SBATCH --mem=32G
#SBATCH --time=01:00:00
#SBATCH --partition=cpu_short

# To clean and load modules defined at the compile and link phases
module purge
module load anaconda3/2021.05/gcc-9.2.0

# To compute in the submission directory
cd ${SLURM_SUBMIT_DIR}
cd TestFenics

source activate fenicsproject

# execution
for i in {0..49}
do
time ./python_exec.sh
done

conda deactivate
```

The resulting log file is then such as this:

```
real 0m28.448s
user 0m6.288s
sys 0m0.848s

real 0m7.753s
user 0m5.695s
sys 0m0.505s

real 0m8.999s
user 0m5.344s
sys 0m0.389s
```

```
real 0m6.302s
user 0m5.454s
sys 0m0.464s
...
```

The **real** time is the important one, and can be easily retrieved with a simple python script, and all the time can be processed to obtain the median and the maximum.

3 Software programs

3.1 OpenFOAM

Presentation

OpenFOAM (Open-source Field Operation And Manipulation) is a Linux software encompassing several 3D numerical solvers for continuum mechanics, and mainly CFD (computational fluid dynamics). These solvers are coded in C++, but OpenFOAM comes with its own programming language, used for three tasks : creating the mesh, choosing the solver (and its options), and the post-processing functions.

Usage on the Mésocentre

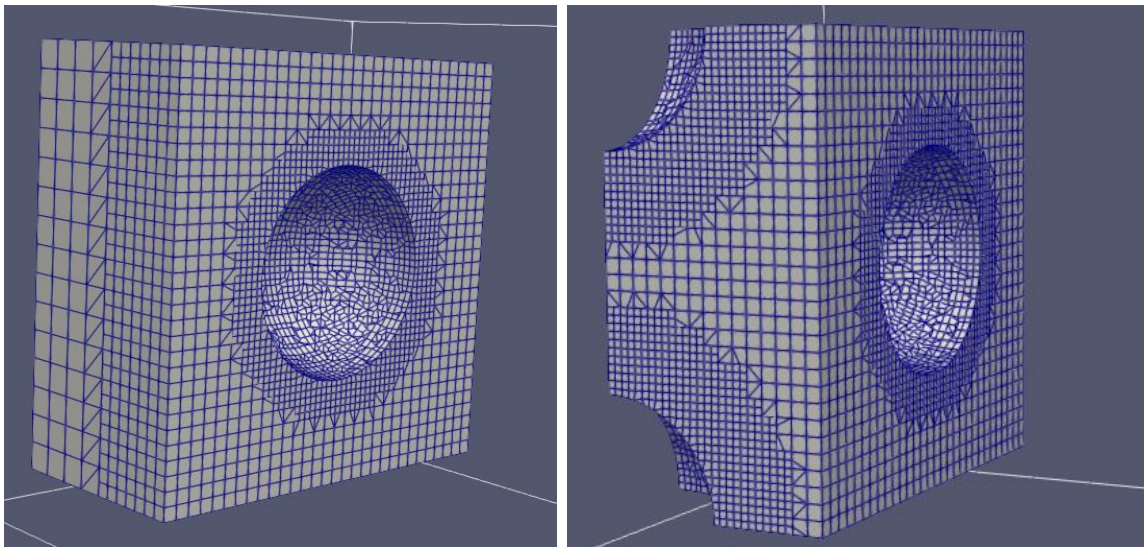
The module `openfoam/2106/gcc-9.2.0-openmpi` must be loaded:
`module load openfoam/2106/gcc-9.2.0-openmpi.`

The version **2106** is important, as tests with the other version *1912* were not successful. In the same way, the opensource version of OpenFOAM (`openfoam/6/intel-19.0.3.199` or `openfoam/8/intel-19.0.3.199`) are not suitable.

Mesh

The two geometries below have been created :

Figure 3: Meshes created by OpenFoam and visualized with Paraview

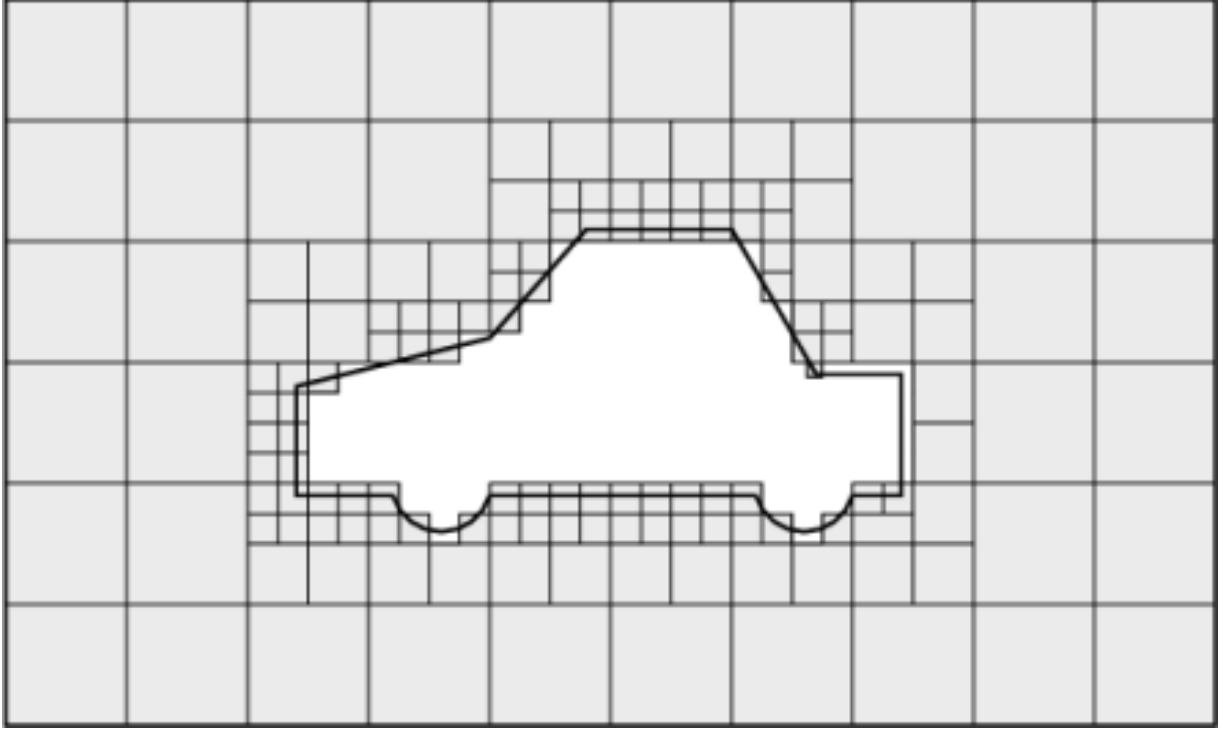


Several tools are available in OpenFoam to create a mesh ; the built-in utility `blockMesh` allows creating simple geometries and was used for the background cube.

To create a mesh with this utility, the geometry must first be split into different blocks, each one with exactly 8 vertices. Then, each block is meshed, and the user can choose the form (hexahedron, prism, wedge...), the number and expansion ratio of the elements for each direction.

For more sophisticated geometries, the utility `snappyHexMesh` can be used. A user-specified volume, defined for instance in an STL file (in our case, a sphere) is "removed" from the background mesh (defined with `blockMesh`). The cells that intersect the surface of the volume are refined, and all the elements that do not have at least 50 % of their volume outside the geometry are removed, as shown below :

Figure 4: Example of cell removal with snappyHexMesh



Solvers

Five solvers are available to solve Navier-Stokes equations for incompressible flow : `boundaryFoam`, `icoFoam`, `pimpleFoam`, `pisoFoam` and `simpleFoam`. `icoFoam` doesn't allow for a source term in the equation, so it was discarded. Since `simpleFoam` is one of the few solvers specialized for the steady-state equation, it was privileged. It does not solve exactly the equation we want, taking into account the inertial term $\nabla \cdot (\mathbf{u} \times \mathbf{u})$. However, this term should be of minor importance, since it was often 2 orders of magnitude below the other terms. This solver is based upon the SIMPLE algorithm. This algorithm seeks to solve the equation :

$$\begin{cases} \nabla \cdot (\mathbf{u} \times \mathbf{u}) - \nabla \cdot \mathbf{R} = -\nabla p + \mathbf{S} \\ \nabla \cdot \mathbf{u} = 0 \end{cases} \quad (8)$$

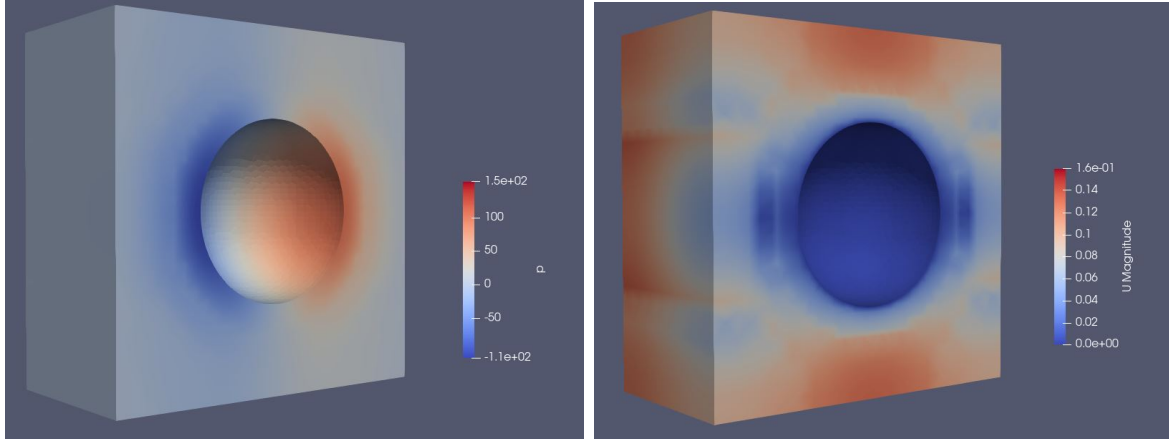
where \mathbf{u} is the velocity, p the pressure, \mathbf{R} the stress tensor and \mathbf{S} the source term. The algorithm works as follows :

1. Advance to the next time step t^{n+1}
2. Initialize \mathbf{u}^{n+1} and p^{n+1} using the latest available values
3. Discretize the momentum equation in the form $\mathbf{M}\mathbf{u} = -\nabla p$ (with \mathbf{M} a matrix), and write $\mathbf{M}\mathbf{u} = \mathbf{A}\mathbf{u} - \mathbf{H}$ with \mathbf{A} the matrix containing only the diagonal elements of \mathbf{M}
4. Use the new expression $\mathbf{u} = \mathbf{A}^{-1}\mathbf{H} - \mathbf{A}^{-1}\nabla p$ in the continuity equation, which becomes $\nabla \cdot (\mathbf{A}^{-1}\mathbf{H}) = \nabla \cdot (\mathbf{A}^{-1}\nabla p)$
5. Solve the continuity equation for p , and update the value for \mathbf{u} with the expression $\mathbf{u} = \mathbf{A}^{-1}\mathbf{H} - \mathbf{A}^{-1}\nabla p$
6. Go back to step 2 if the algorithm has not converged

The user can choose the value of the viscosity ν and the source term, the discretization schemes used by the solver to compute the derivatives, the tolerance value for the residuals below which the algorithm considers there is convergence and stops the loop, the boundary conditions (cyclic, Neumann-type, Dirichlet-type, etc.), the turbulence model (it was deactivated in our case)... Cyclic conditions were enforced on the external boundaries for both pressure and velocity, and the conditions $\mathbf{u} = \mathbf{0}$ and $\nabla p = \mathbf{0}$ were imposed at the interface between the two phases. Since p is defined up to an additive constant, the user must select a reference cell, that will be set to a reference value (also chosen by the user).

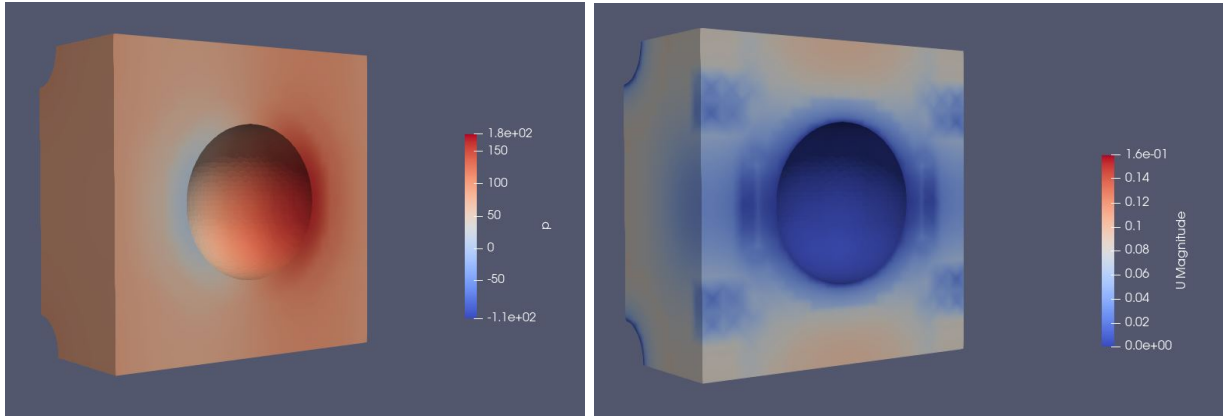
The results obtained with the source term (1 0 0) are shown below :

Figure 5: Fields obtained with simpleFoam, with $r_1 = 0.25$, and $r_2 \in \{0, 0.125\}$



Pseudo-pressure field

Magnitude of pseudo-velocity field



3.2 NGS-Py Finite Element Tool

Presentation

Netgen [4] is an automatic 2D and 3D tetrahedral mesh generator. NGSolve [3, 5] is a finite element library which can be linked to Netgen and contains arbitrary order finite elements of all standard element geometries, scalar, vector-valued, hybrid DG finite element spaces. Program flow as well as geometry description and equation setup can be controlled from Python. Together they offer a rich Python interface inspired by the FEniCS project.

Installation

You first need to create a conda environment for NGSolve:

```
conda create --name ngsolveprojets
```

Then you need to activate it: `conda activate ngsolveprojets`.

Then, instructions can be found on the NGSolve website:

- Add conda-forge and ngsolve channels:
`conda config --add channels conda-forge`
`conda config --add channels ngsolve`
- Install NGSolve: `conda install ngsolve`

The anaconda environment must be activated every time NGSolve will be used:

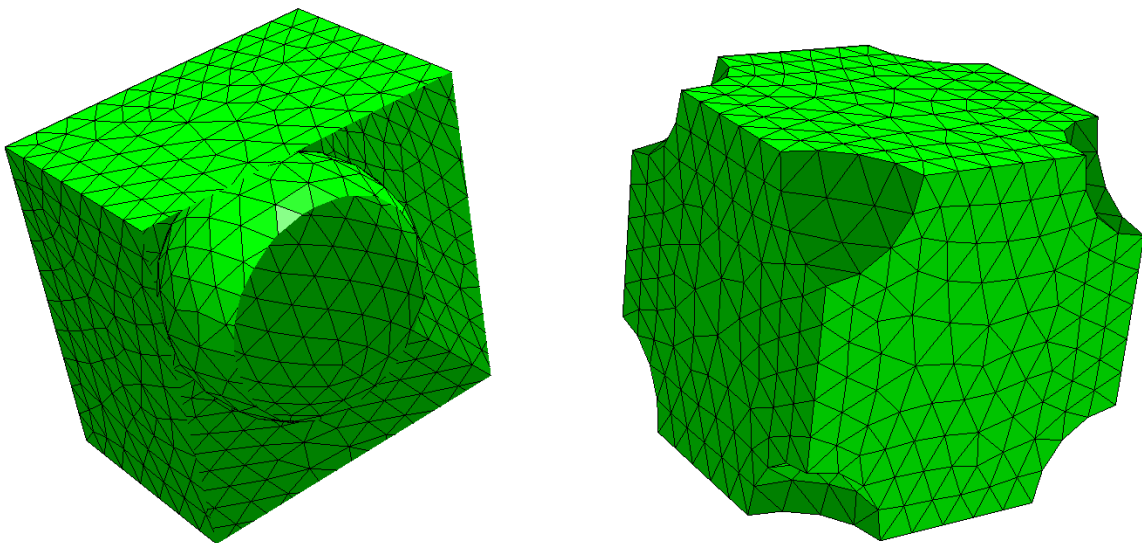
```
conda activate ngsolveprojets
```

Once the environment loaded, a simple `python3 program.py` runs the python program with NGSolve.

Mesh

Netgen allows for precise control of the meshing process while still providing automatic options. The geometry can be constructed either using a point-edge-shape system alike GMSH either using shapes primitives and Boolean operators.

Figure 6: Mesh created and visualized using Netgen for the two cell geometry



Algorithm 1: How to create a mesh in 2D using Netgen

1. Define the geometry object : $geo = SplineGeometry()$
 2. Define the vertex of the geometry : $geo.AppendPoint(x,y) \rightarrow PointObject$
 3. Define the edges of the geometry :
 $geo.Append(["line", PointObjA, PointObjB], leftdomain=(int), rightdomain=(int), bc="name") \rightarrow EdgeObject$
Or for curved edges
/* Some primitives also exist in 2D for rectangles and circles. */
:
 $geo.Append(["spline3", PointObjA, PointObjB, PointObjC], leftdomain=(int), rightdomain=(int), bc="name") \rightarrow EdgeObject$
 - For periodic conditions, just add $copy=OtherEdge$ when defining the second edge.
 - Some good practice :
 - Renaming the domains with str-type name with : $geo.SetMaterial(DomainId, name)$
 - Setting a max mesh size for each domain with :
 $geo.SetDomainMaxH(DomainId, size)$
 4. Generate the mesh :
 $geo.GenerateMesh(maxh=defaultSize) \rightarrow NetgenMeshObject$
 5. Convert it to an NGSolve mesh :
 $geo.GenerateMesh(maxh=defaultSize) \rightarrow NGSolveMeshObject$
-

Algorithm 2: How to create a mesh in 3D using Netgen (using CSG)

1. Create the geometry object : $geo = CSGGeometry()$
 2. Add primitives : Sphere, Cube, Plane, etc.
 3. Define the final geometry with boolean operation : Union (+), Intersection (*), etc.
 4. Generate the mesh : $geo.GenerateMesh(maxh=defaultSize) \rightarrow NetgenMeshObject$
 5. Convert it to an NGSolve mesh :
 $geo.GenerateMesh(maxh=defaultSize) \rightarrow NGSolveMeshObject$
-

Solvers

In NGSolve, no physics solver is provided. The type of spaces (H1, L2, P1, etc), the weak form of the physics equations have to be defined. For linear problems on standard space, only matrix inversion is needed, though tools like Newton solver for non-linear equations are provided. Below is described the procedure for setting up a general problem :

$$\text{Find } u \text{ that satisfies } K(u, v) = f(v), \quad \forall v \in I_{\text{trial functions}} \quad (9)$$

Algorithm 3: How to solve an equation using NGSolve

1. Setup the finite element space :

$H1(mesh, order=(int), dirichlet="boundName1|boundName2") \rightarrow space$

Other type exist like P1, L2, etc.

FE-spaces can have spacial periodicity with $Periodic(Space) \rightarrow PeriodicSpace$

Product space are constructed using the (*) operator.

2. Extract the test and trial function : $u = FESpace.TrialFunction()$

3. Create the bilinear form object and add the left part of (9) :

$K = BilinearForm(FESpace, symmetric=True)$

Note : grad, rot, InnerProduct, OuterProduct, dx(integrator), etc are already defined accordingly to the spaces used

4. Do the same for the linear form and add the source term using the command :

$f = LinearForm(FESpace)$

5. Assemble the form (load them into RAM) using the $.Assemble()$ method

6. Prepare the space for the solution using $gfu = GridFunction(FESpace)$

And (optionally display it with Draw(gfu))

7. Solve the problem with

$gfu.vec.data = K.mat.Inverse(FESpace.FreeDofs()) * f.vec$

8. Do some postprocessing with $Integrate(gfu, mesh)$ for example
-

Figure 7: Magnitude of the pseudo-velocity and pseudo-pressure field with $r_1 = 0.25$, $r_2 = 0$ for the permeability closure problem for the x-component

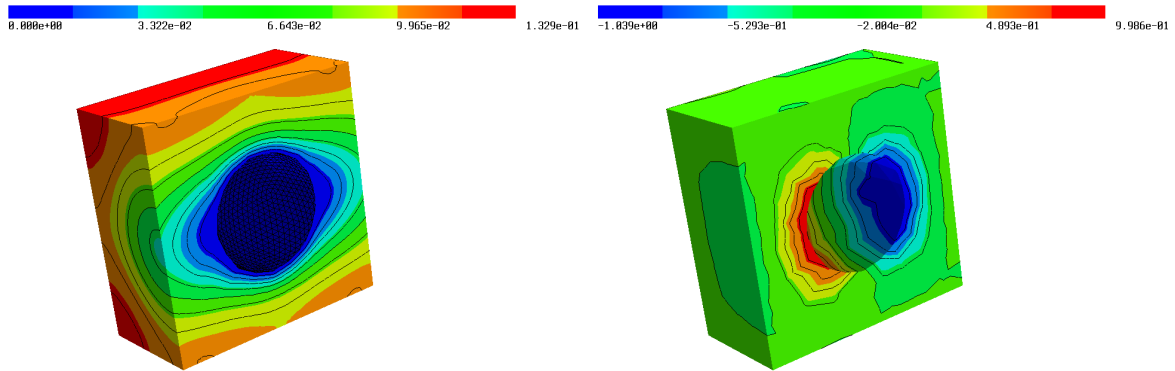


Figure 8: Magnitude of the pseudo-velocity and pseudo-pressure field with $r_1 = 0.25$, $r_2 = 0.125$ for the permeability closure problem for the x-component

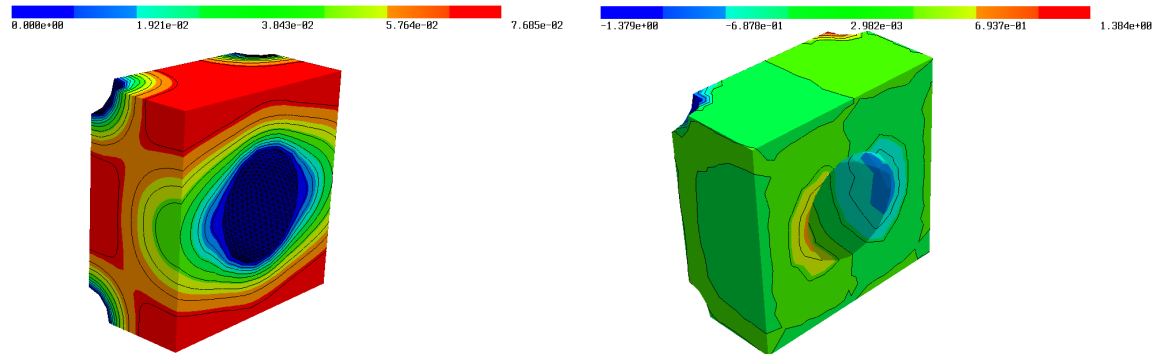
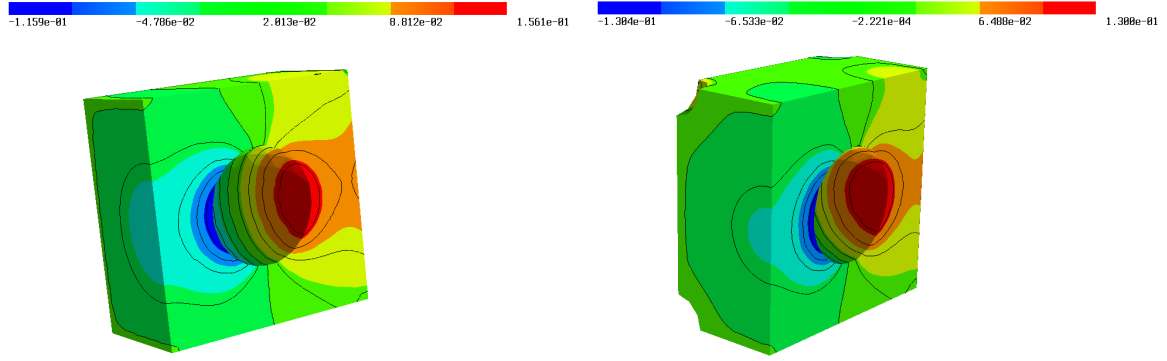


Figure 9: Pseudo-potential fields with $r_1 = 0.25$, $r_2 = 0$ and with $r_1 = 0.25$, $r_2 = 0$ for the diffusive closure problem for the x-component



3.3 FEniCS project

Presentation

FEniCS is a popular open-source computing platform for solving partial differential equations (PDEs). It enables users to quickly translate scientific models into efficient finite element code. With the high-level Python and C++ interfaces to FEniCS, it is easy to get started, but FEniCS offers also powerful capabilities for more experienced programmers. This software runs on a multitude of platforms, ranging from laptops to high-performance clusters.^h

Installation

Instructions can be found on the FEniCS project website:

- Download the anaconda environment:
`conda create -n fenicsproject -c conda-forge fenics`
- Activate the anaconda environment: `source activate fenicsproject`
- Install matplotlib: `conda install matplotlib`

The anaconda environment must be activated every time FEniCS will be used:

`source activate fenicsproject`

Once the environment loaded, a simple `python3 program.py` runs the python program with FEniCS.

Mesh

FEniCS has its own best built-in. It works very well on simple geometries, but has trouble adapting when they become complex. This software program also allows to manually build a mesh.

Both of these methods are not used here. Not efficient for the first one, complicated for the second. It is more relevant to use GMSH to settle this meshing task. This open source 3D finite element mesh generator offers, like Netgen, 2 modes of operation : choose to define the geometry by points, edges and surfaces or by primitives to get curved edges.

Solvers

DOLFIN is how FEniCS' solver is named. As NGsolve it is an adaptive finite element solver for PDEs, but there are two differences between them. DOLFIN mainly uses a functional approach in its coding, while NGsolve uses Object-Oriented Programming. The most important difference in our application is that DOLFIN uses Taylor-Hood's elements, whereas NGsolve uses $H^1 \times H^1 \times H^1 \times L^2$ elements.

Algorithm 4: How to solve an equation using DOLFIN

1. Setup the finite element space
 2. Extract the test and trial function
 3. Define bilinear and linear form of the weak equations
 4. Assemble the form
 5. Prepare the space for the solution
 6. Solve problem
-

4 Results

4.1 Mésocentre

The EM2C Mésocentre is used because it is a reliable way to compute and compare programs. Moreover, it will be used to run those programs as it allows running long scripts on a distant computer, with high memory capacity compared to a standard desktop PC or laptop.

Mésocentre hardware:

Nb of nodes	192
Nb of CPU	2
CPU reference	Intel Xeon Gold 6230 20C @ 2.1 GHz
CPU gen	Cascade Lake
Max memory	192 GB
Max allocable memory	180 GB

Nb of nodes	14
Nb of CPU	4
CPU reference	Intel Xeon Gold 6230 20C @ 2.1 GHz
CPU gen	Cascade Lake
Max memory	1.5 TB
Max allocable memory	1540 GB
SSD disk space (tmp)	870 GB

4.2 Comparison

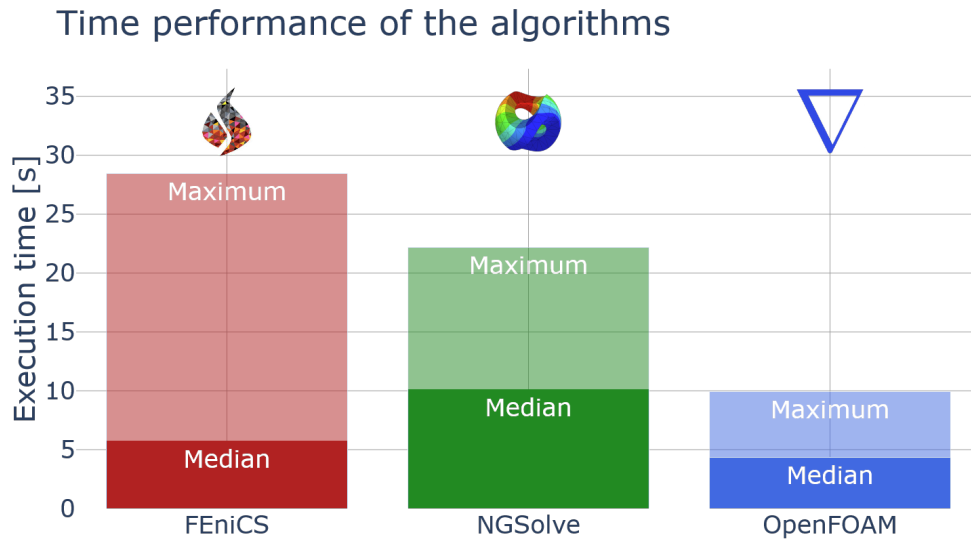
perméabilité (K_{xx}, \dots)

Duration of each program

OpenFOAM is the quickest to run the study, with a relative consistency. A few iterations are about 2 times longer, and correspond to the maximum time. FEniCS is the second quickest. It is really consistent, but the first execution takes a very long time, about 5 times longer. NGSolve is the slowest, but is relatively consistent. The first iteration is also a lot longer, but less than FEniCS' first one.

Convergence en mailles
Ergonomie, Flexibilité

Figure 10: Time execution of each piece of software



5 Conclusion

5.1 Which one should you choose?

Dans quelles mesures prendre l'un ou l'autre

References

- [1] Front matter. In F.A.L. DULLIEN, editor, *Porous Media (Second Edition)*, page iii. Academic Press, San Diego, second edition edition, 1992.
- [2] Morgan Chabanon, Bertrand David, and Benoît Goyeau. Averaged model for momentum and dispersion in hierarchical porous media. *Phys. Rev. E*, 92:023201, Aug 2015.
- [3] Peter Gangl, Kevin Sturm, Michael Neunteufel, and Joachim Schöberl. Fully and semi-automated shape differentiation in ngsolve, 2020.
- [4] Joachim Schöberl. Netgen an advancing front 2d/3d-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1(1):41–52, Jul 1997.
- [5] Joachim Schoeberl. C++11 implementation of finite elements in ngsolve. 09 2014.
- [6] V. V. Tuchin. Polarized light interaction with tissues. *Journal of biomedical optics*, vol. 21, no. 7, 2016.
- [7] Stephen Whitaker. *The Method of Volume Averaging*. 01 1999.