Introduction to fluid mechanics simulation using the OpenFOAM® technology

« Simulation in porous media from pore to large scale »

Part III: First programs, development of a heat transfer solver in porous media

Contact: cyprien.soulaine@stanford.edu
Previously in OpenFOAM training...

So far, we have used OpenFOAM as a black box.

We have used different programs:
- Solvers: laplacianFoam, icoFoam, simpleFoam, interFoam, scalarTransportFoam
- Gridding: blockMesh, snappyHexMesh
- Utilities: checkMesh, surfaceCheck, patchAverage

Actually, one of the strength of OpenFOAM is its simplicity to program partial differential equations:

\[
\frac{\partial \rho U}{\partial t} + \nabla \cdot \phi U - \nabla \cdot \mu \nabla U = -\nabla p
\]
General introduction to the OpenFOAM® technology

- What is OpenFOAM®?
- Where one can find help and documentation?

First simulations with OpenFOAM®

- General structure of an OpenFOAM® case
- #1 – Heat diffusion
- #2 – Cavity
- #3 – Poiseuille flow
- #4 – Drainage experiment in a capillary tube

How to mesh complex geometry with OpenFOAM®?

- snappyHexMesh overview
- #5 – Mesh a pore-space with snappyHexMesh
- #6 – Scalar transport in porous media at the pore-scale

Programming equations with OpenFOAM®

- General structure of an application
- Basics of OpenFOAM programming

Heat transfer in porous media with OpenFOAM®

- #7 – Create a « Darcy » solver
- #8 – Temperature in porous media
- #9 – Two-equations model for heat transfer in porous media

How to solve Navier-Stokes equation with OpenFOAM®?
Where is the source code of a solver?

- OpenFOAM can be seen as an easily customisable toolbox.

- 1 solver = 1 program
  (for instance, the heat equation is solved using the program `laplacianFoam`)

- Where are the solvers in OpenFOAM?
  
  ```
  $ cd $FOAM_APP/solvers
  $ ls
  ```

- The solvers are organized by type (basic, heat transfer, combustion, incompressible, multiphase….). Note that the tutorials have a similar organisation.

- For example, `laplacianFoam` is in `/basic`
  
  ```
  $ cd basic/laplacianFoam
  $ ls
  $ gedit laplacianFoam.C
  ```
Behind laplacianFoam: *laplacianFoam.C*

```c
#include "fvCFD.H"
#include "simpleControl.H"

// ******************************************************************************

int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createFields.H"

    simpleControl simple(mesh);

    // ******************************************************************************

    Info<< "Calculating temperature distribution\n" << endl;

    while (simple.loop())
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

        while (simple.correctNonOrthogonal())
        {
            solve
            {
                fvm::ddt(T) - fvm::laplacian(DT, T)
            }
        }

        #include "write.H"

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << " ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}
```

Call the OpenFOAM libraries

Creation of the matrix

\[
\frac{\partial T}{\partial t} = \nabla \cdot (D_T \nabla T)
\]

fvm:: implicit terms

fvc:: explicit terms

the variable T and DT are declared in *createField.H*
The temperature field $T$ is declared as an instance of the object `volScalarField`

- It is a scalar field with values defined at the cell center
- It must be read at the initial time step
- Dimensions (units) are defined in $0/T$
- $T$ will be written at each time step in the corresponding folder (runTime.timeName())
- This object also includes boundary conditions that are specified in $0/T$

The dictionary `transportProperties` is loaded from the input file `constant/transportProperties`

Declaration of the variable DT

Its value and dimensions are defined in the input file `constant/transportProperties`
General structure of an application

Source code
- The file with the .C extension is the main file

List of files that will be compiled
- Link towards the OpenFOAM libraries used in this program

Example of the *icoFoam* solver

```
$ cd $FOAM_APP/solvers/incompressible/icoFoam
$ ls
```

Create the directory for your personal programs (this stage only needs to be done once)

```
$ mkdir -p $WM_PROJECT_USER_DIR/applications/solvers/
```
Objective: develop a program that solves the flow in a fully saturated porous medium using Darcy's law.

\[ \nabla \cdot U = 0 \quad (1) \]

\[ U = -\frac{k}{\mu} \nabla p \quad (2) \]

How to solve this mathematical problem? The diffusion equation for the pressure field is obtained by combining equation (1) and (2):

\[ \nabla \cdot \left( \frac{k}{\mu} \nabla p \right) = 0 \]

We are going to program our own solver on the basis of the existing \textit{laplacianFoam}\n
$ cd$ \ $WM\_PROJECT\_USER\_DIR/applications/solvers/
$ cp$ \ $-r$ \ $FOAM\_APP/solvers/basic/laplacianFoam darcyFoam$
Once the *laplacianFoam* solver has been copied into the user directory, we rename the main file and edit the Make/files:

```bash
$ cd darcyFoam
$ mv laplacianFoam.C darcyFoam.C
$ gedit Make/files
```

List of the files to compile (only the .C files)

Add _USER_ to specify that the new executable will be compiled into your user directory (without _USER_ your compilation will fail!)

Name of the new program

We can now clean the previous compilation with *wclean* and compile this new program with *wmake*.

```bash
$ wclean
$ wmake
```

At this stage, we have a new program called *darcyFoam* which is exactly a copy of *laplacianFoam* (you can run it on the flange tutorial or Exo1).

It is recommended to use *wmake* as often as possible during the programming stage.
Declaration of the pressure field $p$:
- It will be written at each output time step folders.
- It must be read at the first time step.

Declaration of the velocity vector field $U$:
- It will be written in every time step folder.
- It will not be read from any file (even if 0/$U$ exists).
- To satisfy all the constructors required by the object volVectorField, the dimensions and the initial value of $U$ are specified by an additional argument.

Declaration of the fluid viscosity $\mu$ and the permeability $k$:
They will be loaded from « constant/transportProperties ».
#7 – Program a “Darcy” solver (3/6)

The pressure field $p$ is solved by a diffusion equation:

$$ \text{solve} $$

$$ \text{fvm::laplacian}(k/mu, p) $$

The velocity vector $U$ is deduced from the pressure field using the Darcy's law:

$$ U = -k/mu \cdot \text{fvc::grad}(p) $$

The useless files are removed and the `darcyFoam` executable is then compiled.

$ \text{rm write.H} $
$ \text{wclean} $  
$ \text{wmake} $
#7 – Program a “Darcy” solver (4/6)

To prepare this « case » we are going to mimic the tutorial `laplacianFoam/flange`, since its setup is quite similar to our example.

```bash
$ run
$ cp -r $FOAM_TUTORIALS/basic/laplacianFoam/flange Exo7
$ cd Exo7
$ rm Allrun Allclean flange.ans
```

We are going to simulate flow in an 1D porous medium:

To save time, we can pick up and modify an existing `blockMeshDict`.

```bash
$ cp $FOAM_TUTORIALS/incompressible/icoFoam/cavity/constant/polyMesh/blockMeshDict constant/polyMesh/.
$ gedit constant/polyMesh/blockMeshDict
```

The grid is generated using `blockMesh`.

```bash
$ blockMesh
```
$ mv 0/T 0/p
$ gedit 0/p

A pressure differential is imposed between the inlet and the outlet of the computational domain.
#7 – Program a “Darcy” solver (5b/6)

$ gedit constant/transportProperties

\[
\begin{align*}
\mu & \quad \mu \quad \begin{bmatrix} 1 & -1 & -1 & 0 & 0 & 0 \end{bmatrix} \quad 1e-05; \\
k & \quad k \quad \begin{bmatrix} 0 & 2 & 0 & 0 & 0 & 0 \end{bmatrix} \quad 1e-09;
\end{align*}
\]

$ gedit system/fvSolution

```bash
solvers
{
  p
  {
    solver PCG;
    preconditioner DIC;
    tolerance 1e-06;
    relTol 0;
  }
}
SIMPLE
{
  nNonOrthogonalCorrectors 2;
}
```

```bash
$ gedit system/fvSchemes
```

```bash
ddtSchemes
{
  default steadyState;
}

gradSchemes
{
  default Gauss linear;
}

divSchemes
{
  default none;
}

laplacianSchemes
{
  default none;
  laplacian((k|\mu),p) Gauss linear corrected;
}

interpolationSchemes
{
  default linear;
}

snGradSchemes
{
  default corrected;
}

fluxRequired
{
  default no;
}
```
Since `darcyFoam` is a steady-state solver, only one time step is necessary.

```bash
$ gedit system/controlDict

application     darcyFoam;
startFrom        latestTime;
startTime        0;
stopAt           endTime;
endTime          1;
deltaT           1;
writeControl     runTime;
writeInterval    1;
purgeWrite       0;
writeFormat      ascii;
writePrecision   6;
writeCompression off;
timeFormat       general;
timePrecision    6;
runTimeModifiable true;
```
#7 – Program a “Darcy” solver (6/6)

Run the simulation:

```
$darcyFoam
```

Results will be plotted using the `sample` utility and the program Gnuplot. As `blockMesh`, the program `sample` requires an input dictionary located in `/system`:

```
$ cp $FOAM_UTILITIES/postProcessing/sampling/sample/sampleDict system/
$ gedit system/sampleDict
```

Plot the pressure field with Gnuplot:

```
$ gnuplot

gnuplot> set xlabel "distance (m)"
gnuplot> set ylabel "Pressure (kg/m/s)"
gnuplot> plot "postProcessing/sets/1/lineX1_p.xy" using 1:2 with lines lw 4 title "p"
```

Run the `sample` tool:

```
$ sample
```

**Exo7Bis:** Program a solver for heterogeneous porous media defining the permeability as a `volScalarField` and assigning it different values with the utility `setFields`. 
Objective 1: Develop a program that solves heat transfer in a porous medium

\[ \nabla \cdot \mathbf{U} = 0 \]

\[ \mathbf{U} = -\frac{k}{\mu} \nabla p \]

\[ \left( \varepsilon (\rho C_p)_f + (1 - \varepsilon) (\rho C_p)_s \right) \frac{\partial T}{\partial t} + (\rho C_p)_f \nabla \cdot (\mathbf{U} T) = \nabla \cdot (D_T \nabla T) \]

Objective 2: Use probes to plot the temperature evolution vs time at some points of the domain

Objective 3: Change the discretization schemes

This solver will be based on \textit{darcyFoam}:

\$ cd \$WM_PROJECT_USER_DIR/applications/solvers/
\$ cp –r darcyFoam darcyTemperatureFoam
\$ cd darcyTemperatureFoam
\$ mv darcyFoam.C darcyTemperatureFoam.C
\$ gedit Make/files

\$ wclean
\$ wmake
#8 – Heat transfer in porous media (2a/7)

```c
Info<< "Reading field p\n" << endl;
volScalarField p
(
  IOobject
  {
    "p",
    runTime.timeName(),
    mesh,
    IOobject::MUST_READ,
    IOobject::AUTO_WRITE
  ),
  mesh
);
Info<< "Reading field U\n" << endl;
volVectorField U
(
  IOobject
  {
    "U",
    runTime.timeName(),
    mesh,
    IOobject::NO_READ,
    IOobject::AUTO_WRITE
  ),
  mesh,
  dimensionedVector("U", dimensionSet(0,1,-1,0,0,0,0), vector::zero)
);

surfaceScalarField phi ("phi", linearInterpolate(U) & mesh.Sf());
Info<< "Reading field T\n" << endl;
volScalarField T
(
  IOobject
  {
    "T",
    runTime.timeName(),
    mesh,
    IOobject::MUST_READ,
    IOobject::AUTO_WRITE
  ),
  mesh
);
```

Declaration of the velocity flux \( \phi \).
- It is a surface field (\( U \) is projected onto the face of each cell of the grid)
- It is necessary to use the divergence operator (fvm::div(\( \phi, T \))
- Can also be declared using \#include “createPhi.H”

Declaration of the temperature field \( T \).

$ gedit createFields.H$

$ wmake$
Beside the viscosity \( \mu \) and the permeability \( k \) of the porous medium, we also declare the thermal diffusivity \( DT \), the porosity \( \varepsilon \) and the heat capacities \( \rho Cps \) and \( \rho Cpf \). They will be loaded from the file « constant/transportProperties ».
The surface flux $\phi$ is updated from the new value of the velocity profile $U$.

Solve the advection/diffusion equation for the temperature.

Compilation of `darcyTemperatureFoam`

```cpp
while (simple.loop())
{
   Info<< "Time = " << runTime.timeName() << nl << endl;

   while (simple.correctNonOrthogonal())
   {
      solve
      {
         fvm::laplacian(k/mu, p);
      }

      U = -k/mu*fvc::grad(p);

      phi = linearInterpolate(U) & mesh.Sf();

      solve
      {
         (eps*rhoCpf+(1.-eps)*rhoCps)*fvm::ddt(T)
         + rhoCpf*fvm::div(phi,T)
         == fvm::laplacian(DT, T);
      }
   }

   runTime.write();

   Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
      << " ClockTime = " << runTime.elapsedClockTime() << " s" << nl << endl;
}
```
We want to estimate the temperature evolution in a 1D porous medium.

To save time, we can adapt the previous exercise to setup the case.

```
$ run
$ cp -r ../Exo7 Exo8
$ cd Exo8
$ rm -r 0.* 1* 2* 3* 4* 5* 6* 7* 8* 9* postProcessing
$ cp 0/p 0/T
$ gedit 0/T
```
#8 – Heat transfer in porous media (5a/7)

$ gedit 0/T$

```plaintext
dimensions [0 0 0 1 0 0 0];
internalField uniform 273;
boundaryField {
    inlet {
        type fixedValue;
        value uniform 573;
    }
    outlet {
        type zeroGradient;
    }
    frontAndBack {
        type empty;
    }
}
```

$ gedit 0/p$

```plaintext
dimensions [1 -1 -2 0 0 0 0];
internalField uniform 0;
boundaryField {
    inlet {
        type fixedValue;
        value uniform 1e2;
    }
    outlet {
        type fixedValue;
        value uniform 0;
    }
    frontAndBack {
        type empty;
    }
}
```
#8 – Heat transfer in porous media (5b/7)

```plaintext
$ gedit constant/transportProperties

mu     mu       [ 1 -1 -1 0 0 0 0 ] 1e-05;
k      k        [ 0 2 0 0 0 0 0 ] 1e-09;
eps    eps      [ 0 0 0 0 0 0 0 ] 0.4;
DT     DT       [ 1 1 -3 -1 0 0 0 ] 1e-02;
rhoCps rhoCps   [ 1 -1 -2 -1 0 0 0 ] 2e4;
rhoCpf rhoCpf   [ 1 -1 -2 -1 0 0 0 ] 5e3;

$ gedit system/fvSchemes

ddtSchemes
[]
  default Euler;
}
gradSchemes
{
  default Gauss linear;
  grad(T) Gauss linear;
}
divSchemes
{
  default none;
  div(phi,T) Gauss linear;
}
laplacianSchemes
{
  default none;
  laplacian((k|mu),p) Gauss linear corrected;
  laplacian(DT,T) Gauss linear corrected;
}
interpolationSchemes
{
  default linear;
}

snGradSchemes
{
  default corrected;
}

fluxRequired
{
  default no;
  T yes;
}

SIMPLE
{
  nNonOrthogonalCorrectors 2;
}
```

cyprien.soulaine@stanford.edu
The probes are functions that are executed on-the-fly during the simulation. They allow to record the temperature evolution vs time. You can specify as many probes as you want.

```plaintext
#8 – Heat transfer in porous media (5c/7)

startFrom latestTime;
startTime 0;
stopAt endTime;
endTime 60000;
deltaT 100;
writeControl runTime;
writeInterval 1000;
purgeWrite 0;
writeFormat ascii;
writePrecision 6;
writeCompression off;
timeFormat general;
timePrecision 6;
runTimeModifiable true;
functions {
    probes {
        type probes;
        functionObjectLibs ("libsampling.so");
        enabled true;
        outputControl timeStep;
        outputInterval 1;
        fields {
            T;
        }
        probeLocations {
            ( 2 0.05 0.85) // Probe 1
            ( 5 0.05 0.85) // Probe 2
            ( 9 0.05 0.85) // Probe 3
        }
    }
}
```
Run the simulation:  
```bash
$ darcyTemperatureFoam
```

We are going to plot the probe results with the following gnuplot script:
```bash
$ gedit plot_probes
```

```bash
set key at 50000, 400
set xlabel "temperature (K)"
set xlabel "time (s)"

plot "postProcessing/probes/0/T" using 1:2 with lines lw 4 title "Probe x=2m" ,
"postProcessing/probes/0/T" using 1:3 with lines lw 4 title "Probe x=5m" ,
"postProcessing/probes/0/T" using 1:4 with lines lw 4 title "Probe x=9m"
```

```bash
$ gnuplot --persist plot_probes
```
Note in the previous simulation some unphysical oscillations at the temperature front. They are due to the numerical scheme used to discretize the convection term. To improve the numerical stability, you can use an upwind scheme or a flux limiter by specifying `Gauss upwind` or `Gauss vanLeer` in `system/fvSchemes` instead of `Gauss linear`.

The upwind scheme is better than the linear but also more diffusive. The flux limiter schemes are more suitable for this kind of simulation.

More benchmarks on OpenFOAM numerical schemes:

#9 – Two-temperature model (1/6)

**Objectif n°1 :** Résoudre un transfert de chaleur dans un milieu poreux par un modèle à 2 équations

\[
\nabla \cdot \mathbf{U} = 0
\]

\[
\mathbf{U} = -\frac{k}{\mu} \nabla p
\]

\[
\varepsilon \frac{\partial T_f}{\partial t} + \nabla \cdot \mathbf{U} T_f = \nabla \cdot D T_f \nabla T_f + h_{sf} (T_f - T_s)
\]

\[
(1 - \varepsilon) \frac{\partial T_s}{\partial t} = \nabla \cdot D T_s \nabla T_s - h_{sf} (T_f - T_s)
\]

**Objectif n°2 :** Utiliser des conditions à la limite personnalisées

On va créer notre solveur en se basant sur le solveur `darcyTemperatureFoam`

```bash
$ cd $WM_PROJECT_USER_DIR/applications/solvers/
$ cp –r darcyTemperatureFoam darcyTwoTemperaturesFoam
$ cd darcyTwoTemperaturesFoam
$ mv darcyTemperatureFoam.C darcyTwoTemperaturesFoam.C
$ gedit Make/files
```
Le champ $U$ est maintenant initialisé à partir de $0/U$, ce qui nous permet de définir des conditions d’entrée pour $U$.

Le champ $\phi$ est créé en appelant `createPhi.H`.

Déclaration des champs de température pour le solide et pour le fluide.

Les constantes du modèle seront lues dans le fichier `constant/transportProperties`.
#9 – Two-temperature model (3/6)

```c
// Calculating flow and temperatures in porous media

while (simple.loop())
{
    Info<< "Time = " << runTime.timeName() << nl << endl;
    // Calcul de p
    fvScalarMatrix pEqn
    (    fvm::laplacian(k/mu, p)
    );
    pEqn.solve();
    // Calcul de U et phi
    U = -k/mu*fvc::grad(p);
    phi = linearInterpolate(U) & mesh.Sf();

    // Calcul de Tf
    fvScalarMatrix TfEqn
    (    eps*fvm::ddt(Tf) + fvm::div(phi,Tf) - fvm::laplacian(DTf,Tf) -fvm::Sp(h,Tf) + h*Ts
    );
    TfEqn.solve();

    // Calcul de Ts
    fvScalarMatrix TsEqn
    (    (scalar(1)-eps)*fvm::ddt(Ts) - fvm::laplacian(DTs,Ts) - fvm::Sp(h,Ts) + h*Tf
    );
    TsEqn.solve();

    runTime.write();
    Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s" << " ClockTime = " << runTime.elapsedClockTime() << " s" << nl << endl;
}
Info<< "End\n" << endl;
return 0;
```

Résolution de la température dans le fluide. Une partie du terme d’échange est traitée en implicite, l’autre en explicite.

Résolution de la température dans la matrice poreuse.

```bash
$ gedit darcyTwoTemperaturesFoam.C
$ wclean
$ wmake
```
On cherche à évaluer le transfert de chaleur dans un milieu poreux 1D via un modèle à deux températures.

Dans cet exemple, un milieu poreux initialement à 573K est refroidi par l’injection d’un fluide à 273K.

On va se baser sur l’exercice précédent pour paramétrer le calcul.

```bash
$ run
$ mkdir darcyTwoTemperaturesFoam
$ cp -r ../darcyTemperatureFoam/Exo8 Exo10
$ cd Exo10
$ rm -r 0.* 1* 2* 3* 4* 5* 6* 7* 8* 9* sets probes
$ mv 0/T 0/Tf
$ cp 0/Tf 0/Ts
$ gedit 0/Tf
```
On impose une vitesse du fluide à l’entrée du milieu. Cette vitesse va permettre d’évaluer le gradient de pression à l’entrée.
On utilise ici la condition à la limite `darcyGradPressure` définie dans l’exercice précédent. Le gradient de pression à la limite est évalué via : 

$$ \mathbf{n}.\nabla p = -\frac{\mu}{k}\mathbf{n} \cdot \mathbf{U} $$

On doit préciser dans le fichier `system/controlDict` que l’on utilise la librairie `ldarcyGradPressure.so`. 

---

#9– Two-temperature model (5b/6)
#9 – Two-temperature model (5c/6)

$ gedit 0/Ts$
#9 – Two-temperature model (5d/6)

```yaml
FoamFile
{
    version 2.0;
    format ascii;
    class volScalarField;
    object Tf;
}

// Dimensions: [0 0 0 1 0 0 0];

internalField uniform 573;

boundaryField
{
    inlet
    {
        type fixedValue;
        value uniform 273;
    }

    outlet
    {
        type zeroGradient;
    }

    frontAndBack
    {
        type empty;
    }
}
```

$ gedit 0/Tf
$ gedit constant/transportProperties

---

```
FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    location "constant";
    object transportProperties;
}

// ************************************************************************///

k  k  [ 0 2 0 0 0 0 0 ] 1e-11;
mu  mu  [ 1 -1 -1 0 0 0 0 ] 1e-05;
eps  eps  [ 0 0 0 0 0 0 0 ] 0.4;
DTf  DTf  [ 0 2 -1 0 0 0 0 ] 1e-12;
DTs  DTs  [ 0 2 -1 0 0 0 0 ] 1e-6;
h  h  [ 0 0 -1 0 0 0 0 ] -1e-05;
// ************************************************************************///
```
On précise ici que l'on va utiliser la librairie \textit{ldarcyGradPressure.so} qui nous permettra d'utiliser la condition à la limite personnalisée \textit{darcyGradPressure}.
#9 – Two-temperature model (5g/6)

```plaintext
$ gedit system/fvSolution

FoamFile
{
  version 2.0;
  format ascii;
  class dictionary;
  location "system";
  object fvSolution;
}

// ******************************************************************************

solvers
{
  p
  {
    solver PCG;
    preconditioner DIC;
    tolerance 1e-06;
    relTol 0;
  }

  Tf
  {
    solver PBiCG;
    preconditioner DILU;
    tolerance 1e-06;
    relTol 0;
  }

  Ts
  {
    solver PCG;
    preconditioner DIC;
    tolerance 1e-06;
    relTol 0;
  }
}

SIMPLE
{
  nNonOrthogonalCorrectors 2;
}

$ gedit system/fvSchemes

// ******************************************************************************

ddtSchemes
{
  default Euler;
}

gradSchemes
{
  default Gauss linear;
  grad(p) Gauss linear;
}

divSchemes
{
  default none;
  div(phi,Tf) Gauss linear;
}

laplacianSchemes
{
  default none;
  laplacian(DT|Ts) Gauss linear corrected;
  laplacian(DTf,Tf) Gauss linear corrected;
  laplacian((k|mu),p) Gauss linear corrected;
}

interpolationSchemes
{
  default linear;
}

snGradSchemes
{
  default corrected;
}
```
On lance la simulation : $darcyTwoTemperaturesFoam$

On post-traite ensuite les résultats (ici on a tracé les valeurs de $T_s$ et $T_f$ en fonction du temps pour les 3 sondes)
In the next parts...

Part IV: How to solve Navier-Stokes equations with OpenFOAM?

http://web.stanford.edu/~csoulain/