OpenFOAM: solvers, libraries, patches, utilities, and cases

Vitaliy Starchenko and Anthony J.C. Ladd

June 18, 2016

In this document we provide a brief description of the codes used to generate the data shown in the paper; they were built with OpenFOAM v2.4.0. Earlier versions (OpenFOAM v2.x.x) should also work, but a few features are only supported in v2.4.0. Specifically, the `blockMeshDict` files in the case directories use features from v2.4.0; mesh grading is implemented differently in previous versions of OpenFOAM.

The codes have been organized into the subdirectories outlined below: solvers, libraries, patches, utilities and cases. The most up-to-date version can be found in an open GitHub repository [*Starchenko and Ladd*(2016)]. A snapshot is available for download as Supporting Information.

1 Solvers

We developed separate solvers for two-dimensional and three-dimensional simulations; the separation of time scales between transport and dissolution is utilized in both solvers. On the first time step the steady-state flow and concentration fields are calculated for the initial fracture geometry. In subsequent steps, reactive fluxes from the previous step are used to update the geometry; then the flow and concentration equations are solved in the updated geometry.

1.1 dissolFoam

The solver `dissolFoam` was used for all the 3D simulations; it requires its own dictionary `dissolFoamDict`. It depends on the libraries `steadyStateControl` and `dissolMeshRelax` and may also utilize the boundary conditions provided by `danckwerts` and `nonLinear`. The coordinate system in the code is different from the one in the paper: here x is the transverse direction (across the width of the fracture), y is the direction across the aperture, and z is the flow direction. In this document we use the coordinate system from the code when describing 3D simulations.

1.2 porousFoam

The solver `porousFoam` was used for all the 2D simulations; it requires its own dictionary `porousFoamDict`. It depends on the `steadyStateControl` library and may also utilize the boundary conditions provided by `danckwerts` and `nonLinear`. It uses the same geometry as the paper: x pointing in the flow direction and y transverse to the flow. The correction to the reactive fluxes at the fracture surfaces from diffusion across the aperture (z) is incorporated into the reaction kinetics. The code is a component of a more general Darcy solver, designed

to include user-programmable constitutive models, including multicomponent transport and kinetics.

2 Libraries

The solvers rely on several common libraries which are collected within `libsOpenFOAMauxiliary`.

2.1 dissolMeshRelax

The `dissolMeshRelax` class supports the relaxation of mesh points on the boundaries of the system; it requires its own dictionary `meshRelaxDict`. It is used to create the input to the `pointMotionU` field which serves as boundary conditions for OpenFOAM's internal mesh relaxation. The library is still under development; at present it is limited to geometries with parallel inlet and outlet planes. Moreover, the surface mesh relaxation is not yet coupled with the linear solvers in OpenFOAM and is rather slow. Although we do not yet allow for topological changes, mesh relaxation by itself is effective in extending the time that fracture dissolution can be followed by several orders of magnitude. This class is only used by the 3D solver `dissolFoam`.

2.2 steadyStateControl

The time scale in dissolution simulations is determined by the motion of the fracture surfaces, not the solution of the flow and transport equations. The OpenFOAM class `simpleControl`, which iterates the flow solver, updates the time counter every iteration whereas we would like to update the time only when a fully converged solution is reached. The `steadyStateControl` class is a modification of `simpleControl`, which removes the time update while iterating the `SIMPLE` loop. It requires a patch to the OpenFOAM source code `dataDictPatch` (Sec. 3.1). This class is used by both `dissolFoam` and `porousFoam`.

2.3 nonLinear

The `nonLinearFvPatchField` class implements a nonlinear boundary condition for the reactive flux at a fracture surface. The implementation is derived from the OpenFOAM `mixedFvPatchScalarField` class which implements a Robin boundary condition

$$c_b = f c_{ref} + (1 - f)(c + \delta g_{ref}) \tag{1}$$

with constant `valueFraction` (f), `refValue` (c_{ref}), and `refGrad` g_{ref}. Here c_b is the concentration on the boundary face, c is the concentration at the cell center, and δ is the distance from the cell center to the boundary [*Jasak*(1996)]. The `nonLinearFvPatchField` class adds support for the update of `refValue`, `refGrad`, and `valueFraction` according to the rate law in Eq. 36 of the paper:

$$c_{ref} = c_{old}, \qquad g_{ref} = \frac{-\delta R(c_{ref})}{l_R}, \qquad f = \frac{\delta R'(c_{ref})}{l_R + \delta R'(c_{ref})}, \tag{2}$$

where c_{old} is the value of c obtained on the previous outer iteration. The boundary condition is incorporated into the solution for the concentration field using functions from the `mixedFvPatchScalarField` class. Other functional forms for the reaction rate could be programmed by suitable modifications of the `updateCoeffs()` function in `nonLinearFvPatchField.C`.

2.4 danckwertsFvPatchScalarField

The `danckwertsFvPatchScalarField` class implements a Danckwerts boundary condition (Eq. 43 of the paper), which is frequently used as an inlet condition on the concentration field. The class is also derived from `mixedFvPatchScalarField`, adding support for the update of `refValue`, `refGrad`, and `valueFraction`:

$$c_{ref} = 1, \quad g_{ref} = 0 \quad f = \frac{\delta u_{in}}{D + \delta u_{in}}, \tag{3}$$

where $u_{in} = -\boldsymbol{n} \cdot \boldsymbol{u}$. In this case the boundary condition is linear in the concentration, and no iteration is needed. The Danckwerts condition requires the velocity field `U` and the diffusion `D`, which can be a tensor field, a scalar field, or a constant.

2.5 OFstreamMod

The `OFstreamMod` class is a modification of `OFstream` adding support for appending to an existing file; it is used by the post-processing tool `dissolPostProc` (Sec. 4.1). The code for `OFstreamMod` follows an online posting by Hrvoje Jasak [*Jasak*(2006)].

3 Patches

In some instances additional functionality requires a modification or addition to the OpenFOAM source code. Scripts are provided to apply the patches and also to restore the original source code. Owing to substantial changes introduced with version 3.0+ these patches will only work with v2.x.x. A different set of patches are available for v3.0+.

3.1 dataDictPatch

Adds a direct clear of the `SolverPerfomance` dictionary, which is required by the `steadyStateControl` class (Sec. 2.2).

3.2 vectorNormPatch

In OpenFOAM the residuals of vector and tensor fields are normalized component by component,

$$R_\alpha = \frac{\|\boldsymbol{r}_\alpha\|}{\|N_\alpha\|} \tag{4}$$

where $\boldsymbol{r}_\alpha$ is the α component of the residual and N_α is the corresponding component of the vector (or tensor) used to construct the norm. For vector or tensor fields, the segregated solver iterates over all components calculating the normalized residual separately. Finally, the largest value

$$R_{max} = \max_{1 \le \alpha \le n} R_\alpha, \tag{5}$$

is compared with the convergence criterion. However, R_{max} is not independent of the choice of coordinate system, which can lead to problems if one of the components is small or vanishing; for example a purely two-dimensional flow field. Here, both the residual and the norm are determined by round-off error and R_{max} remains of order 1 indefinitely; the convergence criterion is never satisfied.

One solution is to normalize all the residuals by a single norm summed over all the components of the field,

$$R_\alpha = \frac{\|\boldsymbol{r}_\alpha\|}{\|N\|} \tag{6}$$

where

$$\|N\| = \sum_{1 \le \alpha \le n} \|N_\alpha\|. \tag{7}$$

Unfortunately this requires returning both $\|\boldsymbol{r}_\alpha\|$ and $\|\boldsymbol{N}_\alpha\|$, whereas OpenFOAM's matrix classes only returns a single variable (the ratio). We have solved the problem by making an additional call from `fvMatrixSolve.C` to obtain the normalizing factors $\|\boldsymbol{N}_\alpha\|$. In order to implement Eqs. (6) and (7) three files need to be patched and the source code and dependencies recompiled:

- `fvMatrixSolve.C` located at `finiteVolume/fvMatrices/fvMatrix/`
- `lduMatrix.H` located at `OpenFOAM/matrices/lduMatrix/lduMatrix/`
- `lduMatrixSolver.C` located at `OpenFOAM/matrices/lduMatrix/lduMatrix/`

It is not practical to make alternative libraries to implement these functions because they have a large number of daughter classes that would all need to be modified.

4 Utilities

There are several utilities that have been developed for pre and post processing. Some of them are OpenFOAM solvers, while others are Python scripts.

4.1 dissolPostProc

A postprocessing tool based on the OpenFOAM utility foamCalc. It integrates OpenFOAM field data over the fracture aperture to create two-dimensional average fields for the comparisons in Figs. 8 and 9 of the paper. It can be executed from the case directory: for example

```
dissolPostProc fieldMap2D all 1000 100 16
```

will calculate all the 2D fields (aperture h, average concentration c, and fluxes q_x, q_y) using 1000 cells in x (flow) direction, 100 cells in y (lateral) direction and 16 cells across the aperture.

4.2 surfRoughGen

A preprocessing tool to generate rough surfaces by spectral synthesis. It is controlled by a dictionary `surfRoughGenDict`.

4.3 genBlockMeshDict

A python tool to generate dictionaries for the `blockMesh` utility. The `blockMeshDict` files it creates are only supported by v2.4.0 and later.

5 Cases

We have provided a sample set of case files that can be used to reproduce some of the simulation results reported in the text. Related cases can be generated by small modifications of the input files.

5.1 case1

DissolFoam case file for 3D resolution tests (Figs. 6 and 7). The mesh resolution is $100 \times 8 \times 267$ (transverse, cross-aperture, and flow directions) and the case takes about 1 hour to run on 8 cores.

5.2 case2

DissolFoam case file of a single seed, for comparison with the 2D code (Fig. 8). The mesh resolution is $800 \times 8 \times 267$ and the case takes about 3 days on 32 cores.

5.3 case2a

PorousFoam case file for comparison with the 3D code (Fig. 8). The mesh resolution is 4000×400 and the case takes about 20 minutes on 8 cores.

5.4 case3

DissolFoam case file of two seeds for comparison with the 2D code (Fig. 9). The mesh resolution is $1600 \times 8 \times 267$ and the case takes about 7 days on 32 cores.

5.5 case3a

PorousFoam case file for two seed comparison with 3D code (Fig. 9). The mesh resolution is 4000×800 and the case takes about 30 minutes on 16 cores.

5.6 case4

DissolFoam case file for a laboratory-scale fracture (Fig. 10). The mesh resolution is $400 \times 8 \times 411$ and the case takes about 1 day on 16 cores.

5.7 case5

DissolFoam case file for a laboratory-scale fracture with a random aperture distribution (Fig. 13). The mesh resolution is $512 \times 8 \times 1024$ and the case takes about 1 day on 32 cores.

5.8 case6

DissolFoam case file for a field-scale fracture (Fig. 16). The mesh resolution is $512 \times 8 \times 1024$ and the case takes about 10 hours on 32 cores.

References

[*Jasak*(1996)] Jasak, H. (1996), Error analysis and estimation for the finite volume method with applications to fluid flows, Ph.D. thesis, University of London.

[*Jasak*(2006)] Jasak, H. (2006), How to open an old file for append, http://www.cfd-online.com/Forums/openfoam-solving/58347-how-open-old-file-append.html.

[*Starchenko and Ladd*(2016)] Starchenko, V., and A. J. C. Ladd (2016), A collection of solvers, libraries and unofficial patches for the OpenFOAM project 2.x.x, https://github.com/vitst.