Reactive transport with moving boundaries using OpenFOAM

Vitaliy Starchenko and Tony Ladd

March 11, 2018

Abstract In this document we give a description of the source codes and sample case files that can be used to solve moving boundary problems with OpenFOAM. In this case surface motion is driven by the gradient in a specified field. There are four git repositories in this release: `dissolFoam` – the solver, `libsFoamAux` – auxiliary libraries used by `dissolFoam` and related solvers, `utilities` – additional preprocessing and postprocessing tools, and `cases` – sample case files.

1 libsFoamAux

The solver relies on several libraries which are collected within `libsFoamAux`. A compendium of soft links `lnInclude` must be created alongside `libsFoamAux` with the script `libsFoamAux/mkInclude`

1.1 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.

1.2 boundary conditions

A Robin boundary condition can be written as

$$\boldsymbol{n} \cdot \boldsymbol{\nabla} S + \alpha S = 0, \tag{1}$$

where α is a function of position, but independent of the scalar field S. Both the dissolution kinetics (Eq. 6) and the Danckwerts condition (Eq. 8) can be mapped onto this equation. Within a standard finite-volume approximation, the field value on the boundary S_b is related to the field value at the cell center S by

$$S_b = \frac{S}{1 + \alpha\delta}, \tag{2}$$

where δ is the distance from the cell center to the boundary.

An implementation can derived from the OpenFOAM `mixedFvPatchField` class, which implements a Robin boundary condition,

$$S_b = f S_{\text{ref}} + (1 - f)(S + \delta g_{\text{ref}}), \tag{3}$$

with constant `valueFraction` (f), `refValue` (S_{ref}), and `refGrad` (g_{ref}) [1]. Our coded boundary conditions replace the constant values with functions

$$S_{\text{ref}} = g_{\text{ref}} = 0, \qquad f = \frac{\alpha\delta}{1 + \alpha\delta}, \tag{4}$$

A non-linear boundary condition, where S is replaced by a non-linear function of S, can be implemented within the same framework, by linearization with outer-loop iteration [3]; however it is not required for this paper.

1.3 normalMotionSlip

The `normalMotionSlip` class supports the relaxation of mesh points on the boundaries of the system. It is a generalization of the OpenFOAM `slip` class to include a prescribed normal motion of the boundary, derived from the gradient of a (dimensionless) scalar field:

$$\frac{d\boldsymbol{r}}{dt} = -D(\boldsymbol{n} \cdot \boldsymbol{\nabla} S)\boldsymbol{n}. \tag{5}$$

The motion of the points in the local tangent plane satisfies the slip condition. Although we do not yet allow for topological changes when moving the surface points, mesh relaxation by itself is effective in extending the time that fracture dissolution can be followed by several orders of magnitude.

1.4 velocityDeltatLaplacianFvMotionSolver

The projection operator applied by the tangential slip boundary condition requires an outer-loop iteration of the motion solver. A new motion solver `velocityDeltatLaplacianFvMotionSolver`, which is templated from the OpenFOAM `velocityLaplacianFvMotionSolver`, adds an outer-loop iteration to the standard Laplacian motion solver, with a tolerance set within `dynamicMeshDict`. The inner Laplacian solver relaxes displacements formed by `U*deltaT` so that the tangential motion remains small, controlled by the time step.

1.5 coupledPatchInterpolation

The `coupledPatchInterpolation` class adds synchronization across coupled patches to functions in the `primitivePatchInterpolation` class. It supports the motion of surface points across `processor` and `cyclic` patches. At present, only `faceToPoint` functions have been modified; `faceToEdge` functions are not yet correctly synchronized.

2 dissolFoam solver

The solver `dissolFoam` implements a dimensionless form of the equations laid out in Sec. 2 of the paper; the exact equations can be found in Sec. 2.2 of [3]. 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. It requires its own dictionary `dissolFoamDict` and depends on the library `steadyStateControl` (Sec. 1.1)

Robin boundary conditions on the concentration (Eq. 6) are now implemented as `coded`, meaning they are compiled at run time from code kept in the subdirectory `constant/bcInclude` of the case. In this way the exact code used in the boundary conditions is automatically include with the case. Other dependencies are much more stable and can best be provided by compiled libraries. The paper uses the coded boundary conditions `danckwerts` and `linear` (Sec. 1.2). A coded version of a nonlinear boundary condition is available on in libsFoamAux/boundaryConditions/nonlinear.H.

Boundary motion is provided by the `normalMotionSlip` boundary condition (in pointMotionU), templated from the OpenFOAM `normalMotionSlip` class (Sec. 1.3). The fieldname and a scalename must be provided as indicated in the case files. The boundary condition is solved iteratively by the `velocityDeltatLaplacianFvMotionSolver` (Sec. 1.4)

3 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.

3.1 dissolCalc

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

```
dissolCalc 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. This tool should be integrated with the OpenFOAM `postProcess` utility.

3.2 surfRoughGen

A preprocessing tool to generate rough surfaces by spectral synthesis. It is controlled by a dictionary `surfRoughGenDict`. It currently requires the deprecated mesh relaxation library `dissolMotion`.

3.3 genBlockMeshDict

A python tool to generate dictionaries for the `blockMesh` utility.

4 Cases

The repository `dissolFoamCases` orives sample case files that can be used to as a template for future simulations. They are configured for fracture dissolution (`dissol1706`), a soluble chip being dissolved by flow from above (`dissolChip`), and a soluble cylinder chip being dissolved by flow around it (`dissolCirc`). The last two cases model a specific microfluidic geometry [2]. Sample output files are available in `dissolFoamCases\examples` to facilitate code verification.

References

[1] Hrvoje Jasak. *Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows*. PhD thesis, University of London, 1996.

[2] F. Osselin, A. Budek, O. Cybulski, P. Kondratiuk, P. Garstecki, and P. Szymczak. Microfluidic observation of the onset of reactive infiltration instability in an analog fracture. *Geophys. Res. Lett.*, 43:6907–6915, 2016.

[3] Vitaliy Starchenko, Cameron J. Marra, and Anthony J. C. Ladd. Three-dimensional simulations of fracture dissolution. *J. Geophys. Res. Solid Earth*, 121:6421–6444, 2016.