AM119: Yet another OpenFoam tutorial

Prof. Trask

April 4, 2016

1 From advectionDiffusionFoam to implicitAdvectionDiffusionFoam

We're going to take advectionDiffusionFoam and make a new solver that will treat diffusion implicitly to remove the stiff $\frac{\nu \Delta t}{\Delta x^2}$ timestep restriction.

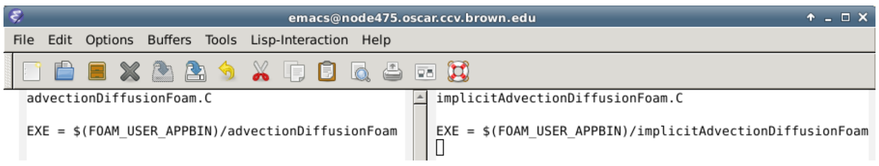
First, we will copy our advection-diffusion code into a new directory to make a custom solver.

```
$ source data/classMaterial/loadFoamModules.sh
$ run
$ cd ../solvers
$ cp advectionDiffusionFoam implicitAdvectionDiffusionFoam -r
```

We'll now go through and change everything to make a new solver

```
$ cd implicitAdvectionDiffusionFoam
$ mv advectionDiffusionFoam.C implicitAdvectionDiffusionFoam.C
$ cd Make
$ gedit files
```

We'll change the makefile to point to *implicitAdvectionDiffusionFoam.C* instead

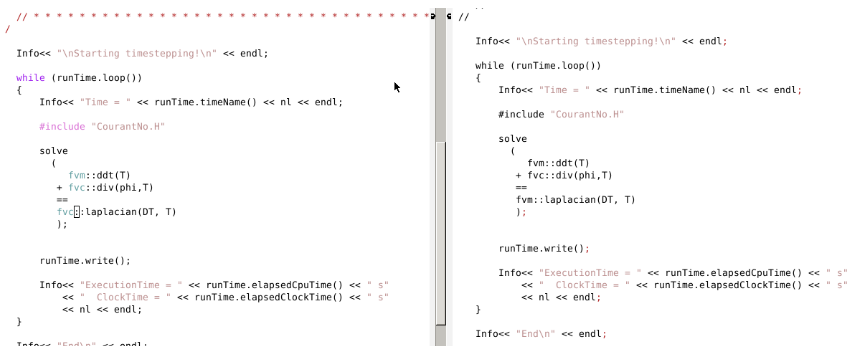

emacs@node475.oscar.ccv.brown.edu
File Edit Options Buffers Tools Lisp-Interaction Help

```
advectionDiffusionFoam.C

EXE = $(FOAM_USER_APPBIN)/advectionDiffusionFoam
```

```
implicitAdvectionDiffusionFoam.C

EXE = $(FOAM_USER_APPBIN)/implicitAdvectionDiffusionFoam
```

Figure 1: Change *Make/files* from the left to the right

The only change is completely trivial - alter the namespace for the Laplacian operator in the advection-diffusion equation from fvc to fvm.



```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Info<< "\nStarting timestepping!\n" << endl;

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

    #include "CourantNo.H"

    solve
      (
         fvm::ddt(T)
       + fvc::div(phi,T)
       ==
       fvc::laplacian(DT, T)
       );


    runTime.write();

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

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

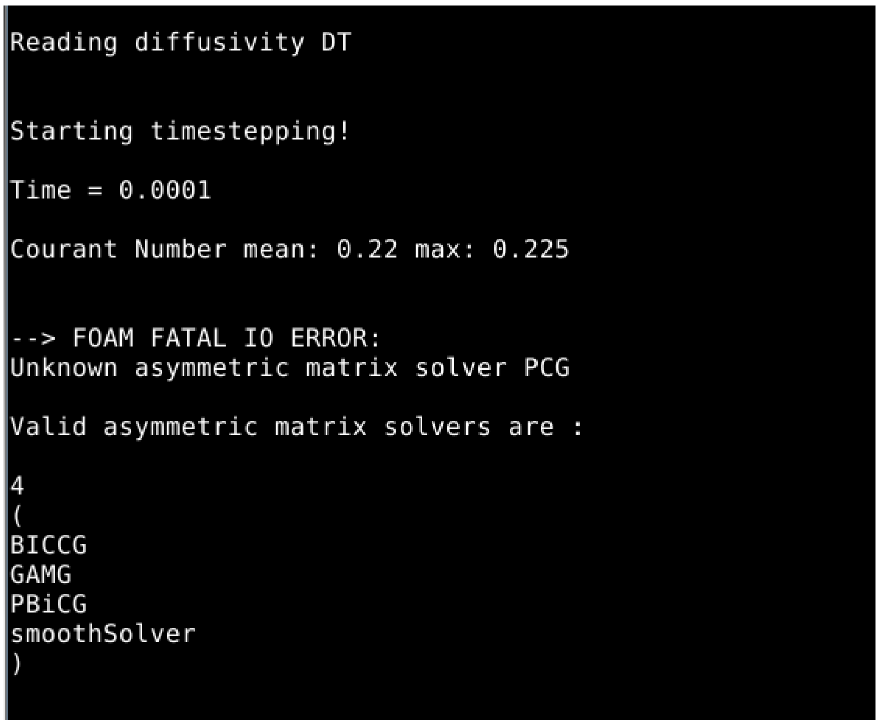
```
Info<< "\nStarting timestepping!\n" << endl;

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

    #include "CourantNo.H"

    solve
      (
         fvm::ddt(T)
       + fvc::div(phi,T)
       ==
       fvm::laplacian(DT, T)
       );


    runTime.write();

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

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

Figure 2: Change *implicitAdvectionDiffusionFoam.C* from the left to the right

and we're done! Run *wclean* and *wmake* and you should have a new implicit solver that you can run in the usual way.

You should be able to rerun the advectionDiffusionTest case with no modifications - try this.

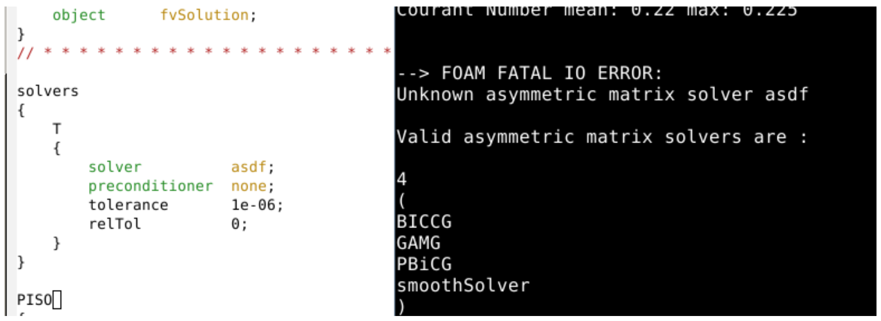
Let's also make the advection term implicit - alter the div term from fvc to fvm and recompile. If you attempt to rerun the case you will get the following error:



```
Reading diffusivity DT


Starting timestepping!

Time = 0.0001

Courant Number mean: 0.22 max: 0.225


--> FOAM FATAL IO ERROR:
Unknown asymmetric matrix solver PCG

Valid asymmetric matrix solvers are :

4
(
BICCG
GAMG
PBiCG
smoothSolver
)
```

Figure 3: Error due to asymmetric global matrix, and attempting to use the symmetric PCG solver

We'll need to go into *system/fvSolution* to alter the solver that we'd like to use. Foam was complaining because we're using preconditioned conjugate gradients (PCG), which is a more sophisticated linear solver that relies on the fact that the matrix is symmetric. By treating the advection term implicitly, we've altered the global matrix to obtain something that is no longer symmetric. To obtain a list of available solvers, change the solver from PCG to something nonsensical (in the figure below I've changed it to *asdf*) and rerun the solver to obtain the following error message.



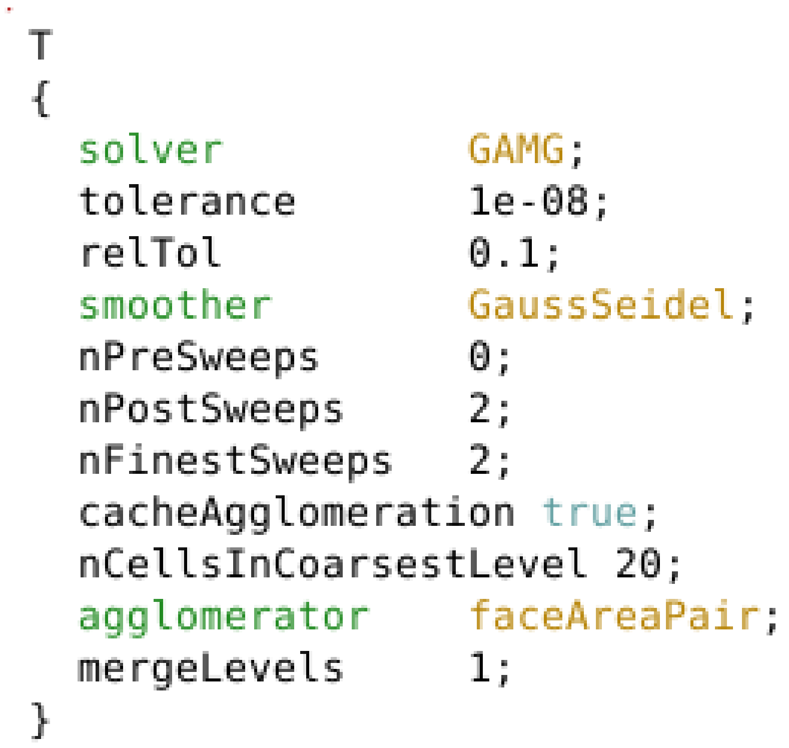
```
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * *

solvers
{
    T
    {
        solver          asdf;
        preconditioner  none;
        tolerance       1e-06;
        relTol          0;
    }
}

PISO
```

```
Courant Number mean: 0.22 max: 0.225


--> FOAM FATAL IO ERROR:
Unknown asymmetric matrix solver asdf

Valid asymmetric matrix solvers are :

4
(
BICCG
GAMG
PBiCG
smoothSolver
)
```

Figure 4: How to get Foam to give you a list of available options

We can see that Foam will accept solvers of type BICCG, GAMG, PBiCG, or smoothSolver. Let's change it to PBiCG - after that the code should be good to go.

The choice of linear solver for a given discretization is pretty sophisticated, and is one of the most fundamental issues of how to obtain a fast discretization. If you're interested, I'd suggest taking APMA117 for a starting point. For now, I'll point out that geometric/algebraic multigrid (GAMG) is the fastest solver available in OpenFoam. For a carefully constructed problem, the method will converge in $O(N)$ iterations. The method itself is pretty sophisticated and well beyond anything we'll discuss in the course - if you'd like to try it out you can swap the following settings in.



```
T
{
  solver              GAMG;
  tolerance           1e-08;
  relTol              0.1;
  smoother            GaussSeidel;
  nPreSweeps          0;
  nPostSweeps         2;
  nFinestSweeps       2;
  cacheAgglomeration true;
  nCellsInCoarsestLevel 20;
  agglomerator        faceAreaPair;
  mergeLevels         1;
}
```

Figure 5: Sample settings for GAMG solver

If you compare this to PBiCG, you should see that for this problem GAMG always converges in a single iteration, while PBiCG took 1-15, depending on how close the solution is to steady-state.