solverFoam

This solver is a modification of laplacianFoam- a solver to solve Laplace's equation for a scalar quantity. Here, laplacianFoam has been modified to port the assembled stiffness matrix and forcing vector to python, which enables one to perform the linear solve in Python as well, as a means to compare built-in solvers in OpenFOAM with linear equation solvers found in other utilities (e.g. NumPy, PETSc etc.).

OpenFOAM uses the Lower Diagonal Upper (LDU) addressing system to store matrices, where the coefficient entries are stored using face order numbering. Thus, an explicit stiffness matrix and forcing vector are not available in any solver. The major code modifications for this solver were to utilize the LDU addressing system to assemble the stiffness matrix and forcing vector, in addition to code coupling the OpenFOAM solver with Python.

By using the LDU system, OpenFOAM only stores the non-zero lower and upper triangular coefficients, and the diagonal components of the stiffness matrix. Face addressing is used to determine which faces do the coefficients belong to.

The lduMatrix and lduAddressing classes contain all utilities required for storing and accessing matrix coefficients. These two classes were utilized to create the matrix which was transferred to Python.

As opposed to other solvers in this repository, data needs to be transferred to Python at every timestep, so the data transfer functions are called within the time loop, otherwise the procedure to call python functions from within the solver is identical to other solvers in this repository.

The test case for this solver is a simple case of a thin plate subjected to constant temperature on certain faces, with Dirichlet BCs being applied on the other faces. The python function solves for the temperature distribution at each timestep using NumPy.linalg (along with the built-in OpenFOAM solver), as well as CG with jacobi preconditioning using PETSc and outputs the minimum temperature value across the entire domain.

```
Time = 2.365


created stiffness matrix

Created input_vals
DILUPBiCG:  Solving for T, Initial residual = 0.000578773, Final residual = 3.64047e-12, No Iterations 1
Created numpy arrays (array_2d and array_1d)
Entered solver function
Converged in 2 iterations.
L2 norm between OpenFOAM and NumPy solution
1.6896705156796816e-09
L2 norm between OpenFOAM and PETSc4Py solution
1.753969634259109e-05
Set array_2d as first argument of my_func_args tuple, array_1d as second argument and array_1d2 as third

created stiffness matrix

Created input_vals
DILUPBiCG:  Solving for T, Initial residual = 3.63461e-12, Final residual = 3.63461e-12, No Iterations 0
Created numpy arrays (array_2d and array_1d)
Entered solver function
Converged in 2 iterations.
L2 norm between OpenFOAM and NumPy solution
1.6896705156796816e-09
L2 norm between OpenFOAM and PETSc4Py solution
1.753969634259109e-05
Set array_2d as first argument of my_func_args tuple, array_1d as second argument and array_1d2 as third

created stiffness matrix

Created input_vals
DILUPBiCG:  Solving for T, Initial residual = 3.63461e-12, Final residual = 3.63461e-12, No Iterations 0
Created numpy arrays (array_2d and array_1d)
Entered solver function
Converged in 2 iterations.
L2 norm between OpenFOAM and NumPy solution
1.6896705156796816e-09
L2 norm between OpenFOAM and PETSc4Py solution
1.753969634259109e-05
```

Figure 1: Log output of solverFoam on running the plate case.

References-

fvMatrix.pdf (chalmers.se)

https://www.openfoam.com/