AM119: HW4 and more OpenFOAM tutorials

Prof. Trask

March 21, 2016

1 Assignment 4: Compressible Euler equations

In class this week we learned a bit about the Euler equations and how they give rise to shock waves in super-sonic flows. For this weeks assignment, we will implement both a 1D solver for the Euler equations and a full 3D solver in OpenFOAM. We learned that for compressible flows, information propogates at different speeds along the characteristics of the flow. In this assignment, we will make a crude approximation when constructing the limiters in our finite volume scheme using a linear reconstruction of velocity at faces, rather than constructing the eigensystem that would be necessary to solve this problem.

2 The Sod shock tube

For this assignment we will be simulating the Sod shock tube problem discussed in class. For this we will solve the compressible Euler equations

$$\mathbf{y}_t + (F(\mathbf{y}))_x = 0$$

where

$$\mathbf{y} = \begin{pmatrix} \rho \\ \rho u \\ \rho E \end{pmatrix}$$

and

$$F(\mathbf{y}) = \begin{pmatrix} \rho u \\ \rho uu + p \\ \rho uE + up \end{pmatrix}$$

and we will adopt the ideal gas equation of state

$$p = \rho(E - \frac{1}{2}u^2)(\gamma - 1)$$

where $\gamma = 1.4$ is the ratio of specific heats for air. This EOS is equivalent to the ideal gas equation $p = \rho RT$ that we might be more familiar with.

We will initialize the problem with the values, $\rho_l = 1.0$, $\rho_r = 0.125$, $p_l = 1.0$, $p_r = 0.1$, and $u_l = u_r = 0$. From these values we can recover the energy from the equation of state, and knowledge of ρ,u, and E are enough to initialize $\mathbf{Y}$.

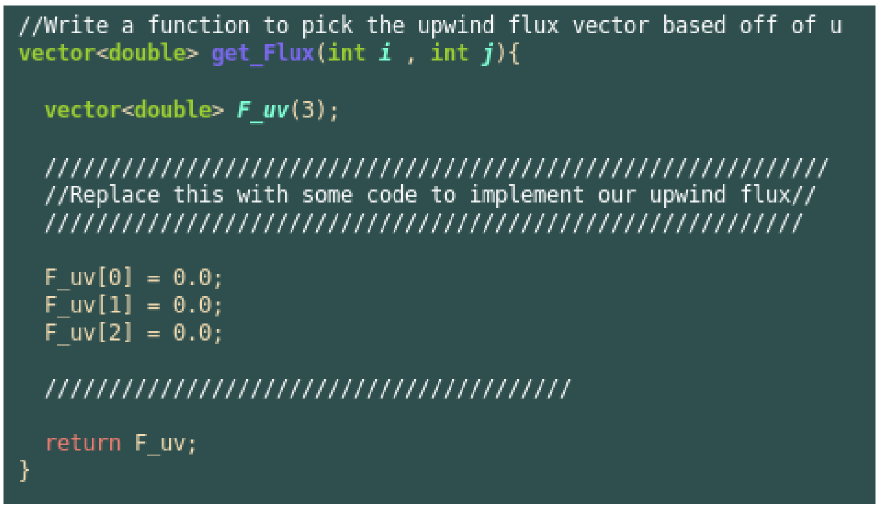
3 Part 1: 1D code

I've done you a favor and wrote most of the code for this assignment (see this week's postings on the website). Your job will be to implement the following limiter:

When constructing the flux $F(\mathbf{y})_{i+\frac{1}{2}}$, we will first reconstruct the velocity at the face $f_{i+\frac{1}{2}}$ with the linear extrapolation $u_{i+\frac{1}{2}} = \frac{u_i + u_{i+1}}{2}$. We will then calculate the flux as follows

$$F(\mathbf{y})_{i+\frac{1}{2}} = \begin{cases} \begin{pmatrix} \rho_i u_{i+\frac{1}{2}} \\ \rho u_i u_{i+\frac{1}{2}} + p_i \\ \rho u_{i+\frac{1}{2}} E_i + u_{i+\frac{1}{2}} p_i \end{pmatrix} & \text{if } u_{i+\frac{1}{2}} \geq 0 \\ \begin{pmatrix} \rho_j u_{i+\frac{1}{2}} \\ \rho u_j u_{i+\frac{1}{2}} + p_j \\ \rho u_{i+\frac{1}{2}} E_j + u_{i+\frac{1}{2}} p_j \end{pmatrix} & \text{if } u_{i+\frac{1}{2}} < 0 \end{cases}$$

This flux handles advection terms with the reconstructed velocity at faces, while using upwind values for the advected quantities and forcing terms. You will only need to modify the following bit of code



```
//Write a function to pick the upwind flux vector based off of u
vector<double> get_Flux(int i , int j){

  vector<double> F_uv(3);

  ///////////////////////////////////////////////////////////////
  //Replace this with some code to implement our upwind flux//
  //////////////////////////////////////////////////////////////

  F_uv[0] = 0.0;
  F_uv[1] = 0.0;
  F_uv[2] = 0.0;

  ////////////////////////////////////////////

  return F_uv;
}
```

Assignment: Implement the above limiter, and reproduce the following plot of the shock profile at time $t = 0.1$.

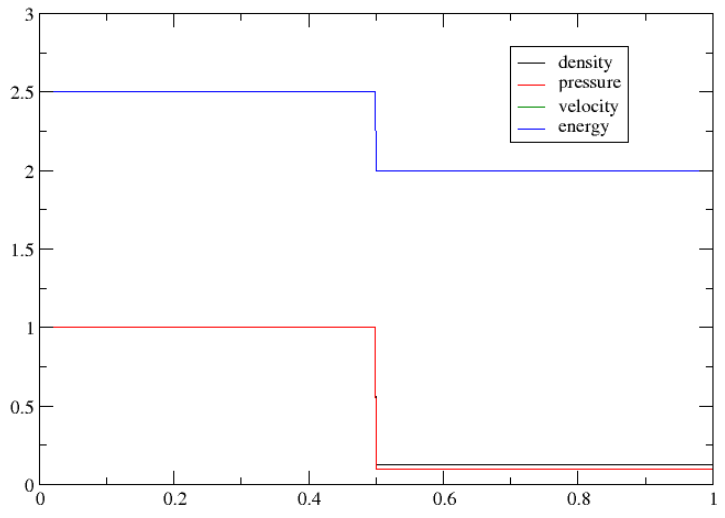


Figure 1: Initial condition

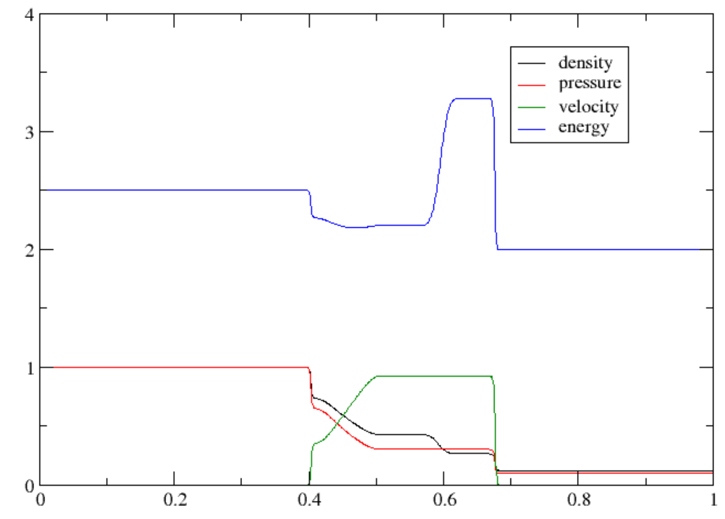


Figure 2: Solution at $t = 0.1$.

You will see that the code is nearly identical to your code for Burgers equation. If you're feeling ambitious you should take a shot at writing the whole code yourself.

4 Part 2: OpenFOAM

The second part of the homework will be to write an identical solver in Foam. This will be more of a tutorial - there is very little work to do here but this will give you a code base that we will use after Spring break to develop a full Navier-Stokes solver.

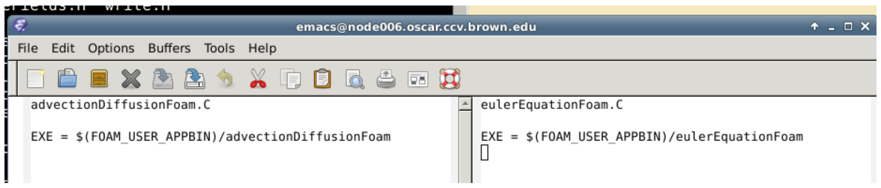
First, we will copy our advection-diffusion code into a new directory to make a custom solver.

```
$ source data/classMaterial/loadFoamModules.sh
$ run
$ cd ../solvers
$ cp advectionDiffusionFoam eulerEquationFoam −r
```

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

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

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



```
advectionDiffusionFoam.C

EXE = $(FOAM_USER_APPBIN)/advectionDiffusionFoam
```

```
eulerEquationFoam.C

EXE = $(FOAM_USER_APPBIN)/eulerEquationFoam
```

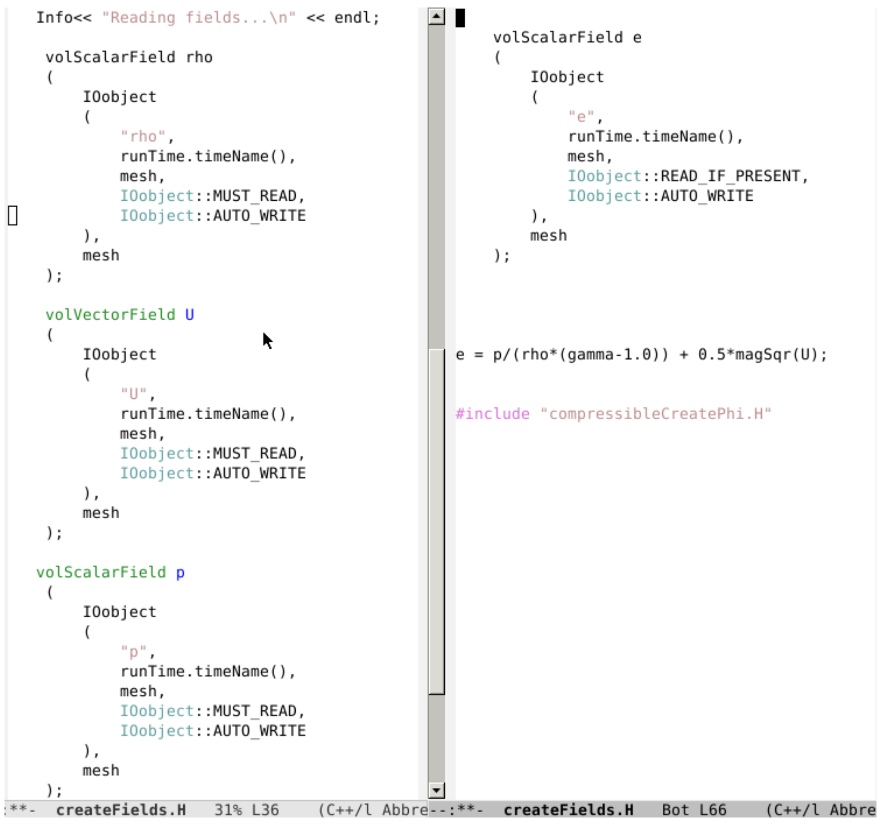
Figure 3: Change *Make/files* from the left to the right

We need a bunch more fields for this problem than the advection-diffusion equation, so next we'll tweak the *createFields.H* file.

First off, we'll need the user to input the ratio of specific heats. We'll get rid of the previous code where we input the diffusion and advection coefficients and replace it as follows:

```
Info<< "Reading transportProperties\n" << endl;

IOdictionary transportProperties
(
    IOobject
    (
        "transportProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
 );

Info<< "Reading material properties\n" << endl;



dimensionedScalar gamma
(
    transportProperties.lookup("gamma")
);
```

Next we need to evolve density, velocity, pressure, and internal energy. We'll handle the first three in the same way as we did with the single field in the advection-diffusion problem. The energy field will be slightly different, as it is specified by an equation of state. Finally, we call a header file that will initialize $\phi = \rho U$ evaluated at faces.



```
    Info<< "Reading fields...\n" << endl;

     volScalarField rho
     (
         IOobject
         (
             "rho",
             runTime.timeName(),
             mesh,
             IOobject::MUST_READ,
             IOobject::AUTO_WRITE
         ),
         mesh
     );

     volVectorField U
     (
         IOobject
         (
             "U",
             runTime.timeName(),
             mesh,
             IOobject::MUST_READ,
             IOobject::AUTO_WRITE
         ),
         mesh
     );

    volScalarField p
     (
         IOobject
         (
             "p",
             runTime.timeName(),
             mesh,
             IOobject::MUST_READ,
             IOobject::AUTO_WRITE
         ),
         mesh
     );
:**-  createFields.H    31% L36      (C++/l Abbre
```

```

    volScalarField e
    (
        IOobject
        (
            "e",
            runTime.timeName(),
            mesh,
            IOobject::READ_IF_PRESENT,
            IOobject::AUTO_WRITE
        ),
        mesh
    );



e = p/(rho*(gamma-1.0)) + 0.5*magSqr(U);


#include "compressibleCreatePhi.H"
--:**-  createFields.H    Bot L66     (C++/l Abbre
```

At this point we've successfully initialized all of the fields - all that's left is to alter the timestepping. Again - this will be similar to the advection diffusion case but now we've got a bunch more fields to handle. This can be handled as follows:

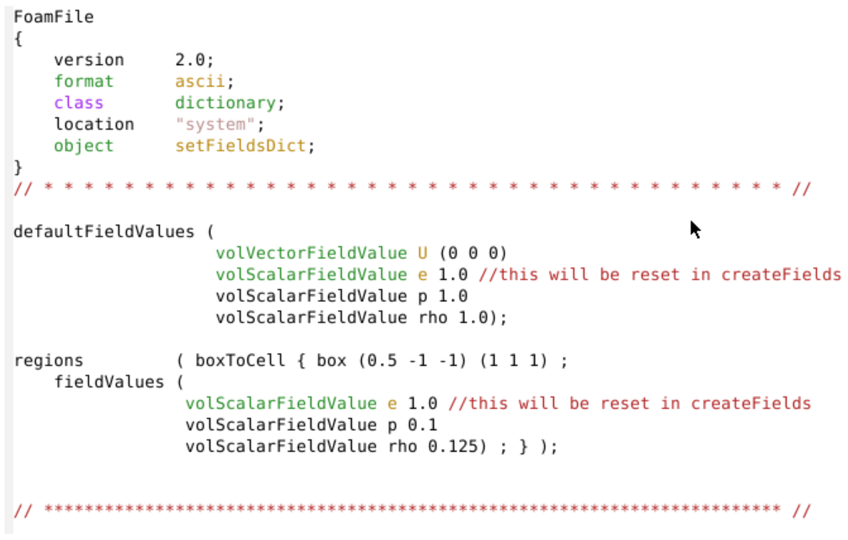
```
Info<< "\nStarting timestepping!\n" << endl;

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

    #include "CourantNo.H"

    //Solve momentum equation
    solve
      (
         fvm::ddt(rho,U)
       + fvc::div(phi,U)
         ==
       - fvc::grad(p)
       );
    //Solve energy equation
    solve
      (
           fvm::ddt(rho,e)
         + fvc::div(phi,e)
         ==
         - fvc::div(phi,p/rho)
       );
    //Solve continuity equation
    solve
      (
         fvm::ddt(rho)
       + fvc::div(phi)
       );

    //Update variables
    phi = fvc::interpolate(rho*U) & mesh.Sf();
    p = rho*(e - 0.5*mag(U)*mag(U))*(gamma-1.0);


    runTime.write();

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

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

And we're done! If we save all of our changes, and compile in the terminal:

```
$ run
$ cd ../solvers/advectionDiffusionFoam
$ wclean
$ wmake
```

We should have a new solver called advectionDiffusionFoam. You can find a new case in */data/foamCases/shockTube*. This case is set up to run the Sod shock tube problem. Copy that over to your run directory. There's one new thing we'll need to check out to set up the initial condition for the problem

```
$ run
$ cp ~/data/foamCases/shockTube . −r
$ cd shockTube
$ gedit system/
```

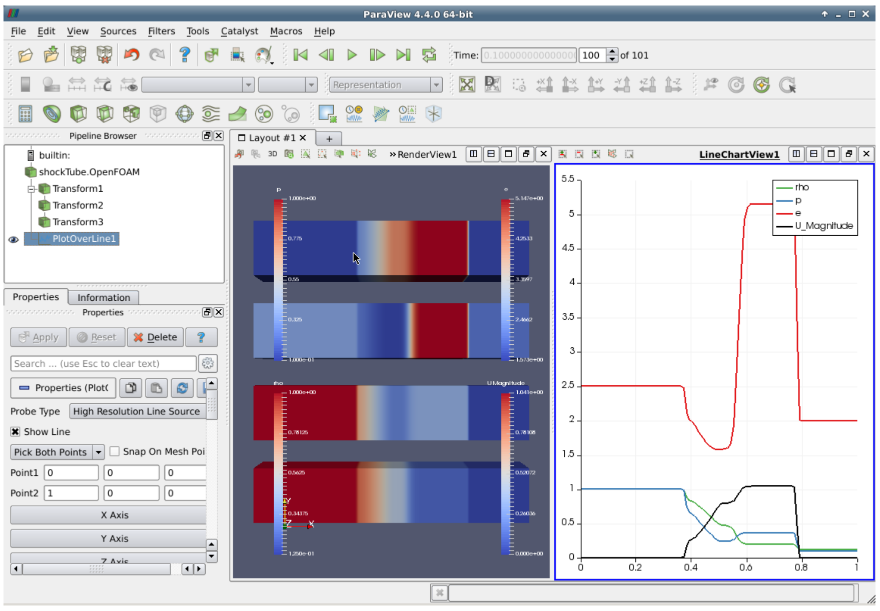


```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues (
                    volVectorFieldValue U (0 0 0)
                    volScalarFieldValue e 1.0 //this will be reset in createFields
                    volScalarFieldValue p 1.0
                    volScalarFieldValue rho 1.0);

regions         ( boxToCell { box (0.5 -1 -1) (1 1 1) ;
    fieldValues (
                 volScalarFieldValue e 1.0 //this will be reset in createFields
                 volScalarFieldValue p 0.1
                 volScalarFieldValue rho 0.125) ; } );


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

This is where we can set the non-uniform initial condition. What this does is set a uniform default value everywhere in the domain. It then identifies a region (in this case the box with bottom left corner $(0.5 - 1 - 1)$ and upper right corner (111)), and applies a uniform field to every cell falling in that region. The workflow for this case looks like:

```
$ run
$ cd shockTube
$ cp original.0/∗ 0/
$ blockMesh
$ setFields
$ eulerEquationFoam
$ paraFoam
$
```

And you can post-process your results in the usual way.



Time: 0.100000000000000 100 of 101

builtin:
shockTube.OpenFOAM
Transform1
Transform2
Transform3
PlotOverLine1

Search ... (use Esc to clear text)
Properties (PlotOverLine1)
Probe Type High Resolution Line Source
☒ Show Line
Pick Both Points ☐ Snap On Mesh Point
Point1 0 0 0
Point2 1 0 0

Assignment: Plot the fields at $t = 0.1$ and demonstrate that you've obtained the same results with OpenFOAM as you have with your 1D code.