

AM119: HW3 and OpenFOAM tutorial

Prof. Trask

March 7, 2016

1 Assignment 3: Burgers equation component

In class, we learned how the balance of momentum gives rise to non-linear flux terms. To obtain an understanding of how this translates to our 1D periodic example, we're going to rerun last weeks code solving Burger's equation instead. Whereas the advection equation had the form:

$$\partial_t u + a \partial_x u = 0$$

We will now we solving what is referred to as **Burger's equation**:

$$\partial_t u + u \partial_x u = 0$$

We can write this in conservative form as:

$$\partial_t u + \partial_x \left(\frac{1}{2} u^2 \right) = 0$$

or

$$\frac{du}{dt} + \nabla \cdot F = 0$$

$$F = \frac{1}{2} u^2$$

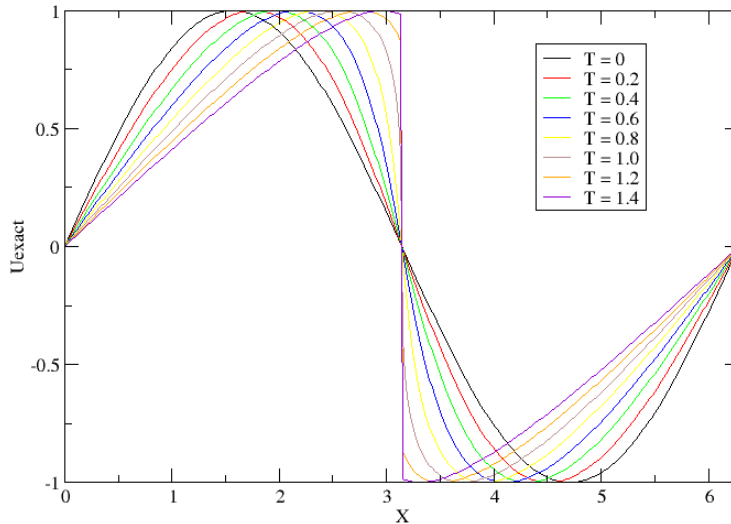
This code will be **identical to your code from HW2**, and will only require the modification of the flux calculation. Whereas previously we treated the advection flux using upwinding, the upwind direction for Burgers equation will depend upon whether u is positive or negative. We can write this mathematically via

$$V_i \frac{u_i^{n+1} - u_i^n}{\Delta t} = - (F_{i+1/2} - F_{i-1/2})$$

where $F_{i+1/2} = f(u_i, u_{i+1})$ and $F_{i-1/2} = f(u_{i-1}, u_i)$, where f is calculated via

$$f(u, v) = \begin{cases} F(\min(u, v)) & \text{if } u \leq v \\ F(\max(u, v)) & \text{if } u \geq v \end{cases}$$

For this assignment, implement the new flux term, run your solver to $T_{final} = 1.4$ and demonstrate that your solution matches the following analytic result.



2 Incompressible flow through a pipe

In the notes for Lecture 3 we discussed an example where initially uniform pipe flow settles into a parabolic velocity profile, and through a mass conservation argument we derived a relationship between the inlet velocity U_∞ and the downstream velocity profile. Today, we will set up an OpenFOAM simulation to demonstrate numerically that this relationship is true - we will see later in the course how to derive the velocity profile analytically.

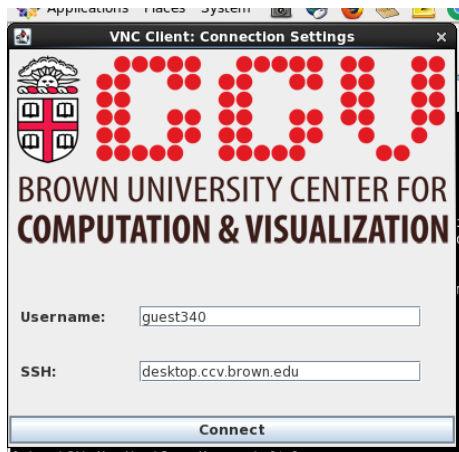
Before we start, you'll need to log into Oscar through CCV's VNC client.

In our last homework assignment we implemented the finite difference method to solve the 1D advection diffusion equation. Today, we've introduced the finite volume method, which we will use at length for the remainder of the class. For this assignment, we will be repeating the previous homework but swapping in a finite volume discretization. This should require minimal changes to the previous homework assignment. A solution to the previous assignment will be put on the website tomorrow (3/1) - if you had difficulty completing the previous assignment feel free to use the solution as a starting point.

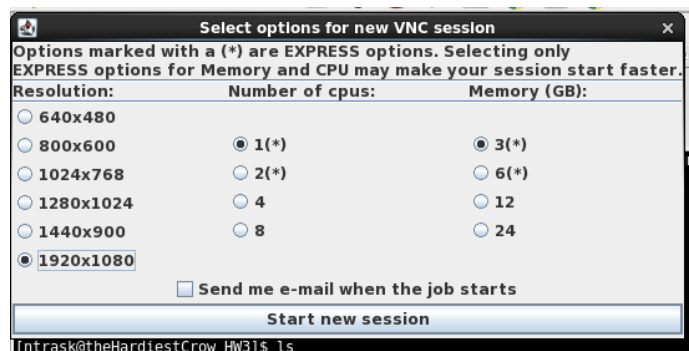
3 Get onto Oscar

We've gone over this in class but I'll gather the relevant bits and pieces here for reference. First download the VNC client from the CCV webpage (<https://www.ccv.brown.edu/technologies/vnc>). You'll need Java to launch this - Java is on all of the machines in the CIT, but see me if you need help getting

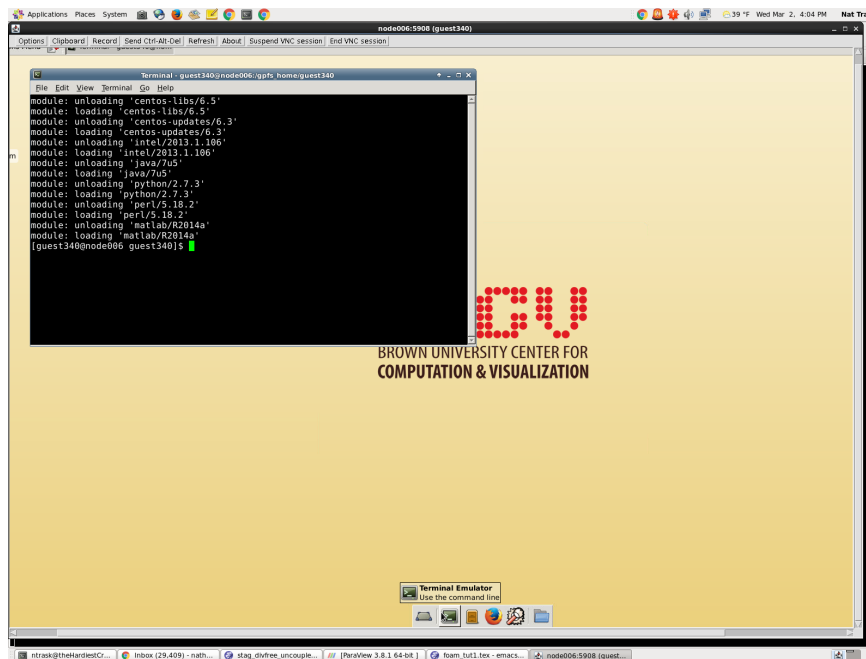
this up on your home machine. Once you launch the jar file, you should see a scene that looks like:



Enter your username and password (case sensitive, and be careful as 5 unsuccessful logins will lock your account). Next some configuration options will pop up. You want the default options, but you might want to play with the resolution settings to find a desktop that matches your screen resolution best. For the purposes of this class, there is no benefit to choosing more processors, and doing so will make your session take longer to load. You might get a message that your session hasn't loaded yet - just keep clicking retry until you're successfully logged in.



You should now have a virtual desktop setup - congratulations!!! you are now remotely logged into the department cluster. What we'll do now is start up a terminal - click on the black boxy looking thing in the tray at the bottom of the screen.



This terminal is where we'll spend all of our time. I've included a cheatsheet of useful Unix commands on the course website. To get warmed up, open a terminal and run the following set of commands to: look at the contents of your home directory, move into the data directory, look around there, and move back home. You should see the following output.

```
$ ls  
$ cd data  
$ ls  
$ cd ~
```

```

Terminal - guest340@node006:--
File Edit View Terminal Go Help
module: unloading 'intel/2013.1.106'
module: loading 'intel/2013.1.106'
module: unloading 'java/7u5'
module: loading 'java/7u5'
module: unloading 'python/2.7.3'
module: loading 'python/2.7.3'
module: unloading 'perl/5.18.2'
module: loading 'perl/5.18.2'
module: unloading 'matlab/R2014a'
module: loading 'matlab/R2014a'
[guest340@node006 guest340]$ ls
batch.script batch_scripts Code data Desktop README scratch TERMS
[guest340@node006 guest340]$ cd data
[guest340@node006 data]$ ls
classMaterial guest325 guest331 guest337 guest379 main.cpp
FoamTest guest326 guest332 guest338 guest380 quota-report.txt
guest321 guest327 guest333 guest339 guest381 #test.cpp#
guest322 guest328 guest334 guest340 guest383 test.cpp
guest323 guest329 guest335 guest377 helloworld
guest324 guest330 guest336 guest378 #main.cpp#
[guest340@node006 data]$ cd ~
[guest340@node006 ~]$ pwd
/users/guest340
[guest340@node006 ~]$

```

If this is your first Unix experience - don't worry. It'll take some time to get comfortable moving around using a command line instead of a GUI, but you'll get the hang of it. Next up we're going to set up the OpenFoam environment. I've stored some useful scripts in the `data/classMaterial\verb` directory. A script is just a collection of terminal commands bundled together in a convenient way. To check them out, move to the `classMaterial` directory, and take a look at the setup script

```

$ cd ~
$ cd data/classMaterial
$ ls
$ gedit loadFoamModules.sh

```

You should be able to see a bunch of commands that this script will execute.

```

Terminal - guest340@node009:igpfs_home/guest340/data/classMaterial
File Edit View Terminal Go Help
module: unloading 'java/7u5'
module: loading 'java/7u5'
module: unloading 'python/2.7.3'
module: loading 'python/2.7.3'
module: unloading 'perl/5.18.2'
module: loading 'perl/5.18.2'
module: unloading 'matlab/R2014a'
module: loading 'matlab/R2014a'
[guest340@node009 guest340]$ cd data/
classMaterial/ guest328/ guest337/ guest383/
FoamTest/ guest329/ guest338/ hellowo
guest321/ guest330/ guest339/ #main.c
guest322/ guest331/ guest340/ main.cp
guest323/ guest332/ guest377/ quota-r
guest324/ guest333/ guest378/ #test.c
guest325/ guest334/ guest379/ test.cp
guest326/ guest335/ guest380/
guest327/ guest336/ guest381/
[guest340@node009 guest340]$ cd data/classMaterial/
[guest340@node009 classMaterial]$ ls
initFoamDirectories.sh loadFoamModules.sh
[guest340@node009 classMaterial]$ gedit loadFoamModules.sh
(gedit:2982): EggSMClient-WARNING **: Could not load desktop
BKUWN UNIVE
COMPUTATION
loadFoamModules.sh [Read Only] [gpfs/data/am119/classMaterial] - gedit
loadFoamModules.sh
#!/bin/bash
module load openfoam
module load openmpi
module load gcc/4.8.2
module load mvapich2
module load paraview
./gpfs/runtime/opt/openfoam/3.0+/OpenFOAM-v3.0+/etc/bashrc
echo "OpenFOAM-v3.0+ environment loaded"
Plain Text Tab Width: 8 Ln 1, Col 1 INS

```

To execute the scripts in this directory, enter

```
$ source ~/data/classMaterial/initFoamDirectories.sh
$ source ~/data/classMaterial/loadFoamModules.sh
```

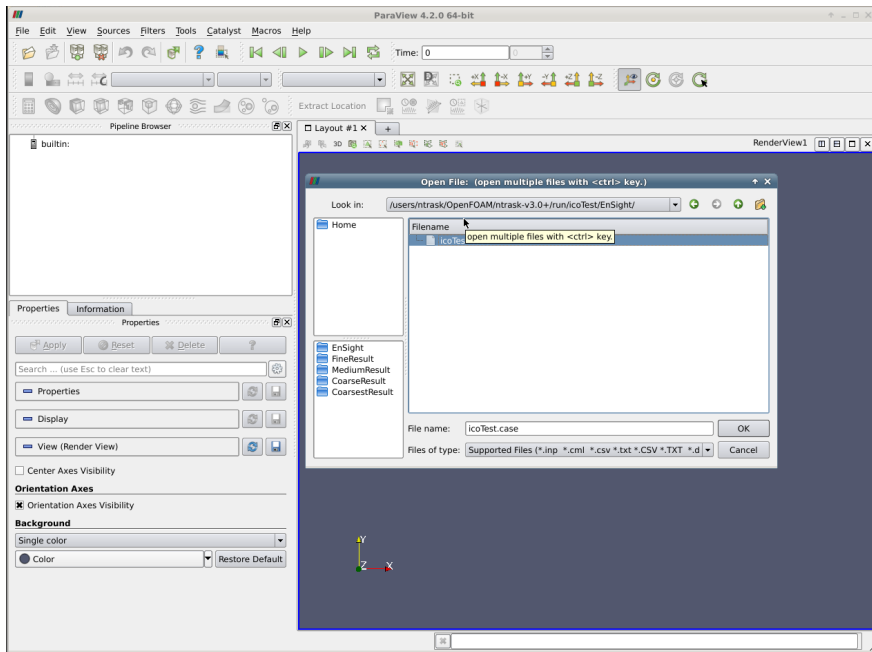
At this point OpenFoam is set up to run! In the future, you only need to run `loadFoamModules.sh` - the other script only needs to be run once to initialize your home directory to be set up with OpenFoam. To get a handle on how to use OpenFoam applications, we're going to simulate the fully developed pipeflow case that we looked at in class. In the data directory you can find a folder called `foamCases`. We're going to copy that into your user directory and run it. OpenFoam has a bunch of terminal shortcuts to make it easy to jump around - for example, typing `run` will take you to your runtime directory.

```
$ run
$ pwd
$ cp ~/data/foamCases/channelCase . -r
$ ls
```

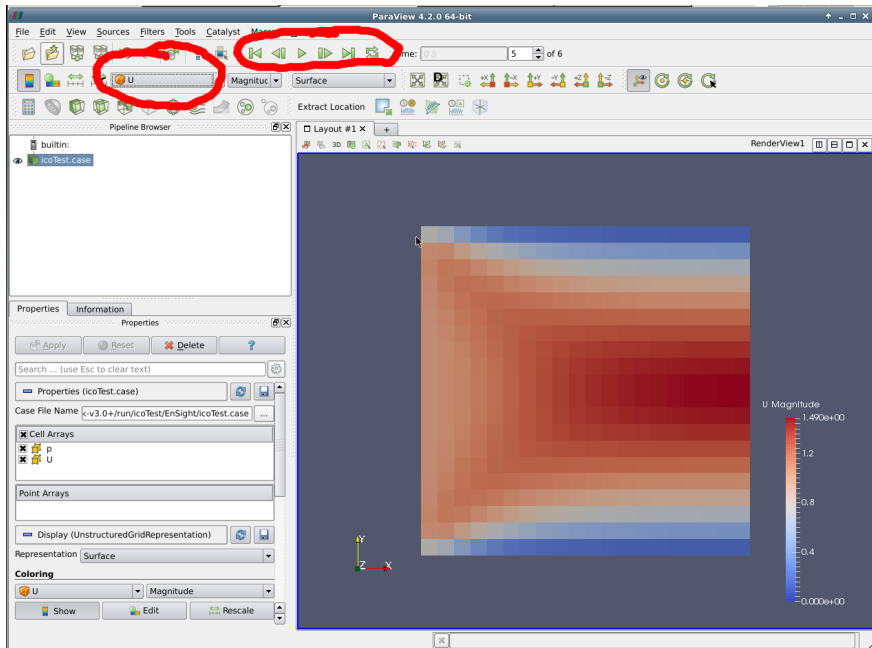
Now you've got a copy of the tutorial case in your run directory that you can mess around with. This case is set up to run the channel flow problem we went over in class. We'll explore the settings for this case, but first let's run the incompressible fluid flow solver and take a look what the flow looks like. We'll run the solver (`icoFoam`), convert the output to a file format that works with the CCV setup (`Enight`), and open a post-processing tool (`paraview`)

```
$ icoFoam
$ foamToEnight
$ paraview
$
```

Paraview will start up, and we'll load the case output by going to `File > Load` and opening `Enight/icoTest.case`.



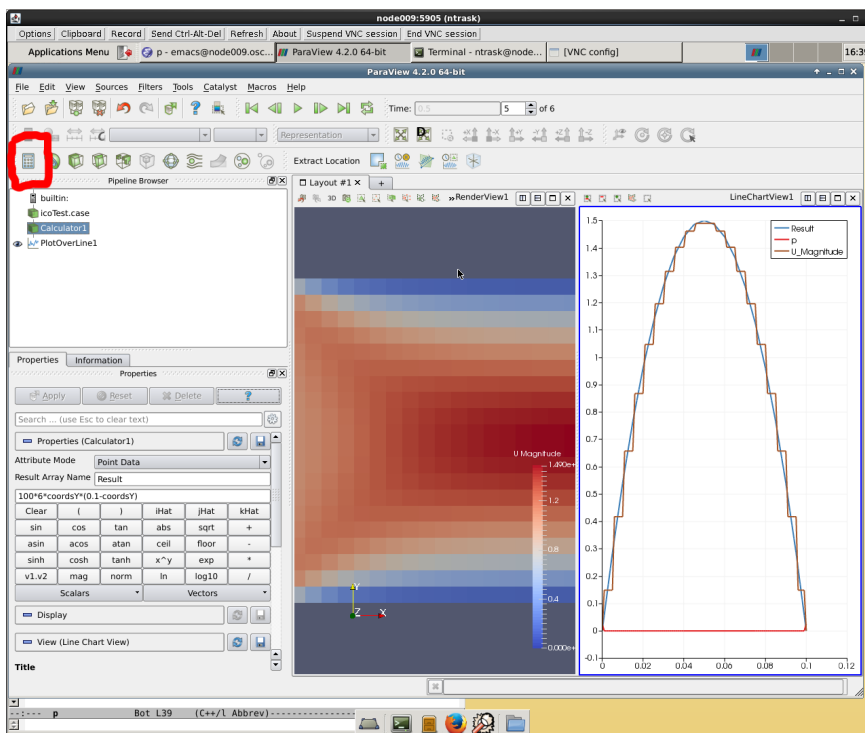
If you click the green button that says “Apply”, the geometry will load. We’re looking at a 0.1×0.1 box, with a specified uniform flow coming in for the left and walls at the top and bottom. What you’re looking at right now is the initial value ($t = 0$) of the pressure - you can move to different timesteps or switch between the velocity and pressure field by playing with the options circled in red below.



Let's make a plot of the downstream velocity profile. Select from the menu **Filters > Alphabetical > Plot over line**, enter values for the endpoints of the line as (0.1, 0, 0.005) and (0.1, 0.1, 0.005), and click apply. The velocity profile that you get should look pretty parabolic. Let's see how well it matches up with our theoretical profile. We know from the derivation in class that the velocity profile should satisfy

$$u(y) = \frac{6U_\infty}{L^2} y(L - y)$$

Let's punch that into the calculator to generate a field we can compare to, and plot how that matches up with what we expect.



Looks good! Now, for the rest of the tutorial, what we're going to do is explore the case directory and I'll show you how you can tweak the case setup so that you can tackle the questions in HW3. You can go ahead and close paraview. Let's take a look at the directory - you'll see: some numbered folders corresponding to timesteps, a folder called constant where we specify physical parameters, and a folder called system where we setup the case. To start with, lets go into the system folder and take a look at all of the files. We'll start with `blockMeshDict`.

```
$ ls
$ cd system
$ ls
$ gedit blockMeshDict
```

This is where we specify the geometry for our case. In general, making a mesh can be a pretty intense job. `blockMesh` is an OpenFoam utility for handling very

simple geometries. We do this by defining the vertices of blocks and explaining how to stitch them together. I'll explain the details of this on the board.

```

format      ascii;
class      dictionary;
object     blockMeshDict;
}
// ***** //

convertToMeters 0.1; //After making the geometry, scale it down by a factor of 0.1. This is useful for switching units.

vertices //The corners of our block mesh
(
(0 0 0)
(1 0 0)
(1 1 0)
(0 1 0)
(0 0 0.1)
(1 0 0.1)
(1 1 0.1)
(0 1 0.1)
);

blocks
(
hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1) //How the vertices are tied together to form a block. (20 20 1)
// specifies we want 20 cubes in the x-direction, 20 in the
// y-direction, and 1 in the z-direction
);

edges
(
);

boundary //Here we specify which vertices belong to the faces where we'll apply boundary conditions
(
inlet
{
type wall;
faces
(
(0 4 7 3)
);
}
outlet
{
type wall;
faces
(
(2 6 5 1)
);
}
);

```

To rebuild the mesh according to any changes we make in the blockMesh dictionary, we'll drop back down to the case directory and run the utility

```

$ cd ..
$ blockMesh

```

If you pop open the other dictionaries in the system folder, you'll see that controlDict is where we specify timestepping parameters, fvSchemes is where we specify which sort of fluxes we want to use, and fvSolution is where we specify how to solve linear systems (we haven't talked about the last one in class yet). For HW3, we'll only need to play with the geometry settings so I won't discuss those guys, other than to say they control the foam solver in much the same way as your 1D solver parameters did.

Next up, we'll dig into the constant/transportProperties dictionary. You can see that the only thing there is a definition of the viscosity coefficient. You can tweak this to change the viscosity of the fluid in the simulation.

Finally, one of the numbered directories in the case directory is called 0. This is the initial condition. When you run the solver, it'll make a bunch of other numbered folders saving the output of the simulation at the corresponding times. We initialize the flow by going into the initial condition directory and

specifying the initial values and boundary conditions there. If you go into this directory, you'll see a file called `U` and another called `p`, corresponding to the velocity and pressure fields. If you pop open the velocity field, you can see the components of what we call a *field* in OpenFoam. A field is a list of values at cell centers in the interior of the domain (*internalField*) and a list of values at face centers on the boundary of the domain (*boundaryField*). You can see that this case is set up so that the velocity on the interior is initially at rest. At the inlet we impose a uniform unit velocity. The walls maintain what's called a no-slip boundary condition, and at the outlet we enforce a zeroGradient boundary condition, which means that $\partial_n u = 0$ for each component of the velocity vector. We'll get into the details of boundary conditions later in the class, but for HW3 you'll be tweaking the inlet velocity.

4 Assignment 3: Foam component

This exercise is going to be pretty informal, and should be completed in the lab session following lecture today. It turns out that for this problem, the distance downstream that it takes for the velocity to settle into a parabolic profile is roughly inversely proportional to the viscosity. Your job is to:

- Change the viscosity from 0.01 to 0.0005
- Run the solver again - if we generate the plot we can see that at the outlet we're still pretty far from the parabolic downstream profile.
- Double the resolution. Convince me that your answer is refined enough that you trust your answer.
- Get into the blockMesh dictionary and extend the domain. Do some experimenting - if you make it too long the code will get progressively more expensive. Give me a bunch of plots showing me that you can characterize (roughly) the point downstream where the profile is pretty much parabolic.
- Play with the boundary conditions: use our formula relating the downstream flow to the inlet velocity to show that we get the correct answer when you change the inlet from $U = 1$ to $U = 2$ or $U = 3$.