entcode multiarray project – pipeline

Lindsay Waldrop

9/24/2020

Introduction

This simulation and data analysis pipeline for the entcode multiarray project. It includes all the steps necessary to generate results through each step of the pipeline from cloning the project on Github to figures for publication.

Overview of the Pipeline

This pipeline takes the following inputs:

- IBAMR source code for rigid body constraint method.
- Generalized polynomial chaos (gPC) simulation set with n number of simulations (either 165, 681, or 1233) establishing parameter values for angle of array to flow, gap width to diameter ratio, and Reynolds number.

This pipeline exists for the follow outputs:

- IBAMR flow data
- Calculated shear around hairs
- Calculated leakiness through each row of the array

The pipeline has the following components:

1. The generation of directories and compiling of the main code (main2d) from C and C++ IBAMR source code.
2. The generation of input files necessary for IBAMR (vertex files, input2d files) and data analysis in VisIt (csv files, lineout files) for the gPC simulation set.
3. The execution of IBAMR simulations for the gPC simulation set.
4. The execution of VisIt data analysis to extract fluid speed information from each IBAMR simulation.
5. The execution of R scripts for calculation of final values.
6. The generation of figures for publication from analyzed data.

1. Cloning from Github, establishing directories, and compiling IBAMR code

To start this project, clone the project from Github using the following in command-line bash:

```
$ git clone https://github.com/lindsaywaldrop/entcode.git
```

After entering the cloned directory, save a path to your main working directory:

```
$ WD=$PWD
```

You will then need to set up the IBAMR code to run with your build of IBAMR. Enter the `src/ibamr` director and edit the following lines of the Makefile:

- Line 4: path to your IBAMR source code
- Line 5: path to the IBAMR build you wish to use

After that, enter the `src/bridges` directory. Run the script in command-line bash:

```
$ sh compile_sniff.sh "$WD"
```

This shell script will set up the directories necessary for the organization of the project and then compile the `main2d` executable program that runs IBAMR simulations.

2. Generation of input files for IBAMR

For IBAMR to successfully run, each simulation needs two files: an input2d file, and a vertex file containing hair positioning data. You will need to generate each of these files based on which gPC simulation set you wish to use.

Input2d file generation

Input2d files are generated using the shell script `setinput2d.sh` in the `src/bridges` folder. Enter this directory and run the script, where n should be the number of simulations in the gPC set (165, 681, or 1233):

```
$ sh setinput2d.sh "$WD" n
```

If you should need to make any changes to the template file, it is `template-input2d1`. Make these changes and then re-run the setinput2d.sh script.

The input2d files will be located in `data/input2d-files`.

Vertex file generation

Vertex files describe the positions of the antennule and each hair in the array. These are divided up by the number of hairs in each array:

- 7 hairs (two rows)
- 12 hairs (three rows)
- 18 hairs (four rows)
- 25 hairs (five rows)

Each vertex file needs to be different depending on the parameters set by the gPC simulation set. Additionally, the center x and y coordinates of each hair are required for data analysis in csv files. Both sets of files are generated by `generate_grid2d_multirow.R` in `src/r-scripts`. To run this file, enter in the appropriate directory and in R and edit the following lines of this script:

- Line 29: the endrun parameter should equal the number of simulations in the gPC set (either 165, 681, or 1233)
- Line 30: the number of hairs in the array (either 7, 12, 18, or 25)

Once changed, run the file using the R console:

```
> source('generate_grid2d_multirow.R')
```

The vertex files will be located in `data/vertex-files` in a directory corresponding to the number of hairs in the array. The csv files will similarly be located in `data/csv-files`.

3. Execution of IBAMR simulations

Note: these instructions are specifically for the Bridges cluster at PSC, but they can be modified to include any standard HPC cluster running SLURM as a job-management system.

There are two files in `src/bridges` that control submitting batch jobs to the Bridges cluster: `runhairs.job` and `setrunhairsjobs.sh`. `setrunhairsjobs.sh` generates a set of temporary job files and submits them via sbatch, based on `runhairs.job` as a template.

Each simulations takes between 30-40 minutes to run on Bridges as it is currently configured. However, should you wish to change parameters, lines 2 through 6 in `runhairs.job` set out options passed to SLURM during submission.

To run, you will need to change the following lines in `runhairs.job`:

- Line 15: the number of hairs in the array you wish to run
- Line 18: the path to the top directory of the entcode project
- Line 24: any modules that must be loaded to run IBAMR

After altering these, you can run any number in a series of simulations at one time using `setrunhairsjobs.sh`. In command-line bash, simply provide the start number as the first number after the shell script and the end number as the second. In this example, simulations 1 through 20 are started using sbatch:

```
$ sh setrunhairsjobs.sh 1 20
```

When these runs have finished successfully, the viz_IB2d files are automatically zipped and will be deposited in `results/runs`. Log files will be located in `results/log-files`. It is recommended that you move completed simulation directory `results/ibamr/runs` to `results/ibamr/nhair_runs` (where n is the number of hairs in the array) in keeping with the structure of the project.

4. Execution of VisIt data analysis

In order to make calculations on flow speed, we need to extract velocities from the data generated by IBAMR. The extraction will take place through VisIt (using a python script). In this script, VisIt will draw a line through the end hairs of the array and sample velocities along that line. For an example, in a 12-hair array, we would want to sample along the first row (hair 3 to hair 2), the second row (hair 7 to hair 6), and the third row (hair 8 to hair 12).

Checking for simulation set completeness

All simulations must be complete before running data analysis in VisIt. To check for completeness of the simulations, run the shell script `checkruns.sh` in command-line bash using the top-level directory as the first argument, the number of hairs in the array as the second, and the number of simulations as the third:

```
$ sh checkruns.sh "$WD" 25 165
```

This script will return "Simulation found" if it exists, is unzipped, and is complete. If these conditions are not met, it will return the number of the simulation missing.

Generating lineout files

Fluid speeds need to be sampled along a line that intersects all hairs in a row. In order for VisIt to extract fluid speeds, it needs to know where each hair exists in the doman for each array configuration in each simulation. Then these values need to be put in a standard form so that VisIt can read them automatically from the python script.

This is done by way of a standard lineout text file, which you can generate using `generate_lineouts.R` in `src/r-scripts`. Be sure to edit line 18 (endrun) with the number of simulations in the set. In the R console, run:

```
> source('generate_lineouts.R')
```

This script will read in the csv files in `results/csv-files` and deposit the lineout files into `data/lineout-files`. This only needs to be run once, as all the hairs in all the arrays conform to the same standards.

Running VisIt python scripts

To execute the python script, use the shell script `setlineout_multihair.sh` through command-line bash:

```
$ sh setlineout_multihair.sh "$WD" 25 1
```

For this example, the 25-hair array is processed for row 1. VisIt will write these values to a .curve file and store them in `results/visit/nhair_runs/simi/hairlinej` where n number of hairs in the array, i is the simulation number, and j is the row number.

This script needs to be executed independently for each row in the array.