To Do List and Open Issues

Javier Villarreal

1 Active Tasks

1.1 Initialization

The first major step in the actual code will now be to initialize all the variables. The steps to accomplish this are:

1. [**Complete**] Allocate the arrays, dimensions should now be read in
2. [**Complete**] Set all arrays equal to zero (**NOTE:** Never assume that allocation will default variables to zero, as it is not guaranteed behavior)
3. Set initial conditions by either:
   - [**Complete**] Static initial conditions throughout the domain; or
   - [**Postponed**] Read in output from prior run.
4. Enforce boundary conditions
5. Calculate derivatives

Update: Reading in initial conditions from previous run is being saved until later, after the code's output is determined, since at this time, it is not fully determined what that output may look like.

1.2 Boundary Condition Enforcement

Under the C++ code's OOP architecture, boundary enforcement was divided between enforcement on the surface boundary (cylinder, airfoil, etc), and the farfield boundaries. The Fortran code shouldn't need such a distintion (I hope), so all the boundary condition subroutines will be in one place.

Here is the general layout of boundary condition algorithms used in the C++ code:

- "Internal Boundary" (surface)
  - Set velocity to zero if viscous flow (no-slip condition)
  - Set density and energy at ghost nodes equal to one-off nodes
  - Set velocities at ghost nodes equal to velocities at one-off nodes if vicsous; and
  - Use slip matrix to set velocities at ghost nodes if inviscid.
- "External Boundary" (farfield)
  - Either set variable to predetermined farfield value; or
  - Extrapolate from nearby values, depending on whether flow is sub- or supersonic and it's an inlet or outlet.

2 Future Improvements

2.1 Build Options

Improvement could be made to the makefile (or a different build software) for having different builds in separate directories for debugg and release versions. There are also some softwares that automatically figure out dependencies, so they don't have to be explicitly stated in the makefile.

2.2 Pre-processing/Data structure

The way the data is stored in the input text files and in variables within the code is suboptimal. Changes could be made both within pre-processing (i.e. the Matlab codes that generate those files) and how the variables are declared in Fortran.

Additionally, the data files containing index "directories" for domain and boundary nodes (i.e. Jd, Jb, Jf.txt) are created by Matlab using scientific notation, and thus have to be read as real numbers. The code would be cleaner if these variables were created as integers.

2.3 OS Independence

During setup, the code sends a shell command to the system to create an output directory if it doesn't already exist. The consequence is that the code is thus Linux (and possible Mac) exclusive. It might be beneficial if the code could be independent of operating system, so it could also run on Windows. Perhaps there is some way to detect OS and use the `system` subroutine within Fortran accordingly, but this has not been looked into yet. Very low priority.

2.4 Initialization

Most of the variable initializations are (and will be) defined in the config.f90 file. However, due to the migration from the C++ code, the Mach number, angle of attack, and Reynolds number are all still read from the Cylinder data files (SimulationValues.txt). It will be a small, and quick change to retire the text file in favor of config.f90, and should be one of the first things to go once the main porting is complete.

3 Completed Tasks

3.1 Build Options

The code is built with a basic Makefile. Compilation flags are hardcoded, so a `make clean` is needed anytime flags are changed.

3.2 Read in simulation data

The data files necessary to make the code run are different depending on whether the code is running a 2- or 3-dimensional problem. For 2D, geometries, the metadata files are:

- SimulationValues.txt (Mach, AOA, Re)
- Sizes.txt (# of domain, body, farfield, cloud, ghost, extrapolation, total nodes)

and the geometry data files are:

- x,y.txt (node coordinates)
- DX,DY.txt (DQ coefficients)
- EC.txt (extrapolation coefficients)
- Jd,Jb,Jf.txt (domain, body, farfield node indices)
- nxb,nyb.txt (body node unit normal vectors)
- nxf,nyf.txt (farfield node unit normal vectors)
- s11,s12,s21,s22.txt (Flow tangency matrices)

3.3 Create data architecture

The most important flow variables and their derivatives are stored as arrays. This is different from the way it was written in the C++ code, but it should allow for a more streamlines architecture and faster access of data. A slightly more in-depth explanation will be found in the documentation notes.

3.4 Ensure output directory exists

If the output directory does not exist before the code is run, previously the code would crash, as it would be unable to find it. The current solution is to include a line in Fortran during the initial data input to send a `mkdir` to the system to create the appropriate folder.

One constraint from this implementation is that the code runs exclusively on Linux (and possible Mac), since it's a sh command sent to the system.

3.5 Initialization

The initialization specific subroutines have been completed. This refers to creating, allocating, and initializing the variables array. Initial conditions are specified through the configuration file.