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. [**Complete**] 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 Numerical Differentiation

Since the boundary conditions are enforced directly onto discrete points, the approximation of the solution is no longer a smooth function, so differentiation must be performed numerically. The C++ code performs differentiation in three steps:

1. Derivatives throughout the domain.
2. Derivatives at the surface boundary.
3. Derivatives on the farfield.

Differentiation is carried out using Differential Quadrature (DQ). The C++ code only calculates derivatives of the flow variables (density, velocity, energy). One "problem" with the C++ code is that viscous flows requires second derivatives, but second order DQ is not used. Instead, the derivatives of the derivatives are calculated. As a consequence, the derivatives at ghost points need to be known. The C++ code calculates derivatives at the ghost points, but it needs to be seriously examined, because to my knowledge it may not be based on any literature or physical property. For reference, meshed FD methods, for instance, don't need derivatives at ghost points, because second derivatives aren't calculated as "derivatives of derivatives". Rather, there is a separate formulation for second derivatives.

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.

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.

2.5 Boundary Condition Enforcement

Significant improvements could be made to boundary condition enforcement, so this is just an overview of some major points.

The major cause of problems currently is with the extrapolation function, but more broadly, the farfield boundary conditions in general. The current extrapolation algorithm is basically built out of an *interpolation* algorithm, but applied outside the cloud of nodes. This can sometimes lead to oscillations developing in the farfield, which propagate into the simulation and negatively impact results. More broadly, the boundary conditions being enforced in the farfield could likely be revisited and reformulated to give a better approximation of the physical equations. Pure Dirichlet enforcement and extrapolation may not be good enough.

The boundary conditions at the surface might be acceptable, though some process might need to be put in place during pre-processing to create a structured grid of nodes near surfaces for the purposes of differentiation. Particuarly for viscous flow problems, higher order differentiation amplifies errors in the boundary conditions.

Overall, the physcial and numerical underlying assumptions could all be reassessed.

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.

3.6 Boundary Condition Enforcement

Boundary condition enforcement algorithms were coded identically to how they behaved on the C++ code (more documentation in doc_notes and in-depth explanation in the thesis). The major difference is that all the boundary condition algorithms were consolidated into one module, as opposed to having distinct files and structures like in the C++ code. The result is hopefully a more streamlined code.