To Do List and Open Issues

Javier Villarreal

To Do

Build Options

Before development can go fully underway, build and compilation options need to be defines in the makefile. Some options to bear in mind:

- Debug vs Release builds. (Note: If using VSCode, the different builds should be reflected in the JSON option files)
- Warning and error flags (`-Wall`, `-pedantic`, `-std=...`)
- Optimizations, `-Og` for debugging and `-O2,-O3` for release.

Update 10/21/21

Creating both debug and release builds in different directories proved to be less straightforward than intended with Make, so instead there will only be the one build and flags will be specified in the makefile. This means that `make clean` will have to be run any time the flags are changed in order to recompile everything. Additionally, dependencies have to be explicitly specified. It is possible to come back and improve on the makefile to be more exhaustive, but it was not worth it at this time, while the code structure is still simple.

Read in simulation metadata

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

- SimulationValues.txt
  - Mach number
  - Angle of Attack
  - Reynolds Number
- Sizes.txt
  - # of domain nodes (including body boundary nodes)
  - # of body boundary nodes
  - # of farfield boundary nodes
  - (Only in 3D code) # of symmetry nodes
  - # of cloud nodes per domain/body boundary node (for differential quadrature)

  - # of ghost nodes per body boundary node (for differential quadrature, should equal (cloud-1)/2)
  - # of extrapolation nodes per farfield boundary node
  - # of total nodes (= domain + farfield + ghost*body)

The current task is to write a subroutine to read those files. The subroutine should reside in a separate module, since file I/O is not inherently a SOMA-specific task.

Data architecture

Geometry and simulation data is structured a certain way in the C++ code (to be documented later). There could be room for improvement in the structure – not necessarily for performance, but for readability of the code.