Final Project
ASEN 5519: Unstructured CFD

Ryan Skinner
Due 2016/??/??

Improving Boundary Condition Stability in PHASTA

1 Introduction

2 Initial Outline of PHASTA

PHASTA begins execution at `main`, located in `phSolver/[in]compressible`, depending on which branch is desired. This function initializes MPI, and then calls `phasta`, located in `/phSolver/common`. Here, inputs are read and computed in `input`, and then the solver is run by calling `proces`, a Fortran routine. Within `proces`, `gendat` generates geometry and BC data.

Routines followed by an asterisk (*) are outlined in further detail separately.

INCOMPRESSIBLE ONLY, and we ignore cardiovascular impedance and RCR boundary stuff.

- `main`
  - initialize MPI
  - `phasta`
    - initialize PETSc
    - set input data paths
    - `input` — populate data structures with problem set-up and solver parameters
      - `readnblk` — read and blocks data
        - read `numstart.dat` and finds appropriate `restart.dat` files
        - read geometry from Posix or SyncIO files using `phio_readheader`
        - calculate maximum number of boundary element nodes
        - initialize constants like `ndof`, `ndofBC`, `ndiBCB`, and `ndBCB`
        - `genblk` — read and block connectivity
        - read BC mapping array into `nBC`
        - read temporary boundary condition code into `iBCtmp`
        - read BC data into `BCinp`
        - read periodic BC data into `iperread`
        - `genbkb` — generate boundary element blocks and traces for gather/scatter operations
        - read restart data into diffusive flux vector `qold`, primitive variables `uold`, and accelerations `acold`
      - echo global information
      - assert valid input constants (e.g. `icoord`, `navier`, `iexec`) defined in `common.h`
      - echo solver and integration information
      - `genint` — generate integration information
      - estimate number of nonzero globals
      - compute fluid thermodynamic properties
    - `proces` — generate problem data and calls the solution driver

      - `gendat` — generate geometry and BC data
        - `getshp` — generate the interior nodal mapping
        - `geniBC` — generate boundary condition codes
        - `genBC` — generate the essential boundary conditions
        - work with Dirichlet-to-Neumann BCs (?)
        - `genshpb` — generate boundary element shape functions
        - `genini` — read initial values in primitive ($\underline{U}$) form, satisfies BCs, and converts to $\underline{Y}$ form, filling the `y` vector
      - `setper` and `perprep` — store inverse of sum of one and number of slaves in `rcount`
      - LES-specific routines `keeplhsG` and `setrls` called as needed
      - `initStats` — allocate arrays to store flow statistics
      - RANS-specific routine `initTurb`
      - cardiovascular-specific routine `initSponge`
      - adjust BCs to interpolate from file `inlet.dat`, if it exists
      - set up eddy-viscosity ramp specific to NGC/Duct case
      - `itrdrv`*— iterate the discrete solution using the predictor multi-corrector algorithm
    - finalize PETSc
  - finalize MPI

Numerical solution of the time-integrated unsteady Navier-Stokes equations occurs within `itrdrv`. Working arrays are listed in Table 2.

- `itrdrv`
  - `initTimeSeries` — initialize time series collection to `varts.*.dat` files using `xyzts.dat` input
  - initialize `istep` and `ifuncs(:)` to zero, and set `yold = y` and `acold = ac`
  - `initEQS` — initialize equation solver (look into this later *?)
  - `do itsq = 1, ntseq` — main loop over time sequences
    - set `itseq = itsq`
    - set iteration-specific variables for `nstep`, `niter`, `loctim`, and `deltol`
    - `itrsetup` — set up time integration parameters
      - calculate α_m, α_f, and γ as functions of ρ_∞
      - set global time increment inverse `Dtgl` and CFL data `CFLfl`
    - calculate number of flow solves per step, store in `nitr`
    - `do istp = 1, nstp` — main loop over time steps
      - `asbwmod` — set traction BCs if turbulence wall model is set (`itwmod`)
      - `itrPredict`*— predict primitive variables at time $n+1$
      - `itrBC`*— satisfy BCs on the primitive variables; returns a modified `y`
      - `itrBCSclr`*— satisfy BCs on the scalar `isclr`; returns a modified `y`
      - `do istepc = 1, seqsize` — loop over individual solves of flow and scalar
        - `icode = stepseq(istepc)` — get sequence code
        - `if` this is a flow solve
          - `SolFlow`*— perform a flow solve
        - `else if` this is a scalar solve
          - `SolSclr` — perform a scalar solve

- `else` this is an update
  - `itrCorrect`*and `itrBC`*— update flow if desired
  - `itrCorrectSclr` and `itrBCSclr` — update scalar if desired
- `stsGetStats` — obtain time averaged statistics
- find solution at end of time step and move it to old solution variables
- increment `istep` and `lstep`
- `Bflux` — compute the consistent boundary flux if desired

- deallocate variables and close files

- deallocate variables and close files

Iteration routines...

- `itrPredict` — predict solution variables at time $n+1$
  - `if (ipred .eq. 1)` — we are using same-velocity prediction, as discussed in class
    - set $\overset{n+1}{\underline{Y}}{}^{(i)} = \overset{n}{\underline{Y}}$
    - set $\overset{n+1}{\underline{Y}}{}^{(i)}_{,t} = (1 - 1/\gamma)\overset{n}{\underline{Y}}_{,t}$
  - other prediction methods (zero-acceleration, same-acceleration, and same-delta) are also supported with different values of `ipred`

Boundary conditions are set with the `iBC` and `BC` arrays. The bits of `iBC`, in increasing order, indicate whether the following BCs are set: ρ, T, p, u_1, u_2, u_3, scalars 1–4, periodicity, scaled plane extraction (SPEBC), axisymmetry, and deformable wall (for cardiovascular cases). This means for each global node, `iBC` has at least 14 bits. Note that `ibits(i,a,l)` extracts bits `a+1` through `a+l` of the integer `i`, and returns the base-10 integer. This routine is used to help identify and process boundary condition flags held in `iBC`.

- `itrBC` — satisfy BCs on the primitive variables
  - impose limits on $\underline{Y}$, using the `ylimit` data structure of dimension `(3, nflow)`, whose first index contains limit flag, lower limit, and upper limit for each flow variable.
  - velocity BCs
    -
  - pressure BCs
    -
  - local periodic BCs
  - global periodic BCs

Symbol	Dimension	Description
`nshg`		# global shape functions (`ngsh` = `nnp` if piecewise linear)
`nnp`		# global nodal points
`npro`		# elements, indexed by *e*
`nshl`		# nodes per element, indexed by *a*
`ndof`		# degrees of freedom, including scalars for turbulence models
`nflow`		# flow variables (4 incompressible, 5 compressible)
`ntseq`		# time sequences (?)
`nstep`		# time steps requested for current run
`lstep`		current time step
`lstep0`		first time step solved by current run, initialized to `lstep+1`
`istep`		step number relative to start of run
`iter`		iteration number
`niter`	`(MAXTS)`	# multi-corrector iterations per time step
`loctim`	`(MAXTS)`	local time stepping flag (?)
`deltol`	`(MAXTS, 2)`	velocity and pressure delta ratios
`impl`	`(MAXTS)`	heat, flow, and scalar solver flags (1's, 10's and 100's places)
`iturb`		indicates which turbulence model to use
`ifunc`		function evaluation counter, `niter*(lstep-lstep0)+iter`
`ifuncs`	`(6)`	function evaluation counter (?)
`y`	`(nshg, ndof)`	$\underline{Y}$ variables
`x`	`(nshg, nsd)`	node coordinates
`iBC`	`(nshg)`	BC codes
`BC`	`(nshg, ndofBC)`	BC constraint parameters
`shp`	`(nshape, ngauss)`	element shape functions at Gauss points (interior)
`shb`	`(nshapeb, ngaussb)`	element shape functions at Gauss points (boundary)
`shgl`	`(nsd, nshape, ngauss)`	local shape function gradients at Gauss points (interior)
`shglb`	`(nsd, nshapeb, nguassb)`	local shape function gradients at Gauss points (boundary)
`iper`	`(nshg)`	periodicity table

3 Life's Persistent PHASTA Questions

- Is `qold`, allocated in `readnblk.f` ever deallocated? Can't find it.
- Why do most of the time step parameters have dimension `MAXTS`?
  - It also seems that some parameters are indexed by `itseq`, but don't change from step to step.
- When is it the case that `ndof` $\neq$ `nflow`? For example during its limit-imposing stage, `itrbc` loops over `nflow` when indexing `y`'s dimension of size `ndof`.