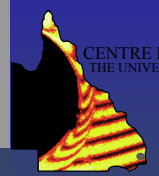
MECH4480 CFD

Eilmer3 Part I

Fabian Zander (f.zander@uq.edu.au)

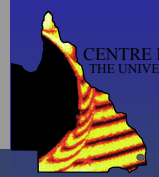
26th April 2013



CENTRE FOR HYPERSONICS
THE UNIVERSITY OF QUEENSLAND

Eilmer3 Structure

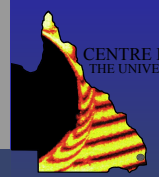
- *e3prep.py*
  - You should already be familiar with this from Herr Dr. Jahn's grid generation lectures
  - Now need to extend the options and settings configured in your input script
- *e3shared.exe*
  - The 'running' program. This executes the simulation
- *e3mpi.exe*
  - Multiprocessor 'running' program
  - Not required for this course
- *e3post.py*
  - You should also be familiar with this
  - The post-processing program to extract all of the important data
  - You will need to know lots of the options from here - refer to Eilmer3 User Guide



Configuring a Simulation

The PJ Setup

1. Set up some general parameters
2. Configure the gas model and flow states
3. Create the grid
4. Initialise the boundary conditions
5. Simulation control parameters
6. Creating a sketch



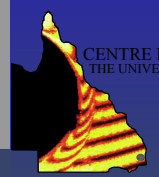
1. Initial General Parameters - Geometry

Within the *job.py* script

- `gdata.title = 'name'`
- `gdata.dimensions = 2 (or 3)`
- `gdata.axisymmetric_flag = 0 (or 1)`

GDATA OBJECT

- Python based 'object'
- Global data object for the simulation
- Allows you to set simulation parameters



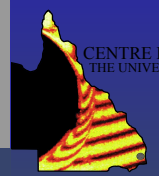
2. Gas Models and Flow States

Gas Models

- `select_gas_model(model='ideal gas', species=['air'])`

MODEL

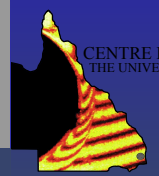
- 'ideal gas'
  - Perfect elastic collisions and constant specific heats
- 'thermally perfect gas'
  - Perfect elastic collisions but specific heat a function of temperature
- 'two temperature gas'
  - Thermally perfect gas with temperature composed of a translation/rotation and a vibration/electronic component
- 'real gas Bender'
- 'real gas MBWR'
- 'real gas REFPROP'



SPECIES

- This defines 'what' is in the flow field
- 'air' is a standard definition
- Otherwise can select any gas species available
- Available species can be found in $HOME/e3bin/species
- This can be added to as required

- Air
  - `species = ['air']`
- Dissociating Nitrogen
  - `species = ['N2', 'N']`
- Hydrogen Combustion
  - `species = ['O2', 'N2', 'H2', 'O', 'H', 'H2O', 'OH', 'HO2']`



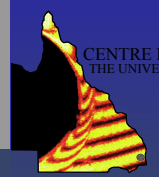
Extra options

Reaction Schemes

- This is also where you can specify a reaction model
- `set_reaction_scheme('scheme_file.lua', reacting_flag=1)`
- More about this later

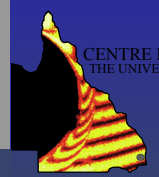
Energy Exchange

- For two-temperature models (or greater) can specify energy exchange mechanism
- `set_energy_exchange_update('exchange_file.lua')`
- Not going to be discussing this any further



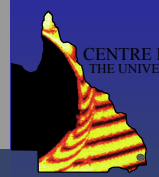
Flow States

- Eilmer uses flow 'objects' to specify different flow states
- These can be initialised in various ways
- `flow1 = FlowCondition(p=5955.0, u=0.0, v=0.0, T=304.0)`
  - Can also add extra values: `massf, tke, omega`
  - `massf` must match the species specified earlier
- `flow2 = ExistingSolution('old_sln', ".", 2, 1, 2)`
  - (rootName, solutionWorkDir, nblock, tindx, dimensions=2)



Block Creation

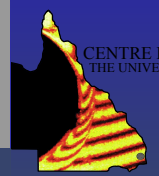
- `blk = Block2D(psurf=None, nni=2, nnj=2, cf_list=[None,]*4, bc_list=[SlipWallBC(),]*4, fill_condition=None, hcell_list=[], xforce_list=[0,]*4, label="", active=1)`
- I'm assuming you know most of this already from earlier lectures on grid generation
- Nodes → Lines → Surfaces
- Surfaces
  - Usually constructed using `make_patch(N,E,S,W)`
  - Other options also exsist; `CoonsPatch(S,N,W,E)`, `AOPatch(S,N,W,E)`
- nni/nnj = i/j discretisation
- cf_list = list of cluster functions; N, E, S, W
  - Have you done cell clustering in Eilmer3?



Block Creation

- bc_list = list of boundary conditions (if specifying this way)
- fill_condition = a flow state as discussed earlier
- hcell_list and xforce_list not required at this stage
- label = for identification in your 'sketches'
- active = 1; default is 1, i.e. on

- There are also extra options available
- Check the user guide for full details

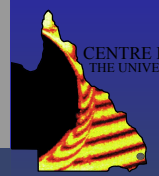


SuperBlocks

Two different options

- `my_block_list = SuperBlock2D(psurf=None, nni=2, nnj=2, nbi=1, nbj=1, cf_list=[None,]*4, bc_list=[SlipWallBC(),]*4, fill_condition=None, hcell_list=[], label="sblk")`
- `my block list = MultiBlock2D(psurf=None, nni=None, nnj=None, bc list=[SlipWallBC(),]*4, nb_w2e=1, nb_s2n=1, nn_w2e=None, nn_s2n=None, cluster_w2e=None, cluster_s2n=None, fill_condition=None, label="blk")`

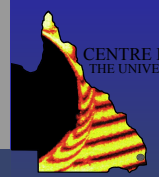
Full details in user guide



BCs Available

There are many 'standard' BCs in Eilmer

1. Block-to-block connections; `identify_block_connections()`
   - This automatically joins blocks - same as specifying `AdjacentBC()`
2. Common BCs used
   - `SupInBC(flow_state)`, `ExtrapolateOutBC()`, `SlipWallBC()`, `FixedTBC(Twall)`
3. User Defined Boundary Condition (UDF)
   - A lua script can be used to create customised BCs if desired

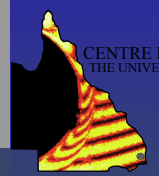


Specifying BCs

Multiple methods of defining BCs

1. Specify a bc_list during the block creation
   - If doing this you must specify a list of BCs in order (N,E,S,W)
2. `blk_0.bc_list[WEST] = SupInBC(inflow, label="inflow-boundary")`
3. `blk_1.set_BC(EAST, EXTRAPOLATE_OUT, label="outflow-boundary")`

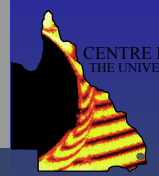
Always worth checking the sketch for BC definitions with the --do-svg option in e3prep.py (see Section 6)



Simulation Control Parameters

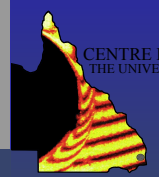
Need to specify simulation details

- `gdata.max_time = n`
  - Simulation time - abs. min. 3 flow lengths
- `gdata.max_step = n`
  - Max. number of steps - make this big enough!
- `gdata.dt = n`
  - Initial time step - if you get a CFL error you can try reducing this. This does increase the computation time though.
- `gdata.dt_plot = n`
  - How often to write out data files (in simulation time)
- `gdata.dt_history = n`
  - The time stepping to write out history data points (if specified)



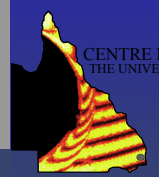
Extra Settings

- `gdata.sequence_blocks = 1`
  - Active the block marching solver, caution required when using this...no 'data' can go upstream
- `gdata.viscous_flag = 1`
  - Activate viscous terms
- `gdata.turbulence_flag = 1`
  - Activate turbulence
- `gdata.turbulence_model = 'name'`
  - Name options are "k_omega" and "baldwin_lomax"
- `cfl_count = n`
  - Number of time steps between CFL checks. Default is 10, can be reduced if you are getting stability problems



2D sketch of model

- `sketch.window(xmin=0.0, ymin=0.0, xmax=1.0, ymax=1.0, page xmin=0.05, page ymin=0.05, page xmax=0.17, page ymax=0.17)`
  - This sets up the 'page' used for the sketch
- `sketch.xaxis(x0, x1, xtic, y offset)`
- `sketch.yaxis(y0, y1, ytic, x offset)`
  - Configure the axes
- See the user guide for full details (Section I.11)

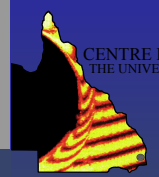


Running Eilmer3

Easy from the command line

- Prepare the job
  - `>> e3prep.py --job=name --do-svg`
- Run the job
  - `>> e3shared.exe --job=name --run`
- Post-processing
  - `>> e3post.py --job=name --tindx=all --vtk-xml`

- The Eilmer3 examples often have shell scripts to do these things. This is fine, but you have to have a good grasp of what they are doing, in particular `e3post.py`
- There are plenty more options!



Tutorial

cone20-simple

- Start by copying the cone20-simple folder from */cfcfd3/examples/eilmer3/2D/* to your working directory
- Have a look at the *cone20.py* script and identify some of the settings I've been talking about
- Change the inflow velocity to 6000m/s
- What will this new velocity change in the simulation?
  - Think about mesh, `max_time` (how many flow lengths), `dt` (CFL problem?), are viscous effects required?
- There are run scripts there - but make sure you can also run the programs from the command line
- When you finish this, have a look around at the other examples