FUN3D Analysis Interface Module (AIM)

Ryan Durscher
AFRL/RQVC

January 23, 2020

Contents

1 Introduction

1.1 FUN3D AIM Overview

A module in the Computational Aircraft Prototype Syntheses (CAPS) has been developed to interact (primarily through input files) with NASA LaRC's unstructured flow solver FUN3D [1]. FUN3D is a parallelized flow analysis and design suite capable of addressing a wide variety of complex aerodynamic configurations by utilizing a mixed-element, node-based, finite volume discretization. The suite can simulate perfect gas (both incompressible and compressible), as well as multi-species equilibrium and non-equilibrium flows. Turbulence effects may be represented through a wide variety of models. Currently only a subset of FUN3D's input options have been exposed in the analysis interface module (AIM), but features can easily be included as future needs arise.

Current issues include:

- A thorough bug testing needs to be undertaken.

- Not all parameters/variables in fun3d.nml are currently available.

An outline of the AIM's inputs and outputs are provided in AIM Inputs and AIM Outputs, respectively.

Details of the AIM's shareable data structures are outlined in AIM Shareable Data if connecting this AIM to other AIMs in a parent-child like manner.

Details of the AIM's automated data transfer capabilities are outlined in FUN3D Data Transfer

1.2 Generating fun3d.nml

FUN3D's primarily input file is a master FORTRAN namelist, fun3d.nml. To generate a bare-bones fun3d.nml file based on the variables set in AIM Inputs, nothing else besides the AIM needs to be provided. Since this will create a new fun3d.nml file every time the AIM is executed it is essential to set the Overwrite_NML input variable to "True". This gives the user ample warning that their fun3d.nml (if it exists) will be over written.

Conversely, to read and append an existing namelist file the user needs Python installed so that the AIM may be complied against the Python API library (and header file - Python.h). The AIM interacts with Python through a Cython linked script that utilizes the "f90nml" Python module; note, having Cython installed is not required. On systems with "pip" installed typing "pip install f90nml", will download and install the "f90nml" module.

The Cython script will first try to read an existing fun3d.nml file in the specified analysis directory; if the file does not exist one will be created. Only modified input variables that have been specified as AIM inputs (currently supported variables are outlined in AIM Inputs) are updated in the namelist file.

1.3 Examples

An example problem using the FUN3D AIM (coupled with a meshing AIM - TetGen) may be found at FUN3D AIM Example.

2 AIM Inputs

The following list outlines the FUN3D inputs along with their default values available through the AIM interface. One will note most of the FUN3D parameters have a NULL value as their default. This is done since a parameter in the FUN3D input deck (fun3d.nml) is only changed if the value has been changed in CAPS (i.e. set to something other than NULL).

- **Proj_Name = "fun3d_CAPS"**
  This corresponds to the project_rootname variable in the &project namelist of fun3d.nml.

- **Mach = NULL**
  This corresponds to the mach_number variable in the &reference_physical_properties namelist of fun3d.nml.

- **Re = NULL**
  This corresponds to the reynolds_number variable in the &reference_physical_properties namelist of fun3d.↩
  nml.

- **Viscous = NULL**
  This corresponds to the viscous_terms variable in the &governing_equation namelist of fun3d.nml.

- **Equation_Type = NULL**
  This corresponds to the eqn_type variable in the &governing_equation namelist of fun3d.nml.

- **Alpha = NULL**
  This corresponds to the angle_of_attack variable in the &reference_physical_properties namelist of fun3d.nml
  [degree].

- **Beta = NULL**
  This corresponds to the angle_of_yaw variable in the &reference_physical_properties namelist of fun3d.nml
  [degree].

- **Overwrite_NML = NULL**

  - If Python is NOT linked with the FUN3D AIM at compile time or Use_Python_NML is set to False this flag
    gives the AIM permission to overwrite fun3d.nml if present. The namelist produced will solely consist of
    input variables present and set in the AIM.

  - If Python IS linked with the FUN3D AIM at compile time and Use_Python_NML is set to True the namelist
    file will be overwritten, as opposed to being appended.

- **Mesh_Format = "AFLR3"**
  Mesh output format. By default, an AFLR3 mesh will be used.

- **Mesh_ASCII_Flag = True**
  Output mesh in ASCII format, otherwise write a binary file if applicable.

- **Num_Iter = NULL**
  This corresponds to the steps variable in the &code_run_control namelist of fun3d.nml.

- **CFL_Schedule = NULL**
  This corresponds to the schedule_cfl variable in the &nonlinear_solver_parameters namelist of fun3d.nml.

- **CFL_Schedule_Inter = NULL**
  This corresponds to the schedule_iteration variable in the &nonlinear_solver_parameters namelist of fun3d.↩
  nml.

- **Restart_Read = NULL**
  This corresponds to the restart_read variable in the &code_run_control namelist of fun3d.nml.

- **Boundary_Condition = NULL**
  See CFD Boundary Conditions for additional details.

- **Use_Python_NML = False**
  By default, even if Python has been linked to the FUN3D AIM it is not used unless the this value is set to True.

- **Pressure_Scale_Factor = 1.0**
  Value to scale Cp data when transferring data. Data is scaled based on Pressure = Pressure_Scale_↩
  Factor∗Cp + Pressure_Scale_Offset.

- **Pressure_Scale_Offset = 0.0**
  Value to offset Cp data when transferring data. Data is scaled based on Pressure = Pressure_Scale_↩
  Factor∗Cp + Pressure_Scale_Offset.

- **NonInertial_Rotation_Rate = NULL [0.0, 0.0, 0.0]**
  Array values correspond to the rotation_rate_x, rotation_rate_y, rotation_rate_z variables, respectively, in the
  &noninertial_reference_frame namelist of fun3d.nml.

- **NonInertial_Rotation_Center = NULL, [0.0, 0.0, 0.0]**
  Array values correspond to the rotation_center_x, rotation_center_y, rotation_center_z variables, respectively, in the &noninertial_reference_frame namelist of fun3d.nml.

- **Two_Dimensional = False**
  Run FUN3D in 2D mode. If set to True, the body must be a single "sheet" body in the x-z plane (a rudimentary node swapping routine is attempted if not in the x-z plane). A 3D mesh will be written out, where the body is extruded a length of 1 in the y-direction.

- **Modal_Aeroelastic = NULL**
  See Modal Aeroelastic Inputs for additional details.

- **Modal_Ref_Velocity = NULL**
  The freestream velocity in structural dynamics equation units; used for scaling during modal aeroelastic simulations. This corresponds to the uinf variable in the &aeroelastic_modal_data namelist of movingbody.input.

- **Modal_Ref_Length = 1.0**
  The scaling factor between CFD and the structural dynamics equation units; used for scaling during modal aeroelastic simulations. This corresponds to the grefl variable in the &aeroelastic_modal_data namelist of movingbody.input.

- **Modal_Ref_Dynamic_Pressure = NULL**
  The freestream dynamic pressure in structural dynamics equation units; used for scaling during modal aeroelastic simulations. This corresponds to the qinf variable in the &aeroelastic_modal_data namelist of movingbody.input.

- **Time_Accuracy = NULL**
  Defines the temporal scheme to use. This corresponds to the time_accuracy variable in the &nonlinear_↩ solver_parameters namelist of fun3d.nml.

- **Time_Step = NULL**
  Non-dimensional time step during time accurate simulations. This corresponds to the time_step_nondim variable in the &nonlinear_solver_parameters namelist of fun3d.nml.

- **Num_Subiter = NULL**
  Number of subiterations used during a time step in a time accurate simulations. This corresponds to the subiterations variable in the &nonlinear_solver_parameters namelist of fun3d.nml.

- **Temporal_Error = NULL**
  This sets the tolerance for which subiterations are stopped during time accurate simulations. This corresponds to the temporal_err_floor variable in the &nonlinear_solver_parameters namelist of fun3d.nml.

- **Reference_Area = NULL**
  This sets the reference area for used in force and moment calculations. This corresponds to the area_↩ reference variable in the &force_moment_integ_properties namelist of fun3d.nml. Alternatively, the geometry (body) attribute "capsReferenceArea" maybe used to specify this variable (note: values set through the AIM input will supersede the attribution value).

- **Moment_Length = NULL, [0.0, 0.0]**
  Array values correspond to the x_moment_length and y_moment_length variables, respectively, in the &force_moment_integ_properties namelist of fun3d.nml. Alternatively, the geometry (body) attributes "caps↩ ReferenceChord" and "capsReferenceSpan" may be used to specify the x- and y- moment lengths, respectively (note: values set through the AIM input will supersede the attribution values).

- **Moment_Center = NULL, [0.0, 0.0, 0.0]**
  Array values correspond to the x_moment_center, y_moment_center, and z_moment_center variables, respectively, in the &force_moment_integ_properties namelist of fun3d.nml. Alternatively, the geometry (body) attributes "capsReferenceX", "capsReferenceY", and "capsReferenceZ" may be used to specify the x-, y-, and z- moment centers, respectively (note: values set through the AIM input will supersede the attribution values).

- **FUN3D_Version = 13.1**
  FUN3D version to generate specific configuration file for; currently only has influence over rubber.data (sensitivity file).

- **Sensitivity = NULL**
  A work in progress - Currently setting this value to any string will retrieve the parameterization for all geometric design parameters.

3   AIM Shareable Data

Currently the FUN3D AIM does not have any shareable data types or values. It will try, however, to inherit a "↩ Volume_Mesh" (for 3D simulations) or a "Surface_Mesh" (for 2D simulations) from any parent AIMs. Note that the inheritance of the volume/surface mesh variable is required if the FUN3D aim is to write a suitable grid and $*$.mapbc (boundary condition file for the AFLR3 mesh format) file.

4   AIM Outputs

The following list outlines the FUN3D outputs available through the AIM interface. All variables currently correspond to values for all boundaries (total) found in the $*$.forces file

Net Forces - Pressure + Viscous:

- **CLtot** = The lift coefficient.

- **CDtot** = The drag coefficient.

- **CMXtot** = The moment coefficient about the x-axis.

- **CMYtot** = The moment coefficient about the y-axis.

- **CMZtot** = The moment coefficient about the z-axis.

- **CXtot** = The force coefficient about the x-axis.

- **CYtot** = The force coefficient about the y-axis.

- **CZtot** = The force coefficient about the z-axis.

Pressure Forces:

- **CLtot_p** = The lift coefficient - pressure contribution only.

- **CDtot_p** = The drag coefficient - pressure contribution only.

- **CMXtot_p** = The moment coefficient about the x-axis - pressure contribution only.

- **CMYtot_p** = The moment coefficient about the y-axis - pressure contribution only.

- **CMZtot_p** = The moment coefficient about the z-axis - pressure contribution only.

- **CXtot_p** = The force coefficient about the x-axis - pressure contribution only.

- **CYtot_p** = The force coefficient about the y-axis - pressure contribution only.

- **CZtot_p** = The force coefficient about the z-axis - pressure contribution only.

Viscous Forces:

- **CLtot_v** = The lift coefficient - viscous contribution only.

- **CDtot_v** = The drag coefficient - viscous contribution only.

- **CMXtot_v** = The moment coefficient about the x-axis - viscous contribution only.

- **CMYtot_v** = The moment coefficient about the y-axis - viscous contribution only.

- **CMZtot_v** = The moment coefficient about the z-axis - viscous contribution only.

- **CXtot_v** = The force coefficient about the x-axis - viscous contribution only.

- **CYtot_v** = The force coefficient about the y-axis - viscous contribution only.

- **CZtot_v** = The force coefficient about the z-axis - viscous contribution only.

Force components:

- **Forces** = Returns a tuple array of JSON string dictionaries of forces and moments for each boundary (combined forces also included). The structure for the Forces tuple = ("Boundary Name", "Value"). "Boundary Name" defines the boundary/component name (or "Total") and the "Value" is a JSON string dictionary. Entries in the dictionary are the same as the other output variables without "tot" in the name (e.g. CL, CD, CMX, CL_p, CMX_v, etc.).

5 data sets consumed by FUN3D

This function checks if a data set name can be consumed by this aim. The FUN3D aim can consume "Displacement" and EigenValue data sets for areolastic analysis.

6 FUN3D Data Transfer

The FUN3D AIM has the ability to transfer surface data (e.g. pressure distributions) to and from the AIM using the conservative and interpolative data transfer schemes in CAPS. Currently these transfers may only take place on triangular meshes.

6.1 Data transfer from FUN3D

- **"Pressure", "P", "Cp", or "CoefficientOfPressure"**
  Loads the coefficient of pressure distribution from [project_name]_ddfdrive_bndry[#].dat file(s) (as generate from a FUN3D command line option of --write_aero_loads_to_file) into the data transfer scheme. This distribution may be scaled based on Pressure = Pressure_Scale_Factor∗Cp + Pressure_Scale_Offset, where "Pressure_Scale_Factor" and "Pressure_Scale_Offset" are AIM inputs (AIM Inputs).

6.2 Data transfer to FUN3D

- **"Displacement"**
  Retrieves nodal displacements (as from a structural solver) and updates FUN3D's surface mesh; a new [project_name]_body1.dat file is written out which may be loaded into FUN3D to update the surface mesh/move the volume mesh using the FUN3D command line option --read_surface_from_file

- **"EigenVector_#"**
  Retrieves modal eigen-vectors from a structural solver, where "#" should be replaced by the corresponding mode number for the eigen-vector (eg. EigenVector_3 would correspond to the third mode, while Eigen↩Vector_6 would be the sixth mode) . A [project_name]_body1_mode#.dat file is written out for each mode.

7 CFD Boundary Conditions

Structure for the boundary condition tuple = ("CAPS Group Name", "Value"). "CAPS Group Name" defines the capsGroup on which the boundary condition should be applied. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword string (see Section Single Value String)

7.1   JSON String Dictionary

If "Value" is a JSON string dictionary (eg. "Value" = {"bcType": "Viscous", "wallTemperature": 1.1}) the following keywords (= default values) may be used:

- **bcType = "Inviscid"**
  Boundary condition type. Options:

    - Inviscid

    - Viscous

    - Farfield

    - Extrapolate

    - Freestream

    - BackPressure

    - Symmetry

    - SubsonicInflow

    - SubsonicOutflow

    - MassflowIn

    - MassflowOut

    - MachOutflow

7.1.1   Wall Properties

- **wallTemperature = 0.0**
  The ratio of wall temperature to reference temperature for inviscid and viscous surfaces. Adiabatic wall = -1

7.1.2   Stagnation Properties

- **totalPressure = 0.0**
  Ratio of total pressure to reference pressure on a boundary surface.

- **totalTemperature = 0.0**
  Ratio of total temperature to reference temperature on a boundary surface.

7.1.3   Static Properties

- **staticPressure = 0.0**
  Ratio of static pressure to reference pressure on a boundary surface.

7.1.4   Velocity Components

- **machNumber = 0.0**
  Mach number on boundary.

7.1.5   Massflow Properties

- **massflow = 0.0**
  Massflow through the boundary in units of grid units squared.

7.2 Single Value String

If "Value" is a single string the following options maybe used:

- "Inviscid" (default)

- "Viscous"

- "Farfield"

- "Extrapolate"

- "Freestream"

- "SymmetryX"

- "SymmetryY"

- "SymmetryZ"

8 Modal Aeroelastic Inputs

Structure for the modal aeroelastic tuple = ("EigenVector_#", "Value"). The tuple name "EigenVector_#" defines the eigen-vector in which the supplied information corresponds to, where "#" should be replaced by the corresponding mode number for the eigen-vector (eg. EigenVector_3 would correspond to the third mode, while EigenVector_6 would be the sixth mode). This notation is the same as found in FUN3D Data Transfer. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

8.1 JSON String Dictionary

If "Value" is a JSON string dictionary (eg. "Value" = {"generalMass": 1.0, "frequency": 10.7}) the following keywords (= default values) may be used:

- **frequency = 0.0**
  This is the frequency of specified mode, in rad/sec.

- **damping = 0.0**
  The critical damping ratio of the mode.

- **generalMass = 0.0**
  The generalized mass of the mode.

- **generalDisplacement = 0.0**
  The generalized displacement used at the starting time step to perturb the mode and excite a dynamic response.

- **generalVelocity = 0.0**
  The generalized velocity used at the starting time step to perturb the mode and excite a dynamic response.

- **generalForce = 0.0**
  The generalized force used at the starting time step to perturb the mode and excite a dynamic response.

9 FUN3D AIM Example

This is a walkthrough for using FUN3D AIM to analyze a three-dimensional two-wing configuration.

9.1 Prerequisites

It is presumed that ESP and CAPS have been already installed, as well as FUN3D. In this example the open-source, tetrahedral mesh generator, TetGen, is coupled to the FUN3D AIM to provide a volumetric mesh.

9.1.1 Script files

Two scripts are used for this illustration:

1. cfdMultiBody.csm: Creates geometry, as described in following section.

2. fun3d_and_Tetgen_PyTest.py: pyCAPS script for performing analysis, as described in Performing analysis using pyCAPS.

9.2 Creating Geometry using ESP

The CSM script generates Bodies which are designed to be used by specific AIMs. The AIMs that the Body is designed for is communicated to the CAPS framework via the "capsAIM" string attribute. This is a semicolon-separated string with the list of AIM names. Thus, the CSM author can give a clear indication to which AIMs should use the Body. In this example, the list contains the list of mesh generators and CFD solvers that can consume the body:

```
attribute capsAIM $fun3dAIM;su2AIM;egadsTessAIM;aflr4AIM;pointwiseAIM;tetgenAIM;aflr3AIM #CFD Analysis
```

A typical geometry model can be created and interactively modified using design parameters. These design parameters are either design- or geometry- based. In this example, a two wing configuration is created using following design parameters,

```
despmtr   area[1,1]        40.00000
despmtr   aspect[1,1]      5.00000
despmtr   taper[1,1]       0.50000
despmtr   twist[1,1]       15.00000
despmtr   lesweep[1,1]     30.00000
despmtr   dihedral[1,1]    1.00000
```

as well as the following configuration paramters. Configuration quantities cannot be used with sensitivities.

```
cfgpmtr   series           8412
cfgpmtr   series2          0020
cfgpmtr   sharpte          0
```

Next, internal CAPS reference attributes are set.

```
# Set reference values
attribute capsReferenceArea      area
attribute capsReferenceChord     sqrt(area/aspect)
attribute capsReferenceSpan      sqrt(area/aspect)*aspect
```

After our design parameters are defined they are used to setup other local variables (analytically) for the wing.

```
set       cmean     sqrt(area/aspect)
set       span      cmean*aspect
set       sspan     span/2
set       croot     2*cmean/(1+taper)
set       ctip      croot*taper
set       xtip      sspan*tand(lesweep)
set       ytip      sspan*tand(dihedral)
set       ybot      -0.1*croot
set       ytop      +0.2*croot+ytip
set       extend    0.02*cmean
```

Once all design and locale variables are defined, a half span, solid model is created by "ruling" together NACA series airfoils (following a series of scales, rotations, and translations).

```
mark
   udprim   naca   Series    series     sharpte sharpte
      scale      croot
   udprim   naca   Series    series2    sharpte sharpte
      scale      ctip
      rotatez   -twist   0    0
      translate xtip   ytip   -sspan
rule      0
```

A full span model is then created by mirroring and joining the half-span model.

```
# Store half of wing
store     HalfWing 0 0

# Restore the half wing
restore   HalfWing 0

# Restore and mirror the half wing
restore   HalfWing 0
   mirror   0   0   1   0


# Combine halfs into a whole
join     0
```

Once the desired model obtained it needs to be rotated so that it is in the expected aero-coordinated system (y- out the right wing, x- in the flow direction, and +z- up).

```
# Get body into a typical aero-system
rotatex 90 0 0
```

Next, an attribute is then placed in the geometry so that the geometry components may be reference by the FUN3D AIM.

```
# Store the wing
store     Wing 0 0

# Wing 1 - Restore
restore   Wing 0
   attribute capsGroup $Wing1
   attribute _name $Wing1
   attribute AFLR4_Cmp_ID 1
   attribute AFLR4_Edge_Scale_Factor_Weight 1
```

Following the completion of the first wing, a second wing is created and scaled using the store/restore operations.

```
# Wing 2 -  Restore and scale, translate
restore   Wing 0
   attribute capsGroup          $Wing2
   attribute _name              $Wing2
   attribute AFLR4_Scale_Factor 10
   attribute AFLR4_Cmp_ID       2

   scale    0.4
   translate 10   0   0
```

Finally, for three-dimensional CFD analysis with the FUN3D AIM a "farfield" or "bounding box" boundary needs to be provided. In this example, a simple sphere is created and tagged as a farfield boundary using the capsGroup attribute.

```
sphere    0   0   0   80
   attribute capsGroup    $Farfield
   attribute _name        $Farfield
   attribute AFLR_GBC     $FARFIELD_UG3_GBC
   attribute AFLR4_Cmp_ID 3
   attribute capsMeshLength  cmean #Charachteristic length for meshing
```

9.3 Performing analysis using pyCAPS

The first step in the pyCAPS script is to import the required modules. For this example, the following modules are used,

```python
from __future__ import print_function

import os
```

In order to create a new capsProblem the pyCAPS module also needs to be imported; on Linux and OSX this is the pyCAPS.so file, while on Windows it is the pyCAPS.pyd file. For convenience, it is recommended that the path to this file is added to the environmental variable PYTHONPATH.

```python
from pyCAPS import capsProblem
```

Similarly, local variables used throughout the script may be defined.

```python
workDir = os.path.join(str(args.workDir[0]), "FUN3DTetgenAnalysisTest")
```

Once the required modules have been loaded, a capsProblem can be instantiated.

```python
myProblem = capsProblem()
```

Using the loadCAPS() function, the desired geometry file is then loaded into the problem.

```python
myGeometry = myProblem.loadCAPS("../csmData/cfdMultiBody.csm", verbosity=args.verbosity)
```

Any design parameters available in ∗.csm file are also available within the pyCAPS script. The following snippet changes the despmtr "area" which will force a rebuild of the geometry that FUN3D will now use.

```python
myGeometry.setGeometryVal("area", 50)
```

A typical high-fidelity CFD analysis requires meshing AIMs to be coupled to the analysis AIM (unless a mesh already exists).For surface meshing, the face tessellation from the ESP geometry can be directly used as the surface mesh. If the face tessellation is not satisfactory, a surface meshing AIM may be coupled to the volume meshing AIM. In this example, the face tessellation is used as the surface mesh and TetGen for volumetric mesh generation. The TetGen AIM in loaded using the following

```python
meshAIM = myProblem.loadAIM(aim = "tetgenAIM",
                            analysisDir = workDir)
```

Once loaded, the appropriate inputs to the mesh generator required to generate mesh with adequate mesh quality are set. Refer TetGen AIM documentation for the list of all the available options.

```python
# Set new EGADS body tessellation parameters
meshAIM.setAnalysisVal("Tess_Params", [.05, 0.01, 20.0])

# Preserve surface mesh while meshing
meshAIM.setAnalysisVal("Preserve_Surf_Mesh", True)
```

After options are set aimPreAnalysis needs to be executed. For example,

```python
meshAIM.preAnalysis()
```

In the case of the TetGen AIM, TetGen was already ran during preAnalysis. To make the volumetric mesh available to other AIMs, aimPostAnalysis just needs to be additionally implemented.

```python
meshAIM.postAnalysis()
```

Next the FUN3D AIM needs to be loaded. In this example, the previously loaded TetGen AIM is linked as a parent. This allows the volume mesh generated by the TetGen AIM to be inherited (see AIM Shareable Data) by the FUN3D AIM, in which case FUN3D will write out the mesh in its preferred, native format.

```
# Load FUN3D aim - child of Tetgen AIM
fun3dAIM = myProblem.loadAIM(aim = "fun3dAIM",
                             altName = "fun3d",
                             analysisDir = workDir, parents = meshAIM.aimName)
```

Once loaded analysis parameters specific to FUN3D need to be set (see AIM Inputs). These parameters are automatically converted into FUN3D specific format and transferred into the FUN3D configuration file. One will note in the following snippet the instance of the AIM is referenced in two different manners: 1. Using the returned object from load call and 2. Using the "altName" name reference in the analysis dictionary. While syntactically different, these two forms are essentially identical.

```
# Set project name
fun3dAIM.setAnalysisVal("Proj_Name", "fun3dTetgenTest")

fun3dAIM.setAnalysisVal("Mesh_ASCII_Flag", False)

# Set AoA number
myProblem.analysis["fun3d"].setAnalysisVal("Alpha", 1.0)

# Set Mach number
myProblem.analysis["fun3d"].setAnalysisVal("Mach", 0.5901)

# Set equation type
fun3dAIM.setAnalysisVal("Equation_Type","compressible")

# Set Viscous term
myProblem.analysis["fun3d"].setAnalysisVal("Viscous", "inviscid")

# Set number of iterations
myProblem.analysis["fun3d"].setAnalysisVal("Num_Iter",10)

# Set CFL number schedule
myProblem.analysis["fun3d"].setAnalysisVal("CFL_Schedule",[0.5, 3.0])

# Set read restart option
fun3dAIM.setAnalysisVal("Restart_Read","off")

# Set CFL number iteration schedule
myProblem.analysis["fun3d"].setAnalysisVal("CFL_Schedule_Iter", [1, 40])

# Set overwrite fun3d.nml if not linking to Python library
myProblem.analysis["fun3d"].setAnalysisVal("Overwrite_NML", True)
```

Along the same lines of setting the other input values the "Boundary_Condition" tuple is used to set the boundary conditions (CFD Boundary Conditions). These boundary tags (which reference capsGroup attributes in the *.csm file) and associated boundary conditions are converted into FUN3D specific boundary conditions and set in the FUN3D configuration file.

```
inviscidBC1 = {"bcType" : "Inviscid", "wallTemperature" : 1}
inviscidBC2 = {"bcType" : "Inviscid", "wallTemperature" : 1.2}
fun3dAIM.setAnalysisVal("Boundary_Condition", [("Wing1", inviscidBC1),
                                               ("Wing2", inviscidBC2),
                                               ("Farfield","farfield")])
```

Again, after all desired options are set aimPreAnalysis needs to be executed.

```
fun3dAIM.preAnalysis()
```

At this point the required files necessary run FUN3D should have be created and placed in the specified analysis working directory. Next FUN3D needs to executed either through its Python interface module (not shown) or an OS system call such as,

```
print ("\n\nRunning FUN3D......")
currentDirectory = os.getcwd() # Get our current working directory

os.chdir(fun3dAIM.analysisDir) # Move into test directory

os.system("nodet_mpi --animation_freq -1 --write_aero_loads_to_file> Info.out"); # Run fun3d via system
      call

os.chdir(currentDirectory) # Move back to top directory
```

After FUN3D is finished running aimPostAnalysis needs to be executed.

```
fun3dAIM.postAnalysis()
```

Finally, available AIM outputs (see AIM Outputs) may be retrieved, for example:

```
print ("Total Force - Pressure + Viscous")
# Get Lift and Drag coefficients
print ("Cl = " , fun3dAIM.getAnalysisOutVal("CLtot"),
       "Cd = " , fun3dAIM.getAnalysisOutVal("CDtot"))

# Get Cmx, Cmy, and Cmz coefficients
print ("Cmx = " , fun3dAIM.getAnalysisOutVal("CMXtot"),
       "Cmy = " , fun3dAIM.getAnalysisOutVal("CMYtot"),
       "Cmz = " , fun3dAIM.getAnalysisOutVal("CMZtot"))

# Get Cx, Cy, Cz coefficients
print ("Cx = " , fun3dAIM.getAnalysisOutVal("CXtot"),
       "Cy = " , fun3dAIM.getAnalysisOutVal("CYtot"),
       "Cz = " , fun3dAIM.getAnalysisOutVal("CZtot"))

print ("Pressure Contribution")
# Get Lift and Drag coefficients
print ("Cl_p = " , fun3dAIM.getAnalysisOutVal("CLtot_p"),
       "Cd_p = " , fun3dAIM.getAnalysisOutVal("CDtot_p"))

# Get Cmx, Cmy, and Cmz coefficients
print ("Cmx_p = " , fun3dAIM.getAnalysisOutVal("CMXtot_p"),
       "Cmy_p = " , fun3dAIM.getAnalysisOutVal("CMYtot_p"),
       "Cmz_p = " , fun3dAIM.getAnalysisOutVal("CMZtot_p"))

# Get Cx, Cy, and Cz, coefficients
print ("Cx_p = " , fun3dAIM.getAnalysisOutVal("CXtot_p"),
       "Cy_p = " , fun3dAIM.getAnalysisOutVal("CYtot_p"),
       "Cz_p = " , fun3dAIM.getAnalysisOutVal("CZtot_p"))

print ("Viscous Contribution")
# Get Lift and Drag coefficients
print ("Cl_v = " , fun3dAIM.getAnalysisOutVal("CLtot_v"),
       "Cd_v = " , fun3dAIM.getAnalysisOutVal("CDtot_v"))

# Get Cmx, Cmy, and Cmz coefficients
print ("Cmx_v = " , fun3dAIM.getAnalysisOutVal("CMXtot_v"),
       "Cmy_v = " , fun3dAIM.getAnalysisOutVal("CMYtot_v"),
       "Cmz_v = " , fun3dAIM.getAnalysisOutVal("CMZtot_v"))

# Get Cx, Cy, and Cz, coefficients
print ("Cx_v = " , fun3dAIM.getAnalysisOutVal("CXtot_v"),
       "Cy_v = " , fun3dAIM.getAnalysisOutVal("CYtot_v"),
       "Cz_v = " , fun3dAIM.getAnalysisOutVal("CZtot_v"))
```

results in,

```
Total Force - Pressure + Viscous
Cl =   0.671595 Cd =   0.517818
Cmx =  -0.002830832 Cmy =  -1.342669 Cmz =  -0.0020397
Cx =   0.5060182 Cy =  -0.004986118 Cz =   0.6805299
Pressure Contribution
Cl_p =   0.671595 Cd_p =   0.517818
Cmx_p =  -0.002830832 Cmy_p =  -1.342669 Cmz_p =  -0.0020397
Cx_p =   0.5060182 Cy_p =  -0.004986118 Cz_p =   0.6805299
Viscous Contribution
Cl_v =   0.0 Cd_v =   0.0
Cmx_v =   0.0 Cmy_v =   0.0 Cmz_v =   0.0
Cx_v =   0.0 Cy_v =   0.0 Cz_v =   0.0
```

When finally finished with the script, the open CAPS problem should be closed.

```
myProblem.closeCAPS()
```

9.4   Executing pyCAPS script

Issuing the following command executes the script:

```
python fun3d_and_Tetgen_PyTest.py
```

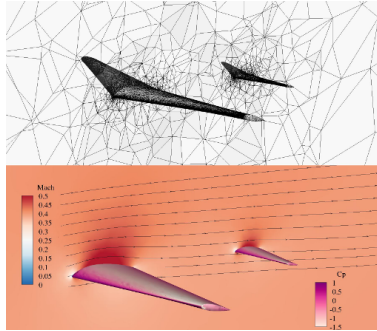Below are representative result images generated by the above script:

Figure 1: FUN3D coupled to TetGen example

References

[1] Robert T. Biedron, Jan-Renee Carlson, Joseph M. Derlaga, Peter A. Gnoffo, Dana P. Hammond, William T. Jones, Bil Kleb, Elizabeth M. Lee-Rausch, Eric J. Nielsen, Michael A. Park, Christopher L. Rumsey, James L. Thomas, and William A. Wood. *FUN3D Manual: 12.7*, May 2015. 2