

## FUN3D Analysis Interface Module (AIM)

Ryan Durscher  
AFRL/RQVC

June 8, 2022



0.1 Introduction	1
0.1.1 FUN3D AIM Overview	1
0.1.2 Generating fun3d.nml	1
0.1.3 Examples	1
0.1.4 Clearance Statement	1
0.2 AIM Units	2
0.3 AIM Inputs	2
0.4 AIM Outputs	4
0.5 FUN3D Data Transfer	6
0.5.1 Data transfer from FUN3D (FieldOut)	6
0.5.2 Data transfer to FUN3D (FieldIn)	6
0.6 CFD Boundary Conditions	6
0.6.1 JSON String Dictionary	6
0.6.1.1 Wall Properties	7
0.6.1.2 Stagnation Properties	7
0.6.1.3 Static Properties	7
0.6.1.4 Velocity Components	7
0.6.1.5 Massflow Properties	7
0.6.2 Single Value String	7
0.7 CFD Modal Aeroelastic	8
0.7.1 JSON String Dictionary	8
0.8 CFD Design Variable	8
0.8.1 JSON String Dictionary	8
0.8.2 Single Value String	9
0.9 CFD Functional	9
0.9.1 JSON String Dictionary	9
0.10 FUN3D AIM Example	10
0.10.1 Prerequisites	10
0.10.1.1 Script files	10
0.10.2 Creating Geometry using ESP	10
0.10.3 Performing analysis using pyCAPS	12
0.10.4 Executing pyCAPS script	14
Bibliography	15



## 0.1 Introduction

### 0.1.1 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:

- 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 on the use of units are outlined in [AIM Units](#).

Details of the AIM's automated data transfer capabilities are outlined in [FUN3D Data Transfer](#)

### 0.1.2 Generating fun3d.nml

FUN3D's primary 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 compiled 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.

### 0.1.3 Examples

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

### 0.1.4 Clearance Statement

This software has been cleared for public release on 05 Nov 2020, case number 88ABW-2020-3462.

## 0.2 AIM Units

A unit system may be optionally specified during AIM instance initiation. If a unit system is provided, all AIM input values which have associated units must be specified as well. If no unit system is used, AIM inputs, which otherwise would require units, will be assumed unit consistent. A unit system may be specified via a JSON string dictionary for example: `unitSys = {"temperature": "Kelvin"}`

- **temperature = "None"**  
Temperature units - e.g. "Kelvin", "Rankin" ...

## 0.3 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.
- **Temperature = NULL**  
This corresponds to the `temperature` variable in the `&reference_physical_properties` namelist of fun3d.nml. Note if no temperature units are set, units of Kelvin are assumed (see [AIM Units](#))
- **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  $\text{Pressure} = \text{Pressure\_Scale\_Factor} * \text{Cp} + \text{Pressure\_Scale\_Offset}$ .
- **Pressure\_Scale\_Offset = 0.0**  
Value to offset Cp data when transferring data. Data is scaled based on  $\text{Pressure} = \text{Pressure\_Scale\_Factor} * \text{Cp} + \text{Pressure\_Scale\_Offset}$ .
- **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 [CFD Modal Aeroelastic](#) 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_ref` reference variable in the `&force_moment_integ_properties` namelist of `fun3d.nml`. Alternatively, the geometry (body) attribute "capsReferenceArea" may be 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 "capsReferenceChord" 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) and aeroelastic modal data namelist in `moving_body.input`.
- **Design\_Variable = NULL**  
List of AnalysisIn and/or GeometryIn variable names used to compute sensitivities of Design\_Functional for optimization, see [CFD Design Variable](#) for additional details.
- **Design\_Functional = NULL**  
The design functional tuple is used to input functional information for optimization, see [CFD Functional](#) for additional details. Using this requires `Design_SensFile = False`.
- **Design\_SensFile = False**  
Read `<Proj_Name>.sens` file to compute functional sensitivities w.r.t Design\_Variable. Using this requires `Design_Functional = NULL`.
- **Design\_Sensitivity = False**  
If True and Design\_Functional is set, create geometric sensitivities Fun3D input files needed to compute Design\_Functional sensitivities w.r.t Design\_Variable. If True and Design\_SensFile = True, read functional sensitivities from `<Proj_Name>.sens` and compute sensitivities w.r.t Design\_Variable. The value of the design functionals become available as Dynamic Output Value Objects using the "name" of the functionals.
- **Mesh = NULL**  
A Area\_Mesh or Volume\_Mesh link for 2D and 3D calculations respectively.

## 0.4 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.).

## 0.5 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.

### 0.5.1 Data transfer from FUN3D (FieldOut)

- **"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  $\text{Pressure} = \text{Pressure\_Scale\_Factor} * \text{Cp} + \text{Pressure\_Scale\_Offset}$ , where "Pressure\_Scale\_Factor" and "Pressure\_Scale\_Offset" are AIM inputs ([AIM Inputs](#)).

### 0.5.2 Data transfer to FUN3D (FieldIn)

- **"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 Eigenvector\_6 would be the sixth mode) . A [project\_name]\_body1\_mode#.dat file is written out for each mode.

## 0.6 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](#))

### 0.6.1 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
- SubsonicInflow
- SubsonicOutflow
- MassflowIn
- MassflowOut
- MachOutflow

### 0.6.1.1 Wall Properties

- **wallTemperature = 0.0**

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

### 0.6.1.2 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.

### 0.6.1.3 Static Properties

- **staticPressure = 0.0**

Ratio of static pressure to reference pressure on a boundary surface.

### 0.6.1.4 Velocity Components

- **machNumber = 0.0**

Mach number on boundary.

### 0.6.1.5 Massflow Properties

- **massflow = 0.0**

Massflow through the boundary in units of grid units squared.

## 0.6.2 Single Value String

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

- "Inviscid" (default)
- "Viscous"
- "Farfield"
- "Extrapolate"
- "Freestream"
- "SymmetryX"
- "SymmetryY"
- "SymmetryZ"

## 0.7 CFD Modal Aeroelastic

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](#)).

### 0.7.1 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.

## 0.8 CFD Design Variable

Structure for the design variable tuple = ("DesignVariable Name", "Value"). "DesignVariable Name" defines the reference name for the design variable being specified. The "Value" may be a JSON String dictionary (see Section [JSON String Dictionary](#)) or just a blank string (see Section [Single Value String](#)).

Note that any JSON string inputs are written to the input files as information only. They are only use if the analysis is executed with the analysis specific design framework.

### 0.8.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"upperBound": 10.0}) the following keywords (= default values) may be used:

- **lowerBound = 0.0 or [0.0, 0.0,...]**  
Lower bound for the design variable.
- **upperBound = 0.0 or [0.0, 0.0,...]**  
Upper bound for the design variable.

## 0.8.2 Single Value String

If "Value" is a string, the string value will be ignored.

## 0.9 CFD Functional

Structure for the design functional tuple = ("Functional Name", "Value"). "Functional Name" defines the functional returned as a dynamic output. The "Value" must be a JSON String dictionary (see Section [JSON String Dictionary](#)).

For FUN3D, a functional in which the adjoint will be taken with respect to can be build up using:

Function Names	Description
"cl", "cd"	Lift, drag coefficients
"clp", "cdp"	Lift, drag coefficients: pressure contributions
"clv", "cdv"	Lift, drag coefficients: shear contributions
"cmx", "cmy", "cmz"	x/y/z-axis moment coefficients
"cmxp", "cmyp", "cmzp"	x/y/z-axis moment coefficients: pressure contributions
"cmxv", "cmyv", "cmzv"	x/y/z-axis moment coefficients: shear contributions
"cx", "cy", "cz"	x/y/z-axis force coefficients
"cxp", "cyp", "czp"	x/y/z-axis force coefficients: pressure contributions
"cxv", "cyv", "czv"	x/y/z-axis force coefficients: shear contributions
"powerx", "powery", "powerz"	x/y/z-axis power coefficients
"clcd"	Lift-to-drag ratio
"fom"	Rotorcraft figure of merit
"propeff"	Rotorcraft propulsive efficiency
"rtr"	thrust Rotorcraft thrust function
"pstag"	RMS of stagnation pressure in cutting plane disk
"distort"	Engine inflow distortion
"boom"	Near-field $p/p_\infty$ pressure target
"sboom"	Coupled sBOOM ground-based noise metrics
"ae"	Supersonic equivalent area target distribution
"press"	box RMS of pressure in user-defined box, also pointwise $dp/dt$ , $d\rho/dt$
"cpstar"	Target pressure distributions

FUN3D calculates a functional using the following form:  $f = \sum_i (w_i * (C_i - C_i^*)^{p_i})$

Where:

$f$  : Functional

$w_i$  : Weighting of function

$C_i$  : Function type (cl, cd, etc.)

$C_i^*$  : Function target

$p_i$  : Exponential factor of function

### 0.9.1 JSON String Dictionary

If "Value" is a JSON string dictionary (eg. "Value" = "Composite":[{"function": "cl", "weight": 3.0, "target": 10.7}, {"function": "cd", "weight": 2.0, "power": 2.0}])

which represents the composite functional:  $Composite = 3(c_l - 10.7) + 2c_d^2$

The following keywords ( = default values) may be used:

- **capsGroup = "GroupName"**  
Name of boundary to apply for the function  $C_i$ .
- **function = NULL**  
The name of the function  $C_i$ , e.g. "cl", "cd", etc.
- **weight = 1.0**  
This weighting  $w_i$  of the function.
- **target = 0.0**  
This is the target value  $C_i^*$  of the function.
- **power = 1.0**  
This is the user defined power operator  $p_i$  for the function.

## 0.10 FUN3D AIM Example

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

### 0.10.1 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.

#### 0.10.1.1 Script files

Two scripts are used for this illustration:

1. `cfMultiBody.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](#).

### 0.10.2 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 40.00000
DESPMTR aspect 5.00000
DESPMTR taper 0.50000
DESPMTR twist 15.00000
DESPMTR lesweep 30.00000
```

```
DESPMTR    dihedral    1.00000
```

as well as the following configuration parameters. 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
```

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

```
# Store half of wing and keep a copy on the stack
STORE      HalfWing 0 1
# Restore and mirror the half wing
RESTORE    HalfWing 0
  MIRROR   0 0 1 0
# Combine halves 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 capsMesh     $Wing1
  ATTRIBUTE _name        $Wing1
  ATTRIBUTE AFLR4_Cmp_ID 1
  ATTRIBUTE AFLR4_Edge_Refinement_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 capsMesh     $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 capsMesh     $Farfield
  ATTRIBUTE _name        $Farfield
  ATTRIBUTE AFLR_GBC     $FARFIELD_UG3_GBC
  ATTRIBUTE AFLR4_Cmp_ID 3
  ATTRIBUTE capsMeshLength cmean #Characteristic length for meshing
  ATTRIBUTE .tParam      "30.;5.;30;"
```

### 0.10.3 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,

```
import pyCAPS
import os
import argparse
```

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

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

Once the required modules have been loaded, a pyCAPS.Problem can be instantiated with the desired geometry file.

```
geometryScript = os.path.join("../", "csmData", "cfdMultiBody.csm")
myProblem = pyCAPS.Problem(problemName=workDir,
                           capsFile=geometryScript,
                           outLevel=args.outLevel)
```

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.

```
myProblem.geometry.despmtr.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 is loaded using the following

```
# Load egadsTess aim
myProblem.analysis.create(aim = "egadsTessAIM", name = "egadsTess")
# Load Tetgen aim
meshAIM = myProblem.analysis.create(aim = "tetgenAIM", name = "tetgen")
```

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.

```
# Set new EGADS body tessellation parameters
myProblem.analysis["egadsTess"].input.Tess_Params = [1.0, 0.01, 20.0]
# Preserve surface mesh while meshing
meshAIM.input.Preserve_Surf_Mesh = True
# Link Surface_Mesh
meshAIM.input["Surface_Mesh"].link(myProblem.analysis["egadsTess"].output["Surface_Mesh"])
```

In the case, the EGADS and TetGen AIMS execute in memory automatically.

Next the FUN3D AIM needs to be loaded.

```
fun3dAIM = myProblem.analysis.create(aim = "fun3dAIM",
                                     name = "fun3d")
```

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. In this example, the Volume\_Mesh from TetGen AIM is linked to the FUN3D Mesh input. This allows the volume mesh generated by the TetGen AIM to be transferred by the FUN3D AIM, in which case FUN3D will write out the mesh in its preferred, native format.

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 "name" reference in the analysis dictionary. While syntactically different, these two forms are essentially identical.

```
# Set project name
fun3dAIM.input.Proj_Name = "fun3dTetgenTest"
# Link the mesh
fun3dAIM.input["Mesh"].link(meshAIM.output["Volume_Mesh"])
fun3dAIM.input.Mesh_ASCII_Flag = False
# Set AoA number
myProblem.analysis["fun3d"].input.Alpha = 1.0
# Set Mach number
myProblem.analysis["fun3d"].input.Mach = 0.5901
# Set equation type
fun3dAIM.input.Equation_Type = "compressible"
# Set Viscous term
myProblem.analysis["fun3d"].input.Viscous = "inviscid"
# Set number of iterations
```



```

myProblem.analysis["fun3d"].input.Num_Iter = 10
# Set CFL number schedule
myProblem.analysis["fun3d"].input.CFL_Schedule = [0.5, 3.0]
# Set read restart option
fun3dAIM.input.Restart_Read = "off"
# Set CFL number iteration schedule
myProblem.analysis["fun3d"].input.CFL_Schedule_Iter = [1, 40]
# Set overwrite fun3d.nml if not linking to Python library
myProblem.analysis["fun3d"].input.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.input.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 been created and placed in the specified analysis working directory. Next FUN3D needs to be executed either through its Python interface module (not shown) or an OS system call such as,

```

print ("\n\nRunning FUN3D.....")
fun3d.system("nodet_mpi --animation_freq -1 --write_aero_loads_to_file> Info.out"); # Run fun3d via system call

```

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.output.CLtot,
      "Cd = " , fun3dAIM.output.CDtot)
# Get Cmx, Cmy, and Cmc coefficients
print ("Cmx = " , fun3dAIM.output.CMXtot,
      "Cmy = " , fun3dAIM.output.CMYtot,
      "Cmc = " , fun3dAIM.output.CMZtot)
# Get Cx, Cy, Cz coefficients
print ("Cx = " , fun3dAIM.output.CXtot,
      "Cy = " , fun3dAIM.output.CYtot,
      "Cz = " , fun3dAIM.output.CZtot)
print ("Pressure Contribution")
# Get Lift and Drag coefficients
print ("Cl_p = " , fun3dAIM.output.CLtot_p,
      "Cd_p = " , fun3dAIM.output.CDtot_p)
# Get Cmx, Cmy, and Cmc coefficients
print ("Cmx_p = " , fun3dAIM.output.CMXtot_p,
      "Cmy_p = " , fun3dAIM.output.CMYtot_p,
      "Cmc_p = " , fun3dAIM.output.CMZtot_p)
# Get Cx, Cy, and Cz coefficients
print ("Cx_p = " , fun3dAIM.output.CXtot_p,
      "Cy_p = " , fun3dAIM.output.CYtot_p,
      "Cz_p = " , fun3dAIM.output.CZtot_p)
print ("Viscous Contribution")
# Get Lift and Drag coefficients
print ("Cl_v = " , fun3dAIM.output.CLtot_v,
      "Cd_v = " , fun3dAIM.output.CDtot_v)
# Get Cmx, Cmy, and Cmc coefficients
print ("Cmx_v = " , fun3dAIM.output.CMXtot_v,
      "Cmy_v = " , fun3dAIM.output.CMYtot_v,
      "Cmc_v = " , fun3dAIM.output.CMZtot_v)
# Get Cx, Cy, and Cz coefficients
print ("Cx_v = " , fun3dAIM.output.CXtot_v,
      "Cy_v = " , fun3dAIM.output.CYtot_v,
      "Cz_v = " , fun3dAIM.output.CZtot_v)

```

results in,

```

Total Force - Pressure + Viscous
Cl = 0.671595 Cd = 0.517818
Cmx = -0.002830832 Cmy = -1.342669 Cmc = -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 Cmc_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 Cmc_v = 0.0
Cx_v = 0.0 Cy_v = 0.0 Cz_v = 0.0

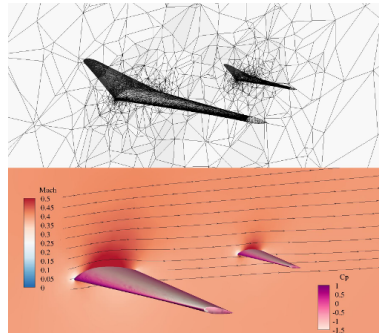
```

#### 0.10.4 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**

# Bibliography

- [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. [1](#)

