Masstran Analysis Interface Module (AIM)

Ryan Durscher
AFRL/RQVC

January 23, 2020

Contents

1 Introduction

1.1 Masstran AIM Overview

A module in the Computational Aircraft Prototype Syntheses (CAPS) has been developed to compute mass properties using attributions for finite element structural solvers.

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

The mass properties are computed via the formulas:

$$
\begin{aligned}
m &= \sum_i m_i \\
x_{cg} &= \frac{1}{m}\sum_i m_i x_i \\
y_{cg} &= \frac{1}{m}\sum_i m_i y_i \\
z_{cg} &= \frac{1}{m}\sum_i m_i z_i \\
(I_{xx})_{cg} &= \sum_i m_i\left(y_i^2+z_i^2\right) - m\left(y_{cg}^2+z_{cg}^2\right) \\
(I_{yy})_{cg} &= \sum_i m_i\left(x_i^2+z_i^2\right) - m\left(x_{cg}^2+z_{cg}^2\right) \\
(I_{zz})_{cg} &= \sum_i m_i\left(x_i^2+y_i^2\right) - m\left(x_{cg}^2+y_{cg}^2\right) \\
(I_{xy})_{cg} &= \sum_i m_i\left(x_i y_i\right) - m\left(x_{cg} y_{cg}\right) \\
(I_{xz})_{cg} &= \sum_i m_i\left(x_i z_i\right) - m\left(x_{cg} z_{cg}\right) \\
(I_{yz})_{cg} &= \sum_i m_i\left(y_i z_i\right) - m\left(y_{cg} z_{cg}\right),
\end{aligned}
$$

where i represents an element index in the mesh, and the mass m_i is computed from the density, thickness, and area of the element.

The moment of inertias are accessible individually, in vector form as

$$\vec{I} = \begin{bmatrix} I_{xx} & I_{yy} & I_{zz} & I_{xy} & I_{xz} & I_{yz} \end{bmatrix},$$

as lower/upper triangular form

$$\vec{I}_{lower} = \begin{bmatrix} I_{xx} & -I_{xy} & I_{yy} & -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix},$$

$$\vec{I}_{upper} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} & I_{yy} & -I_{yz} & I_{zz} \end{bmatrix},$$

or in full tensor form as

$$\bar{\bar{I}} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}.$$

1.2 Examples

An example problem using the Masstran AIM may be found at Masstran AIM Basic Example.

2 Masstran AIM attributes

The following list of attributes are required for the MYSTRAN AIM inside the geometry input.

- **capsAIM** This attribute is a CAPS requirement to indicate the analysis the geometry representation supports.
- **capsGroup** This is a name assigned to any geometric body. This body could be a solid, surface, face, wire, edge or node. Recall that a string in ESP starts with a $. For example, attribute `capsGroup $Wing`.
- **capsIgnore** It is possible that there is a geometric body (or entity) that you do not want the Masstran AIM to pay attention to when creating a finite element model. The capsIgnore attribute allows a body (or entity) to be in the geometry and ignored by the AIM. For example, because of limitations in OpenCASCADE a situation where two edges are overlapping may occur; capsIgnore allows the user to only pay attention to one of the overlapping edges.

3 AIM Inputs

The following list outlines the Masstran inputs along with their default value available through the AIM interface.

- **Tess_Params = [0.025, 0.001, 15.0]**
  Body tessellation parameters used when creating a boundary element model. Tess_Params[0] and Tess↩_Params[1] get scaled by the bounding box of the body. (From the EGADS manual) A set of 3 parameters that drive the EDGE discretization and the FACE triangulation. The first is the maximum length of an ED↩GE segment or triangle side (in physical space). A zero is flag that allows for any length. The second is a curvature-based value that looks locally at the deviation between the centroid of the discrete object and the underlying geometry. Any deviation larger than the input value will cause the tessellation to be enhanced in those regions. The third is the maximum interior dihedral angle (in degrees) between triangle facets (or Edge segment tangents for a WIREBODY tessellation), note that a zero ignores this phase
- **Edge_Point_Min = 0**
  Minimum number of points along an edge to use when creating a boundary element model.
- **Edge_Point_Max = 0**
  Maximum number of points along an edge to use when creating a boundary element model.
- **Quad_Mesh = False**
  Create a quadratic mesh on four edge faces when creating the boundary element model.
- **Property = NULL**
  Property tuple used to input property information for the model, see FEA Property for additional details.
- **Material = NULL**
  Material tuple used to input material information for the model, see FEA Material for additional details.

4 AIM Outputs

The following list outlines the Masstran outputs available through the AIM interface.

- **Area** = Total area of the mesh.
- **Mass** = Total mass of the model.
- **Centroid** = Centroid of the model.
- **CG** = Center of gravity of the model.
- **Ixx** = Moment of inertia
- **Iyy** = Moment of inertia
- **Izz** = Moment of inertia
- **Ixy** = Moment of inertia
- **Izy** = Moment of inertia
- **Iyz** = Moment of inertia
- **I_Vector** = Moment of inertia vector

$$\vec{I} = \begin{bmatrix} I_{xx} & I_{yy} & I_{zz} & I_{xy} & I_{xz} & I_{yz} \end{bmatrix}$$

- **I_Lower** = Moment of inertia lower triangular tensor

$$\vec{I}_{lower} = \begin{bmatrix} I_{xx} & -I_{xy} & I_{yy} & -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix},$$

- **I_Upper** = Moment of inertia upper triangular tensor

$$\vec{I}_{upper} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} & I_{yy} & -I_{yz} & I_{zz} \end{bmatrix},$$

- **I_Tensor** = Moment of inertia tensor

$$\bar{\bar{I}} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}$$

5 FEA Material

Structure for the material tuple = ("Material Name", "Value"). "Material Name" defines the reference name for the material being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

5.1 JSON String Dictionary

If "Value" is JSON string dictionary (e.g. "Value" = {"density": 7850, "youngModulus": 120000.0, "poissonRatio": 0.5, "materialType": "isotropic"}) the following keywords (= default values) may be used:

- **materialType = "Isotropic"**
  Material property type. Options: Isotropic, Anisothotropic, Orthotropic, or Anisotropic.

- **youngModulus = 0.0**
  Also known as the elastic modulus, defines the relationship between stress and strain. Default if ‘shear↩ Modulus' and ‘poissonRatio' != 0, youngModulus = 2∗(1+poissonRatio)∗shearModulus

- **shearModulus = 0.0**
  Also known as the modulus of rigidity, is defined as the ratio of shear stress to the shear strain. Default if 'youngModulus' and 'poissonRatio' != 0, shearModulus = youngModulus/(2∗(1+poissonRatio))

- **poissonRatio = 0.0**
  The fraction of expansion divided by the fraction of compression. Default if 'youngModulus' and 'shear↩ Modulus' != 0, poissonRatio = (2∗youngModulus/shearModulus) - 1

- **density = 0.0**
  Density of the material.

- **thermalExpCoeff = 0.0**
  Thermal expansion coefficient of the material.

- **thermalExpCoeffLateral = 0.0**
  Thermal expansion coefficient of the material.

- **temperatureRef = 0.0**
  Reference temperature for material properties.

- **dampingCoeff = 0.0**
  Damping coefficient for the material.

- **allowType = 0**
  This flag defines if the above allowables `compressAllow` etc. are defined in terms of stress (0) or strain (1). The default is stress (0).

- **youngModulusLateral = 0.0**
  Elastic modulus in lateral direction for an orthotropic material

- **shearModulusTrans1Z = 0.0**
  Transverse shear modulus in the 1-Z plane for an orthotropic material

- **shearModulusTrans2Z = 0.0**
  Transverse shear modulus in the 2-Z plane for an orthotropic material

5.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined material lookup table. NOT YET IMPLEMENTED!!!!

6 FEA Property

Structure for the property tuple = ("Property Name", "Value"). "Property Name" defines the reference `capsGroup` for the property being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

6.1 JSON String Dictionary

If "Value" is JSON string dictionary (e.g. "Value" = {"shearMembraneRatio": 0.83, "bendingInertiaRatio": 1.0, "membraneThickness": 0.2, "propertyType": "Shell"}) the following keywords (= default values) may be used:

- **propertyType = No Default value**
  Type of property to apply to a given capsGroup `Name`. Options: ConcentratedMass, Rod, Bar, Shear, Shell, Composite, and Solid

- **material = "Material Name" (FEA Material)**
  "Material Name" from FEA Material to use for property. If no material is set the first material created will be used

- **crossSecArea = 0.0**
  Cross sectional area.

- **torsionalConst = 0.0**
  Torsional constant.

- **torsionalStressReCoeff = 0.0**
  Torsional stress recovery coefficient.

- **massPerLength = 0.0**
  Mass per unit length.

- **zAxisInertia = 0.0**
  Section moment of inertia about the element z-axis.

- **yAxisInertia = 0.0**
  Section moment of inertia about the element y-axis.

- **yCoords[4] = [0.0, 0.0, 0.0, 0.0]**
  Element y-coordinates, in the bar cross-section, of four points at which to recover stresses

- **zCoords[4] = [0.0, 0.0, 0.0, 0.0]**
  Element z-coordinates, in the bar cross-section, of four points at which to recover stresses

- **areaShearFactors[2] = [0.0, 0.0]**
  Area factors for shear.

- **crossProductInertia = 0.0**
  Section cross-product of inertia.

- **shearPanelThickness = 0.0**
  Shear panel thickness.

- **nonStructMassPerArea = 0.0**
  Nonstructural mass per unit area.

- **membraneThickness = 0.0**
  Membrane thickness.

- **bendingInertiaRatio = 1.0**
  Ratio of actual bending moment inertia to the bending inertia of a solid plate of thickness "membrane↩ Thickness"

- **shearMembraneRatio = 5.0/6.0**
  Ratio shear thickness to membrane thickness.

- **materialBending = "Material Name" (FEA Material)**
  "Material Name" from FEA Material to use for property bending. If no material is given and "bendingInertia↩Ratio" is greater than 0, the material name provided in "material" is used.

- **materialShear = "Material Name" (FEA Material)**
  "Material Name" from FEA Material to use for property shear. If no material is given and "shearMembrane↩Ratio" is greater than 0, the material name provided in "material" is used.

- **massPerArea = 0.0**
  Mass per unit area.

- **compositeMaterial = "no default"**
  List of "Material Name"s, ["Material Name -1", "Material Name -2", ...], from FEA Material to use for composites.

- **shearBondAllowable = 0.0**
  Allowable interlaminar shear stress.

- **symmetricLaminate = False**
  Symmetric lamination option. True- SYM only half the plies are specified, for odd number plies 1/2 thickness of center ply is specified with the first ply being the bottom ply in the stack, default (False) all plies specified.

- **compositeFailureTheory = "(no default)"**
  Composite failure theory. Options: "HILL", "HOFF", "TSAI", and "STRN"

- **compositeThickness = (no default)**
  List of composite thickness for each layer (e.g. [1.2, 4.0, 3.0]). If the length of this list doesn't match the length of the "compositeMaterial" list, the list is either truncated [$>$length("compositeMaterial")] or expanded [$<$length("compositeMaterial")] in which case the last thickness provided is repeated.

- **compositeOrientation = (no default)**
  List of composite orientations (angle relative element material axis) for each layer (eg. [5.0, 10.0, 30.0]). If the length of this list doesn't match the length of the "compositeMaterial" list, the list is either truncated [$>$length("compositeMaterial")] or expanded [$<$length("compositeMaterial")] in which case the last orientation provided is repeated.

- **mass = 0.0**
  Mass value.

- **massOffset = [0.0, 0.0, 0.0]**
  Offset distance from the grid point to the center of gravity for a concentrated mass.

- **massInertia = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]**
  Mass moment of inertia measured at the mass center of gravity.

6.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined property lookup table. NOT YET IMPLEMENTED!!!!

7 FEA Constraint

Structure for the constraint tuple = ("Constraint Name", "Value"). "Constraint Name" defines the reference name for the constraint being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

7.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"groupName": "plateEdge", "dofConstraint": 123456}) the following keywords (= default values) may be used:

- **constraintType = "ZeroDisplacement"**
  Type of constraint. Options: "Displacement", "ZeroDisplacement".

- **groupName = "(no default)"**
  Single or list of `capsConstraint` names on which to apply the constraint (e.g. "Name1" or ["Name1","↩ Name2",...]. If not provided, the constraint tuple name will be used.

- **dofConstraint = 0**
  Component numbers / degrees of freedom that will be constrained (123 - zero translation in all three directions).

- **gridDisplacement = 0.0**
  Value of displacement for components defined in "dofConstraint".

7.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined constraint lookup table. NOT YET IMPLEMENTED!!!!

8 FEA Support

Structure for the support tuple = ("Support Name", "Value"). "Support Name" defines the reference name for the support being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

8.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"groupName": "plateEdge", "dofSupport": 123456}) the following keywords (= default values) may be used:

- **groupName = "(no default)"**
  Single or list of `capsConstraint` names on which to apply the support (e.g. "Name1" or ["Name1","↩ Name2",...]. If not provided, the constraint tuple name will be used.

- **dofSupport = 0**
  Component numbers / degrees of freedom that will be supported (123 - zero translation in all three directions).

8.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined support lookup table. NOT YET IMPLEMENTED!!!!

9 FEA Connection

Structure for the connection tuple = ("Connection Name", "Value"). "Connection Name" defines the reference name to the capsConnect being specified and denotes the "source" node for the connection. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

9.1 JSON String Dictionary

If "Value" is JSON string dictionary (e.g. "Value" = {"dofDependent": 1, "propertyType": "RigidBody"}) the following keywords (= default values) may be used:

- **connectionType = RigidBody**
  Type of connection to apply to a given capsConnect pair defined by "Connection Name" and the "groupName". Options: Mass (scalar), Spring (scalar), Damper (scalar), RigidBody.

- **dofDependent = 0**
  Component numbers / degrees of freedom of the dependent end of rigid body connections (ex. 123 - translation in all three directions).

- **componentNumberStart = 0**
  Component numbers / degrees of freedom of the starting point of the connection for mass, spring, and damper elements (scalar) (0 <= Integer <= 6).

- **componentNumberEnd= 0**
  Component numbers / degrees of freedom of the ending point of the connection for mass, spring, and damper elements (scalar) (0 <= Integer <= 6).

- **stiffnessConst = 0.0**
  Stiffness constant of a spring element (scalar).

- **dampingConst = 0.0**
  Damping coefficient/constant of a spring or damping element (scalar).

- **stressCoeff = 0.0**
  Stress coefficient of a spring element (scalar).

- **mass = 0.0**
  Mass of a mass element (scalar).

- **groupName = "(no default)"**
  Single or list of `capsConnect` names on which to connect the nodes found with the tuple name ("↩
  Connection Name") to. (e.g. "Name1" or ["Name1","Name2",...].

9.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined connection lookup table. NOT YET IMPLEMENTED!!!!

10 FEA Load

Structure for the load tuple = ("Load Name", "Value"). "Load Name" defines the reference name for the load being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

10.1 JSON String Dictionary

If "Value" is JSON string dictionary (e.g. "Value" = {"groupName": "plate", "loadType": "Pressure", "pressureForce": 2000000.0}) the following keywords (= default values) may be used:

- **loadType = "(no default)"**
  Type of load. Options: "GridForce", "GridMoment", "Rotational", "Thermal", "Pressure", "PressureDistribute", "PressureExternal", "Gravity".

- **groupName = "(no default)"**
  Single or list of `capsLoad` names on which to apply the load (e.g. "Name1" or ["Name1","Name2",...]. If not provided, the load tuple name will be used.

- **loadScaleFactor = 1.0**
  Scale factor to use when combining loads.

- **forceScaleFactor = 0.0**
  Overall scale factor for the force for a "GridForce" load.

- **directionVector = [0.0, 0.0, 0.0]**
  X-, y-, and z- components of the force vector for a "GridForce", "GridMoment", or "Gravity" load.

- **momentScaleFactor = 0.0**
  Overall scale factor for the moment for a "GridMoment" load.

- **gravityAcceleration = 0.0**
  Acceleration value for a "Gravity" load.

- **pressureForce = 0.0**
  Uniform pressure force for a "Pressure" load (only applicable to 2D elements).

- **pressureDistributeForce = [0.0, 0.0, 0.0, 0.0]**
  Distributed pressure force for a "PressureDistribute" load (only applicable to 2D elements). The four values correspond to the 4 (quadrilateral elements) or 3 (triangle elements) node locations.

- **angularVelScaleFactor = 0.0**
  An overall scale factor for the angular velocity in revolutions per unit time for a "Rotational" load.

- **angularAccScaleFactor = 0.0**
  An overall scale factor for the angular acceleration in revolutions per unit time squared for a "Rotational" load.

- **coordinateSystem = "(no default)"**
  Name of coordinate system in which defined force components are in reference to. If no value is provided the global system is assumed.

- **temperature = 0.0**
  Temperature at give node for a "Temperature" load. </ ul>

  - **temperatureDefault = 0.0**
    Default temperature at a node not explicitly being used for a "Temperature" load. </ ul>

10.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined load lookup table. NOT YET IMPLEMENTED!!!!

11 FEA Analysis

Structure for the analysis tuple = (‘Analysis Name', ‘Value'). 'Analysis Name' defines the reference name for the analysis being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

11.1 JSON String Dictionary

If "Value" is JSON string dictionary (e.g. "Value" = {"numDesiredEigenvalue": 10, "eigenNormaliztion": "MAS↩S", "numEstEigenvalue": 1, "extractionMethod": "GIV", "frequencyRange": [0, 10000]}) the following keywords (= default values) may be used:

- **analysisType = "Modal"**
  Type of load. Options: "Modal", "Static".

- **analysisLoad = "(no default)"**
  Single or list of "Load Name"s defined in FEA Load in which to use for the analysis (e.g. "Name1" or ["↩Name1","Name2",...].

- **analysisConstraint = "(no default)"**
  Single or list of "Constraint Name"s defined in FEA Constraint in which to use for the analysis (e.g. "Name1" or ["Name1","Name2",...].

- **analysisSupport = "(no default)"**
  Single or list of "Support Name"s defined in FEA Support in which to use for the analysis (e.g. "Name1" or ["Name1","Name2",...].

- **extractionMethod = "(no default)"**
  Extraction method for modal analysis.

- **frequencyRange = [0.0, 0.0]**
  Frequency range of interest for modal analysis.

- **numEstEigenvalue = 0**
  Number of estimated eigenvalues for modal analysis.

- **numDesiredEigenvalue = 0**
  Number of desired eigenvalues for modal analysis.

- **eigenNormaliztion = "(no default)"**
  Method of eigenvector renormilization. Options: "POINT", "MAX", "MASS"

- **gridNormaliztion = 0**
  Grid point to be used in normalizing eigenvector to 1.0 when using eigenNormaliztion = "POINT"

- **componentNormaliztion = 0**
  Degree of freedom about "gridNormalization" to be used in normalizing eigenvector to 1.0 when using eigen↩Normaliztion = "POINT"

- **lanczosMode = 2**
  Mode refers to the Lanczos mode type to be used in the solution. In mode 3 the mass matrix, Maa,must be nonsingular whereas in mode 2 the matrix K aa - sigma∗Maa must be nonsingular

- **lanczosType = "(no default)"**
  Lanczos matrix type. Options: DPB, DGB.

- **aeroSymmetryXY = "(no default)"**
  Aerodynamic symmetry about the XY Plane. Options: SYM, ANTISYM, ASYM. Aerodynamic symmetry about the XY Plane. Options: SYM, ANTISYM, ASYM. SYMMETRIC Indicates that a half span aerodynamic model is moving in a symmetric manner with respect to the XY plane. ANTISYMMETRIC Indicates that a half span aerodynamic model is moving in an antisymmetric manner with respect to the XY plane. ASYMMETRIC Indicates that a full aerodynamic model is provided.

- **aeroSymmetryXZ = "(no default)"**
  Aerodynamic symmetry about the XZ Plane. Options: SYM, ANTISYM, ASYM. SYMMETRIC Indicates that a half span aerodynamic model is moving in a symmetric manner with respect to the XZ plane. ANTISYM↩ METRIC Indicates that a half span aerodynamic model is moving in an antisymmetric manner with respect to the XZ plane. ASYMMETRIC Indicates that a full aerodynamic model is provided.

11.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined analysis lookup table. NOT YET IMPLEMENTED!!!!

12 FEA DesignVariable

Structure for the design variable tuple = ("DesignVariable Name", "Value"). "DesignVariable Name" defines the reference name for the design variable being specified. This string will be used in the FEA input directly. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

12.1 JSON String Dictionary

If "Value" is JSON string dictionary the following keywords (= default values) may be used:

13 FEA DesignConstraint

Structure for the design constraint tuple = ('DesignConstraint Name', 'Value'). 'DesignConstraint Name' defines the reference name for the design constraint being specified. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

13.1 JSON String Dictionary

If "Value" is JSON string dictionary the following keywords (= default values) may be used:

14 Masstran AIM Basic Example

This is a walkthrough for using Masstran AIM to analyze a three-dimensional wing with internal ribs and spars.

14.1 Prerequisites

It is presumed that ESP and CAPS have been already installed, as well as Masstran.

14.1.1 Script files

Two scripts are used for this illustration:

1. feaWingBEM.csm: Creates geometry, as described in the next section (Creating Geometry using ESP).
2. masstran_PyTest.py: pyCAPS script for performing analysis, as described in Performing analysis using py↩ CAPS .

14.2 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 structural finite element analysis tools that can analyze the body:

```
attribute capsAIM $nastranAIM;astrosAIM;mystranAIM;masstranAIM;egadsTessAIM
```

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 wing configuration is created using following design parameters.

```
# Design Parameters for OML
despmtr   thick     0.12      frac of local chord
despmtr   camber    0.04      frac of loacl chord

despmtr   area      10.0
despmtr   aspect    6.00
despmtr   taper     0.60
despmtr   sweep     20.0      deg (of c/4)

despmtr   washout   5.00      deg (down at tip)
despmtr   dihedral  4.00      deg

# Design Parameters for BEM
cfgpmtr   nrib      11        number of ribs
despmtr   spar1     0.20      frac of local chord
despmtr   spar2     0.75      frac of local chord
```

After our design parameters are defined they are used to setup other local variables (analytically) for the outer model line (OML).

```
# OML
set       span      sqrt(aspect*area)
set       croot     2*area/span/(1+taper)
set       ctip      croot*taper
set       dxtip     (croot-ctip)/4+span/2*tand(sweep)
set       dytip     span/2*tand(dihedral)
```

In a similar manner, local variables are defined for the ribs and spars.

```
# wing ribs
set       Nrib    nint(nrib)
# wing spars
set       eps             0.01*span
```

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

```
mark
   # Right tip
   udprim    naca      Thickness thick     Camber    camber
   scale     ctip
   rotatez   washout   ctip/4    0
   translate dxtip     dytip     -span/2

   # root
   udprim    naca      Thickness thick     Camber    camber
   scale     croot

   # left tip
   udprim    naca      Thickness thick     Camber    camber
   scale     ctip
   rotatez   washout   ctip/4    0
   translate dxtip     dytip     +span/2
rule
   attribute OML 1
```

Once complete, the wing is stored for later use under the name OML.

```
store     OML
```

Next, the inner layout of the ribs and spars are created using the waffle udprim.

```
udprim    waffle    Depth     +6*thick*croot      Filename <<

   patbeg    i Nrib
      point A   at (span/2)*(2*i-Nrib-1)/Nrib   -0.01*croot
      point B   at (span/2)*(2*i-Nrib-1)/Nrib   max(croot,dxtip+ctip)
      line  AB  A  B   tagComponent=rib tagIndex=!val2str(i,0)
   patend

   point A   at -span/2-eps   spar1*ctip+dxtip
   point B   at 0             spar1*croot
   line  AB  A  B   tagComponent=spar tagIndex=1 tagPosition=left

   point A   at span/2+eps    spar1*ctip+dxtip
   point B   at 0             spar1*croot
   line  AB  A  B    tagComponent=spar tagIndex=1 tagPosition=right

   point A   at -span/2-eps   spar2*ctip+dxtip
   point B   at 0             spar2*croot
   line  AB  A  B   tagComponent=spar tagIndex=2 tagPosition=left

   point A   at span/2+eps    spar2*ctip+dxtip
   point B   at 0             spar2*croot
   line  AB  A  B   tagComponent=spar tagIndex=2 tagPosition=right
>>
```

An attribute is then placed on ribs and spars so that the geometry components may be reference by the Masstran AIM.

```
attribute capsGroup $Ribs_and_Spars
```

Following a series of rotations and translations the ribs and spars are stored for later use.

```
translate 0         0         -3*thick*croot
rotatey   90        0         0
rotatez  -90        0         0

store     layoutRibSpar
```

Next, the layout of the ribs and spars are intersected the outer mold line of wing, which results in only keeping the part of layout that is inside the OML.

```
restore   layoutRibSpar
restore   OML
intersect
```

Finally, select faces (airfoil sections at the root) are tagged, so that a constraint may be applied later.

```
udprim editAttr filename <<
   edge adj2face tagComponent=spar tagPosition=right
   and  adj2face tagComponent=spar tagPosition=left
   set  capsConstraint=Rib_Constraint

   node adj2face tagComponent=spar tagPosition=right
   and  adj2face tagComponent=spar tagPosition=left
   set  capsConstraint=Rib_Constraint
>>

ifthen nint(mod(Nrib,2)) ne 0
   set       midRib Nrib/2
   select    face   $tagComponent $rib $tagIndex val2str(midRib,0)
   attribute tagPosition $root

   udprim editAttr filename <<
      face   has tagComponent=rib tagPosition=root
      set    capsConstraint=Rib_Constraint

      edge   adj2face tagComponent=rib tagPosition=root
      set    capsConstraint=Rib_Constraint

      node   adj2face tagComponent=rib tagPosition=root
      set    capsConstraint=Rib_Constraint
>>
endif
```

The above ∗.csm file results in the follow geometry model:

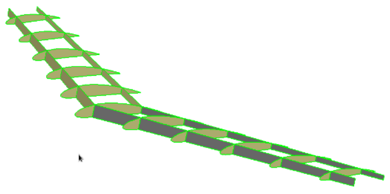

Figure 1: Wing built up element model

14.3 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,

```
from __future__ import print_function

import os
import argparse
```

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.

```
from pyCAPS import capsProblem
```

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

```
myProblem = capsProblem()
```

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

```
myProblem.loadCAPS("../csmData/feaWingBEM.csm", verbosity=args.verbosity)
```

After the geometry is loaded, a structural mesh is generated using the egadsTessAIM.

```
# Load egadsTess aim
myProblem.loadAIM( aim = "egadsTessAIM",
                   altName = "tess" )

# All triangles in the grid
myProblem.analysis["tess"].setAnalysisVal("Mesh_Elements", "Quad")

# Set global tessellation parameters
myProblem.analysis["tess"].setAnalysisVal("Tess_Params", [.05,.5,15])

# Generate the surface mesh
myProblem.analysis["tess"].preAnalysis()
myProblem.analysis["tess"].postAnalysis()
```

Next, the Masstran AIM is instantiated with the surface mesh as a parent.

```
masstranAIM = myProblem.loadAIM(aim = "masstranAIM",
                                altName = "masstran",
                                parents = ["tess"] )
```

Once loaded analysis parameters specific to Masstran need to be set (see AIM Inputs). These parameters are automatically converted into Masstran specific format and transferred into the Masstran 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 meshing inputs
myProblem.analysis["masstran"].setAnalysisVal("Edge_Point_Min", 2)
myProblem.analysis["masstran"].setAnalysisVal("Edge_Point_Max", 3)

myProblem.analysis["masstran"].setAnalysisVal("Quad_Mesh", True)
```

Along the same lines of setting the input values above the "Material" (see FEA Material) and "Property" (see F↩ EA Property) tuples are used to set more complex information. The user is encouraged to read the additional documentation on these inputs for further explanations. Once provided this information is converted into Masstran specific syntax and set in the Masstran configuration file.

```
# Set materials
unobtainium  = {"youngModulus" : 2.2E11 ,
                "poissonRatio" : .33,
                "density"      : 7850}

madeupium    = {"materialType" : "isotropic",
                "youngModulus" : 1.2E9 ,
                "poissonRatio" : .5,
                "density"      : 7850}
masstranAIM.setAnalysisVal("Material", [("Unobtainium", unobtainium),
                                        ("Madeupium", madeupium)])

# Set property
shell  = {"propertyType"       : "Shell",
          "membraneThickness" : 0.2,
          "bendingInertiaRatio"  : 1.0, # Default
          "shearMembraneRatio"   : 5.0/6.0} # Default }

masstranAIM.setAnalysisVal("Property", ("Ribs_and_Spars", shell))
```

After all desired options are set aimPreAnalysis needs to be executed by calling the pre/post-Analysis. The MasstrainAIM will compute all mass properties in memory without writing files.

```
masstranAIM.preAnalysis()
masstranAIM.postAnalysis()
```

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

```
# Get mass properties
print  ("\nGetting results mass properties.....\n")
Area     = masstranAIM.getAnalysisOutVal("Area")
Mass     = masstranAIM.getAnalysisOutVal("Mass")
Centroid = masstranAIM.getAnalysisOutVal("Centroid")
CG       = masstranAIM.getAnalysisOutVal("CG")
Ixx      = masstranAIM.getAnalysisOutVal("Ixx")
Iyy      = masstranAIM.getAnalysisOutVal("Iyy")
Izz      = masstranAIM.getAnalysisOutVal("Izz")
Ixy      = masstranAIM.getAnalysisOutVal("Ixy")
Ixz      = masstranAIM.getAnalysisOutVal("Ixz")
Iyz      = masstranAIM.getAnalysisOutVal("Iyz")
I        = masstranAIM.getAnalysisOutVal("I_Vector")
II       = masstranAIM.getAnalysisOutVal("I_Tensor")

print("Area     ", Area)
print("Mass     ", Mass)
print("Centroid ", Centroid)
print("CG       ", CG)
print("Ixx      ", Ixx)
print("Iyy      ", Iyy)
print("Izz      ", Izz)
print("Ixy      ", Ixy)
print("Ixz      ", Ixz)
print("Iyz      ", Iyz)
print("I        ", I)
print("II       ", II)
```

results in,

```
Area       3.28946557
Mass       5164.46094491
Centroid   [1.2409841844368583, 0.16359702451265337, 4.0874212239589455e-09]
CG         [1.2409841844368585, 0.16359702451265348, 4.087420991763218e-09]
Ixx        21325.300951
Iyy        22558.0731769
Izz        1292.98036179
```

```
Ixy       153.720903861
Ixz       2.06373216532e-06
Iyz       2.36311990987e-06
I         [21325.300951015903, 22558.07317691867, 1292.9803617927173, 153.72090386142395, 2.063732165317917
      e-06, 2.363119909871653e-06]
II        [[21325.300951015903, -153.72090386142395, -2.063732165317917e-06], [-153.72090386142395, 22558.0
      7317691867, -2.363119909871653e-06], [-2.063732165317917e-06, -2.363119909871653e-06, 1292.9803617927173]]
```

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

```
myProblem.closeCAPS()
```

14.4 Executing pyCAPS script

Issuing the following command executes the script:

```
python masstran_PyTest.py
```