Nastran Analysis Interface Module (AIM)

Ryan Durscher and Ed Alyanak
AFRL/RQVC

June 12, 2017

Contents

1 Introduction

1.1 Nastran AIM Overview

A module in the Computational Aircraft Prototype Syntheses (CAPS) has been developed to interact (primarily through input files) with the finite element structural solver Nastran [1].

Current issues include:

- A thorough bug testing needs to be undertaken.

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

The accepted and expected geometric representation and analysis intentions are detailed in Geometry Representation and Analysis Intent.

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 Nastran Data Transfer

1.2 Examples

Example problems using the Nastran AIM may be found at Nastran AIM Examples .

- Single Load Case Example
- Multiple Load/Boundary Case Example
- Modal Analysis Example Case
- Optimization Example Case
- Composite Wing Example
- Composite Wing Optimization Example

2 Nastran AIM attributes

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

- **capsIntent** This attribute is a CAPS requirement to indicate the analysis fidelity the geometry representation supports. Options are: ALL, STRUCTURE
- **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`.
- **capsLoad** This is a name assigned to any geometric body where a load is applied. This attribute was separated from the `capsGroup` attribute to allow the user to define a local area to apply a load on without adding multiple `capsGroup` attributes. Recall that a string in ESP starts with a $. For example, attribute `capsLoad $force`.
- **capsConstraint** This is a name assigned to any geometric body where a constraint/boundary condition is applied. This attribute was separated from the `capsGroup` attribute to allow the user to define a local area to apply a boundary condition without adding multiple `capsGroup` attributes. Recall that a string in ESP starts with a $. For example, attribute `capsConstraint $fixed`.
- **capsIgnore** It is possible that there is a geometric body that you do not want the Nastran AIM to pay attention to when creating a finite element model. The capsIgnore attribute allows a body to be in the geometry and ignored by the AIM. Because of limitations in OpenCASCADE situation where two edges are overlapping may occur. This allows the user to only pay attention to on of the overlapping edges.
- **capsConnect** This is a name assigned to any geometric body where the user wishes to create "fictitious" connections such as springs, dampers, and/or rigid body connections to. The user must manually specify the connection between two `capsConnect` entities using the "Connect" tuple (see AIM Inputs). Recall that a string in ESP starts with a $. For example, attribute `capsConnect $springStart`.
- **capsConnectLink** Similar to `capsConnect`, this is a name assigned to any geometric body where the user wishes to create "fictitious" connections to. A connection is automatically made if a `capsConnectLink` matches a `capsConnect` group. Again further specifics of the connection are input using the "Connect" tuple (see AIM Inputs). Recall that a string in ESP starts with a $. For example, attribute `capsConnect↩ Link $springEnd`.
- **capsReferenceArea** Reference area to use when doing aeroelastic analysis.
- **capsReferenceSpan** Reference span to use when doing aeroelastic analysis.
- **capsReferenceChord** Reference chord to use when doing aeroelastic analysis.

3 Geometry Representation and Analysis Intent

The attribute capsIntent may be set to either ALL or STRUCTURE for the Nastran AIM. The geometric representation for the AIM requires that bodies be:

- WIREBODY for purely 1D simulations
- FACEBODY or SHEETBODY (non-manifold) for 2D simulations
- SOLIDBODY or SHEETBODY (manifold) for 3D simulations

4 AIM Inputs

The following list outlines the Nastran inputs along with their default value available through the AIM interface. Unless noted these values will be not be linked to any parent AIMs with variables of the same name.

- **Proj_Name = "nastran_CAPS"**
  This corresponds to the project name used for file naming.
- **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 = 4**
  Minimum number of points along an edge to use when creating a boundary element model.
- **Edge_Point_Max = 10**
  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.
- **Constraint = NULL**
  Constraint tuple used to input constraint information for the model, see FEA Constraint for additional details.
- **Load = NULL**
  Load tuple used to input load information for the model, see FEA Load for additional details.
- **Analysis = NULL**
  Analysis tuple used to input analysis/case information for the model, see FEA Analysis for additional details.
- **Analysis_Type = "Modal"**
  Type of analysis to generate files for, options include "Modal", "Static", "StaticOpt", and "Aeroelastic".
- **File_Format = "Small"**
  Formatting type for the bulk file. Options: "Small", "Large", "Free".
- **Mesh_File_Format = "Small"**
  Formatting type for the mesh file. Options: "Small", "Large", "Free".

- **Design_Variable = NULL**
  The design variable tuple used to input design variable information for the model optimization, see FEA DesignVariable for additional details.
- **Design_Constraint = NULL**
  The design constraint tuple used to input design constraint information for the model optimization, see FEA DesignConstraint for additional details.
- **ObjectiveMinMax = "Max"**
  Maximize or minimize the design objective during an optimization. Option: "Max" or "Min".
- **ObjectiveResponseType = "Weight"**
  Object response type (see Nastran manual).
- **VLM_Surface = NULL**
  Vortex lattice method tuple input. See Vortex Lattice Surface for additional details.
- **Support = NULL**
  Support tuple used to input support information for the model, see FEA Support for additional details.
- **Connect = NULL**
  Connect tuple used to define connection to be made in the, see FEA Connection for additional details.

5 AIM Shareable Data

Currently the Nastran AIM does not have any shareable data types or values. It will try, however, to inherit a "FEA_MESH" or "Volume_Mesh" from any parent AIMs. Note that the inheritance of the mesh is not required.

6 AIM Outputs

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

- **EigenValue** = List of Eigen-Values (λ) after a modal solve.
- **EigenRadian** = List of Eigen-Values in terms of radians ($\omega = \sqrt{\lambda}$) after a modal solve.
- **EigenFrequency** = List of Eigen-Values in terms of frequencies ($f = \frac{\omega}{2\pi}$) after a modal solve.
- **EigenGeneralMass** = List of generalized masses for the Eigen-Values.
- **EigenGeneralStiffness** = List of generalized stiffness for the Eigen-Values.

7 Nastran Data Transfer

The Nastran AIM has the ability to transfer displacements and eigenvectors from the AIM and pressure distributions to the AIM using the conservative and interpolative data transfer schemes in CAPS. Currently these transfers may only take place on triangular meshes.

7.1 Data transfer from Nastran

- **"Displacement"**
  Retrieves nodal displacements from the $*$.f06 file.

- **"EigenVector_#"**
  Retrieves modal eigen-vectors from the $*$.f06 file, 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).

7.2 Data transfer to Nastran

- **"Pressure"**
  Writes appropriate load cards using the provided pressure distribution.

8 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).

8.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "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**
  Is 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.

- **yieldAllow = 0.0**
  Yield strength/allowable for the material.

- **tensionAllow = 0.0**
  Tension allowable for the material.

- **tensionAllowLateral = 0.0**
  Lateral tension allowable for the material.

- **compressAllow = 0.0**
  Compression allowable for the material.

- **compressAllowLateral = 0.0**
  Lateral compression allowable for the material.

- **shearAllow = 0.0**
  Shear allowable 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

8.2 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!!!!

9 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).

9.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "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 "membranThickness"

- **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 "bendingIneria↩
  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 (eg. [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 gratvity.

9.2 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!!!!

10 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).

10.1 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".

10.2 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!!!!

11 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).

11.1 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).

11.2 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!!!!

12 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).

12.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "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",...].

12.2 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!!!!

13 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).

13.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "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.

- **pressureDistributeForce = [0.0, 0.0, 0.0, 0.0]**
  Distributed pressure force for a "PressureDistribute" load.

- **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.

13.2 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!!!!

14 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).

14.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "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", "StaticOpt", "AeroelasticStatic".

- **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",...].

- **analysisDesignConstraint = "(no default)"**
  Single or list of "Design Constraint Name"s defined in FEA DesignConstraint 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.

- **machNumber = 0.0**
  Mach number used in trim analysis.

- **dynamicPressure = 0.0**
  Dynamic pressure used in trim analysis.

- **rigidVariable = ["no default"]**
  List of rigid body motions to be used as trim variables during a trim analysis. Nastran valid labels are: ANG↩ LEA, SIDES, ROLL, PITCH, YAW, URDD1, URDD2, URDD3, URDD4, URDD5, URDD6

- **rigidConstraint = ["no default"]**
  List of rigid body motions to be used as trim constraint variables during a trim analysis. Nastran valid labels are: ANGLEA, SIDES, ROLL, PITCH, YAW, URDD1, URDD2, URDD3, URDD4, URDD5, URDD6

- **magRigidConstraint = [0.0 , 0.0, ...]**
  List of magnitudes of trim constraint variables. If none and 'rigidConstraint'(s) are specified then 0.0 is assumed for each rigid constraint.

- **controlConstraint = ["no default"]**
  List of controls surfaces to be used as trim constraint variables during a trim analysis.

- **magControlConstraint = [0.0 , 0.0, ...]**
  List of magnitudes of trim control surface constraint variables. If none and 'controlConstraint'(s) are specified then 0.0 is assumed for each control surface constraint.

14.2 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!!!!

15 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). In Nastran the DesignVariable Name will be the LABEL used in the DESVAR input. For this reason the user should keep the length of this input to a minimum number of characters, ideally 7 or less.

- `DESVAR ID LABEL XINIT XLB XUB DELXV DDVAL`

15.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"designVariableType": "Property", "groupName": "plate", "upper↩ Bound": 10.0, "fieldName": "TM"}) the following keywords (= default values) may be used:

- **designVariableType = "Property"**
  Type of design variable in an optimization problem. Options: "Material", "Property".

- **groupName = "(no default)"**
  Single or list of `capsGroup` or FEA Material name(s) to the design variable (e.g. "Name1" or ["Name1","↩ Name2",...].
  - For `designVariableType` Property a `capsGroup` Name (or names) is given. The property (see FEA Property) also assigned to the same `capsGroup` will be automatically related to this design variable entry.
  - For `designVariableType` Material a FEA Material name (or names) is given.

- **initialValue = 0.0**
  Initial value for the design variable.

- **lowerBound = 0.0**
  Lower bound for the design variable.

- **upperBound = 0.0**
  Upper bound for the design variable.

- **maxDelta = 0.0**
  Move limit for the design variable.

- **discreteValue = 0.0**
  List of discrete values do use for the design variable (e.g. [0.0,1.0,1.5,3.0].

- **fieldName = "(no default)"**
  Fieldname of variable (e.g. "E" for Young's Modulus). Design Variables can be defined as two types based on the `designVariableType` value. These are Material or Property. This means that an aspect of a material or property input can change in the optimization problem. This input specifies what aspect of the Material or Property is changing.

  1. **Material Types** Selected based on the material type (see FEA Material, materialType) referenced in the `groupName` above.
     - **MAT1,** `materialType = "Isotropic"`
       - "E", "G", "NU", "RHO", "A"
     - **MAT2,** `materialType = "Anisothotropic"`
       - "G11", "G12", "G13", "G22", "G23", "G33", "RHO", "A1", "A2", "A3"
     - **MAT8,** `materialType = "Orthotropic"`
       - "E1", "E2", "NU12", "G12", "G1Z", "G2Z", "RHO", "A1", "A2"
     - **MAT9,** `materialType = "Anisotropic"`
       - "G11", "G12", "G13", "G14", "G15", "G16"
       - "G22", "G23", "G24", "G25", "G26"
       - "G33", "G34", "G35", "G36"
       - "G44", "G45", "G46"
       - "G55", "G56", "G66"
       - "RHO", "A1", "A2", "A3", "A4", "A5", "A6"
  2. **Property Types** (see FEA Property)
     - **PROD** `propertyType = "Rod"`
       - "A", "J"
     - **PBAR** `propertyType = "Bar"`
       - "A", "I1", "I2", "J"
     - **PSHELL** `propertyType = "Shell"`
       - "T"
     - **PCOMP** `propertyType = "Composite"`
       - "T1", "THETA1", "T2", "THETA2", ... "Ti", "THETAi"
     - **PSOLID** `propertyType = "Solid"`
       - not supported

- **fieldPosition = 0**
  This input is ignored if not defined. The user may use this field instead of the `fieldName` input defined above to relate design variables and property inputs. This requires knowledge of Nastran bulk data input format for material and property input cards.

- **independentVariable = "(no default)"**
  Single or list of "DesignVariable Name"s (that is the Tuple name) used to create/designate a dependent design variable.

  - independentValue = variableWeight[1] + variableWeight[2] $*$ SUM{independentVariableWeight[i] $*$ independentVariable[i]}

- **independentVariableWeight = 1.0 or [1.0, 1.0, ...]**
  Single or list of weighting constants with respect to the variables set for "independentVariable". If the length of this list doesn't match the length of the "independentVariable" list, the list is either truncated [$>$length("independentVariable")] or expanded [$<$length("independentVariable")] in which case the **last weight is repeated**.

- **variableWeight = [1.0, 1.0]**
  Weighting constants for a dependent variable - used if "independentVariable"(s) have been provided.

16 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).

16.1 JSON String Dictionary

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

- **groupName = "(no default)"**
  Single or list of `capsGroup` name(s) to the design variable (e.g. "Name1" or ["Name1","Name2",...].The property (see FEA Property) also assigned to the same `capsGroup` will be automatically related to this constraint entry.

- **lowerBound = 0.0**
  Lower bound for the design constraint.

- **upperBound = 0.0**
  Upper bound for the design constraint.

- **responseType = "(no default)"**
  Response type options for DRESP1 Entry (see Nastran manual).

  - Implemented Options

    1. `STRESS`, for `propertyType = "Rod" or "Shell"` (see FEA Property)
    2. `CFAILURE`, for `propertyType = "Composite"` (see FEA Property)

- **fieldName = "(no default)"**
  For constraints this field is only used currently when applying constraints to composites. This field is used to identify the specific lamina in a stacking sequence that a constraint is being applied too. Note if the user has design variables for both THEATA1 and T1 it is likly that only a single constraint on the first lamina is required. For this reason the user can simply enter LAMINA1 in addition to the possible entries defined in the FEA DesignVariable section. Additionally, the `fieldPosition` integer entry below can be used. In this case `"LAMINA1" = 1`.

  - -# **Property Types** (see FEA Property)

    * **PCOMP** `propertyType = "Composite"`
      · "T1", "THETA1", "T2", "THETA2", ... "Ti", "THETAi"
      · "LAMINA1", "LAMINA2", ... "LAMINAi"

- **fieldPosition = 0**
  This input is ignored if not defined. The user may use this field instead of the `fieldName` input defined above to identify a specific lamina in a composite stacking sequence where a constraint is applied. Please read the `fieldName` information above for more information.

17 Vortex Lattice Surface

Structure for the Vortex Lattice Surface tuple = ("Name of Surface", "Value"). "Name of surface defines the name of the surface in which the data 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).

17.1 JSON String Dictionary

If "Value" is a JSON string dictionary (eg. "Value" = {"numChord": 5, "spaceChord": 1.0, "numSpan": 10, "space↩
Span": 0.5}) the following keywords (= default values) may be used:

- **groupName = "(no default)"**
  Single or list of *capsGroup* names used to define the surface (e.g. "Name1" or ["Name1","Name2",...]. If no groupName variable is provided an attempted will be made to use the tuple name instead;

- **numChord = 10**
  The number of chordwise horseshoe vortices placed on the surface.

- **spaceChord = 0.0**
  The chordwise vortex spacing parameter.

- **numSpan = 10**
  The number of spanwise horseshoe vortices placed on the surface.

- **spaceSpan = 0.0**
  The spanwise vortex spacing parameter.

- **yMirror = False**
  Mirror surface about the y-direction.

17.2 Single Value String

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

- (NONE Currently)

18 Vortex Lattice Control Surface

Structure for the Vortex Lattice Control Surface tuple = ("Name of Control Surface", "Value"). "Name of control surface defines the name of the control surface in which the data should be applied. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

18.1 JSON String Dictionary

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

18.2 Single Value String

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

- (NONE Currently)

19 Nastran AIM Examples

This section introduces the user to the Nastran AIM via examples. These examples are intended to introduce the user to nastran functionality. They make use of the information found in the AIM Inputs, AIM Outputs and Nastran AIM attributes sections.

19.1 Single Load Case Example

The first example is a simple three bar truss structure. This example is intended to demonstrate the use of all the attributes in addition to introducing the user to the Nastran AIM.

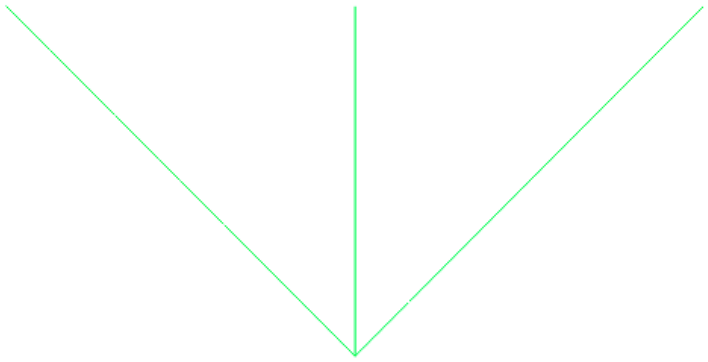

Figure 1: Three Bar Truss

The follow code details the process in a $*$.csm file that generates a three bar truss. Note to execute in serveCSM a dictionary file must be included

- `serveCSM -dict $ESP_ROOT/include/fidelity.dict feaThreeBar.csm`

First step is to define the analysis fidelity that the geometry is intended support.

```
attribute capsIntent STRUCTURE
```

Next we will define the design parameters to define the wing cross section and planform. Notice that the despmtr entries have a dimension input that must be defined for inputs with a length greater than one.

```
dimension   X   1 4 1
dimension   Y   1 4 1
dimension   Z   1 4 1

despmtr X   "-10; 0; 10;   0;"
despmtr Y   "  0; 0;  0; -10;"
despmtr Z   "  0; 0;  0;   0;"
```

Next the three bar truss is defined using the points defined in the `despmtr` entries. Notice that the middle edge is "drawn" twice. This is done because OpenCASCADE can not perform boolean operations on non-manifold (not closed) wire bodies.

```
skbeg  X[1,1] Y[1,1] Z[1,1]
linseg X[1,4] Y[1,4] Z[1,4]
linseg X[1,2] Y[1,2] Z[1,2]
linseg X[1,4] Y[1,4] Z[1,4]
linseg X[1,3] Y[1,3] Z[1,3]
skend
```

In this section the edge elements are attributed with a `capsGroup` string `$bar1` etc. so information can be assigned to them. Notice the `capsIgnore` attribute assingned to one of the overlapping "middle" edges defined in the geometry above. The reason for this is discussed in the Nastran AIM attributes section.

```
select edge 1
attribute capsGroup $bar1
select edge 2
attribute capsGroup $bar2
select edge 3
attribute capsIgnore $multipleEdge
select edge 4
attribute capsGroup $bar3
```

Finally the nodes are attributed. In this case the three nodes across the top are given the same `capsConstraint` name. They could be assigned a different name allowing the user to define a different boundary condition at each location. The lower node is given a different `capsLoad` name so a load can be applied.

```
select node 1
attribute capsConstraint $boundary
select node 2
attribute capsLoad $force
select node 3
attribute capsConstraint $boundary
select node 4
attribute capsConstraint $boundary
```

The following input defines a pyCAPS input that can be used along with the above ∗.csm input to create a nastran input. First the pyCAPS and os module needs to be imported.

```
# Import pyCAPS class file
from pyCAPS import capsProblem

# Import os module
try:
    import os
except:
    print ("Unable to import os module")
    raise SystemError
```

Note if your Python major version is less than 3 (i.e. Python 2.7) the following statement should also be included so that print statements work correctly.

```
from __future__ import print_function
```

Once the modules have been loaded the problem needs to be initiated.

```
# Initialize capsProblem object
myProblem = capsProblem()
```

Next the ∗.csm file is loaded. Though not shown in this example the user has access to the X, Y and Z depmtr inputs from this pyCAPS script.

```
# Load CSM file
myProblem.loadCAPS("./csmData/feaThreeBar.csm")
```

The Nastran AIM is then loaded with the capsIntent set to STRUCTURE (this is consistent with the fidelity specified above in the ∗.csm file.

```
# Load nastran aim
nastranAIM = myProblem.loadAIM(aim = "nastranAIM",
                               altName = "nastran",
                               analysisDir= "NastranThreeBar",
                               capsIntent = "STRUCTURE")
```

After the AIM is loaded some of the inputs to the AIM are defined. A full list of options can be found in the AIM Inputs section. In this case the `Proj_Name` is entered. The project name is becomes the Nastran input file names. Two are create `projectName.bdf` and `projectName.dat`. The `*.bdf` file contains the grid and connectivity information. The data file contains the case control and other bulk data inputs required by Nastran. The input format is selected as `Free` and large field format is used when the option is available. This is most likely in the GRID entries only. Additionally the analysis type selected is `Static`. The maximum and minimum points that can be placed along an edge is set to be two. This ensures that each edge shown in the figure will be represented by a single finite element bar.

```
# Set project name so a mesh file is generated
projectName = "threebar_nastran_Test"
nastranAIM.setAnalysisVal("Proj_Name", projectName)
nastranAIM.setAnalysisVal("File_Format", "Free")
nastranAIM.setAnalysisVal("Mesh_File_Format", "Large")
nastranAIM.setAnalysisVal("Edge_Point_Max", 2);
nastranAIM.setAnalysisVal("Edge_Point_Min", 2);
nastranAIM.setAnalysisVal("Analysis_Type", "Static");
```

Next the material inputs, property selection, constraints and loads are defined. First materials are defined.

```
madeupium    = {"materialType" : "isotropic",
                "youngModulus" : 1.0E7 ,
                "poissonRatio" : .33,
                "density"      : 0.1}

nastranAIM.setAnalysisVal("Material", [("Madeupium", madeupium)])
```

Next these materials are used in the property definition. In this case two `bar` type properties are assigned to the edges. The outer bars have a property with a different area then the center bar. Note the relationship of `bar1` etc. between this pyCAPS input and the *,csm input previously shown.

```
rod  =   {"propertyType"      : "Rod",
          "material"          : "Madeupium",
          "crossSecArea"      : 1.0}

rod2  =   {"propertyType"     : "Rod",
          "material"          : "Madeupium",
          "crossSecArea"      : 2.0}

nastranAIM.setAnalysisVal("Property", [("bar1", rod),
                                       ("bar2", rod2),
                                       ("bar3", rod)])
```

Next the three nodes with `capsConstraint` boundary are constrained in all six degrees of freedom.

```
constraint = {"groupName"       : ["boundary"],
              "dofConstraint"   : 123456}

nastranAIM.setAnalysisVal("Constraint", ("BoundaryCondition", constraint))
```

Finally a load is applied the the node with the `capsLoad` force.

```
load = {"groupName"         : "force",
        "loadType"          : "GridForce",
        "forceScaleFactor"  : 20000.0,
        "directionVector"   : [0.8, -0.6, 0.0]}

nastranAIM.setAnalysisVal("Load", ("appliedForce", load ))
```

Finally an analysis case is defined that connects an analysis type to the load and constraint condition by name.

```
value = {"analysisType"         : "Static",
         "analysisConstraint"   : "BoundaryCondition",
         "analysisLoad"         : "appliedForce"}

myProblem.analysis["nastran"].setAnalysisVal("Analysis", ("SingleLoadCase", value ))
```

Once all the inputs have been set, aimPreanalysis needs to be executed. During this operation all the necessary files to run Nastran are generated and placed in the analysis working directory (analysisDir)

```
nastranAIM.aimPreAnalysis()
```

An OS system call is then made from Python to execute Nastran.

```
print ("\n\nRunning Nastran......")
currentDirectory = os.getcwd() # Get our current working directory
os.chdir(nastranAIM.analysisDir) # Move into test directory
os.system("nastran old=no notify=no batch=no scr=yes sdirectory=./ " + projectName +  ".dat"); # Run
       Nastran via system call
os.chdir(currentDirectory) # Move back to working directory
print ("Done running Nastran!")
```

A call to aimPostanalysis is then made to check to see if AVL executed successfully and the expected files were generated.

```
nastranAIM.aimPostAnalysis()
```

Finally the session is closed

```
myProblem.closeCAPS()
```

19.2 Multiple Load/Boundary Case Example

To create multiple load cases with different boundary conditions the pyCAPS input for constraints, and loads changes with respect to the Single Load Case Example. In addition an analysis section is added.

The constraint section may expand to allow multiple boundary conditions. In this way each load case can have a seperate boundary condition. If the input is left identical to the single load case example then the same boundary condition will be applied to each load case.

```
constraints = []

constraint = {"groupName"       : ["boundary"],
              "dofConstraint"   : 123456}
tmp = ( "conOne", constraint )
constraints.append( tmp )

constraint = {"groupName"       : ["boundary"],
              "dofConstraint"   : 123}
tmp = ( "conTwo", constraint )
constraints.append( tmp )

nastranAIM.setAnalysisVal("Constraint", constraints)
```

Notice that an empty `constraints` variable has been defined. Then a `tmp` tuple is created with the name `"conOne"` paired with the dictionary `constraint`. This tuple is appended to the empty `constraints` variable. The process is repeated for the second boundary condition. Then the AIM input `"Constraint"` is defined with the information.

Next the load input is expanded to contain muliple cases.

```
loads = []

load = {"groupName"         : "force",
        "loadType"          : "GridForce",
        "forceScaleFactor"  : 20000.0,
        "directionVector"   : [0.8, -0.6, 0.0]}
loads.append( ("loadOne", load) )

load = {"groupName"         : "force",
        "loadType"          : "GridForce",
        "forceScaleFactor"  : 20000.0,
        "directionVector"   : [-0.8, -0.6, 0.0]}
loads.append( ("loadTwo", load) )

nastranAIM.setAnalysisVal("Load", loads)
```

The process is identical to the constraint input.

Finally analysis cases are defined that connect an analysis type to a load and constraint condition by name.

```
analysisCases = []

value = {"analysisType"         : "Static",
         "analysisConstraint"   : "conOne",
         "analysisLoad"         : "loadOne"}
analysisCases.append( ("analysisOne", value ) )

value = {"analysisType"         : "Static",
         "analysisConstraint"   : "conTwo",
         "analysisLoad"         : "loadTwo"}
analysisCases.append( ("analysisTwo", value ) )

myProblem.analysis["nastran"].setAnalysisVal("Analysis", analysisCases)
```

Notice how the tuple names `"conOne"`, `"loadOne"` and `"analysisOne"` are all tied together. The `"analysisOne"` string also becomes the case control `LABEL` for the load case in the Nastran input file.

To finish the pyCAPS input the process starting with the pre-analysis input is identical to the Single Load Case Example input.

19.3 Modal Analysis Example Case

To create input for a modal analysis a two simple changes are required to the Single Load Case Example input. The first change is to the AIM Inputs `Analysis_Type`. This input is the last input in the list below.

```
# Set project name so a mesh file is generated
projectName = "threebar_nastran_Test"
nastranAIM.setAnalysisVal("Proj_Name", projectName)
nastranAIM.setAnalysisVal("File_Format", "Free")
nastranAIM.setAnalysisVal("Mesh_File_Format", "Large")
nastranAIM.setAnalysisVal("Edge_Point_Max", 2);
nastranAIM.setAnalysisVal("Edge_Point_Min", 2);
nastranAIM.setAnalysisVal("Analysis_Type", "Modal");
```

A description of each of these inputs can be found in Single Load Case Example.

The second change is replacing the load case information with a definition for the `Analysis` AIM input.

```
eigen = { "extractionMethod"     : "Lanczos",
          "frequencyRange"       : [0, 10000],
          "numEstEigenvalue"     : 1,
          "numDesiredEigenvalue" : 10,
          "eigenNormaliztion"    : "MASS"}

nastranAIM.setAnalysisVal("Analysis", ("EigenAnalysis", eigen))
```

This information defines the eigenvalue solver method and parameters and assigns it as an analysis case.

19.4 Optimization Example Case

This section creates a design model out of the single load case example. The first change is an update to the `Analysis_Type` AIM Input to `StaticOpt`.

```
# Set project name so a mesh file is generated
projectName = "threebar_nastran_Test"
nastranAIM.setAnalysisVal("Proj_Name", projectName)
nastranAIM.setAnalysisVal("File_Format", "Free")
nastranAIM.setAnalysisVal("Mesh_File_Format", "Large")
nastranAIM.setAnalysisVal("Edge_Point_Max", 2);
nastranAIM.setAnalysisVal("Edge_Point_Min", 2);
nastranAIM.setAnalysisVal("Analysis_Type", "StaticOpt");
```

The next update adds a material allowable to the material input `yieldAllow`. This is not a requirement for THIS optimization problem, but this input is referenced when design constraints are added later.

```
madeupium    = {"materialType" : "isotropic",
                "youngModulus" : 1.0E7 ,
                "poissonRatio" : .33,
                "density"      : 0.1,
                "yieldAllow"   : 5.6E7}

nastranAIM.setAnalysisVal("Material", [("Madeupium", madeupium)])
```

The first large optimization input is the design variable definition section. For more information the user is pointed to the FEA DesignVariable section. In this section the area of each rod element in the three bar truss is defined as a separate design variable. Finally each of these variables are used to defined AIM Input `Design_Variable`

```
DesVar1    = {"groupName" : "bar1",
              "initialValue" : rod["crossSecArea"],
              "lowerBound" : rod["crossSecArea"]*0.5,
              "upperBound" : rod["crossSecArea"]*1.5,
              "maxDelta"   : rod["crossSecArea"]*0.1,
              "fieldName" : "A"}

DesVar2    = {"groupName" : "bar2",
              "initialValue" : rod2["crossSecArea"],
              "lowerBound" : rod2["crossSecArea"]*0.5,
              "upperBound" : rod2["crossSecArea"]*1.5,
              "maxDelta"   : rod2["crossSecArea"]*0.1,
              "fieldName" : "A"}

DesVar3    = {"groupName" : "bar3",
              "initialValue" : rod["crossSecArea"],
              "lowerBound" : rod["crossSecArea"]*0.5,
              "upperBound" : rod["crossSecArea"]*1.5,
              "maxDelta"   : rod["crossSecArea"]*0.1,
              "fieldName" : "A"}
```

```
myProblem.analysis["nastran"].setAnalysisVal("Design_Variable", [("Bar1A", DesVar1),
                                                                 ("Bar2A", DesVar2),
                                                                 ("Bar3A", DesVar3)])
```

The next unique section is the addition of design constraints. In this problem stress constraints in each rod element are added.

```
designConstraint1 = {"groupName" : "bar1",
                     "responseType" : "STRESS",
                     "lowerBound" : -madeupium["yieldAllow"],
                     "upperBound" :  madeupium["yieldAllow"]}

designConstraint2 = {"groupName" : "bar2",
                     "responseType" : "STRESS",
                     "lowerBound" : -madeupium["yieldAllow"],
                     "upperBound" :  madeupium["yieldAllow"]}

designConstraint3 = {"groupName" : "bar3",
                     "responseType" : "STRESS",
                     "lowerBound" : -madeupium["yieldAllow"],
                     "upperBound" :  madeupium["yieldAllow"]}

myProblem.analysis["nastran"].setAnalysisVal("Design_Constraint",[("stress1", designConstraint1),
                                                                  ("stress2", designConstraint2),
                                                                  ("stress3", designConstraint3)])
```

This completes the unique parts of the design inputs required for a Nastran optimization problem.

19.5 Composite Wing Example

This example introduces the use of composite materials. Initially a composite wing frequency analysis is completed. This example will grow to introduce design optimization with composites, including design variable linking.

Figure 2: Composite Wing Example

First step is to define the analysis fidelity that the geometry is intended support. In this case the fidelity is `STRUCTURE`.

```
attribute capsIntent STRUCTURE
```

The parameter being set in this case is a definition for a coordinate system in Engineering Sketch Pad. The documentation for `csystem` inputs is as follows.

```
CSYSTEM   $csysName csysList
        use:    attach a Csystem to Body on top of stack
        pops:   any
        pushes: any
        notes:  Sketch may not be open
                if     csysList contains 9 entries:
                   {x0, y0, z0, dx1, dy1, dz1, dx2, dy2, dz2}
                   origin is at (x0,y0,q0)
                   dirn1  is in (dx1,dy1,dz1) direction
                   dirn2  is part of (dx2,dy2,dz2) that is orthog. to dirn1
                elseif csysList contains 5 entries and first is positive
                   {+iface, ubar0, vbar0, du2, dv2}
                   origin is at normalized (ubar0,vbar0) in iface
                   dirn1  is normal to Face
```

```
   dirn2  is in (du2,dv2) direction
elseif csyList contains 5 entries and first is negative
   {-iedge, tbar, dx2, dy2, dz2}
   origin is at normalized (tbar) in iedge
   dirn1  is tangent to Edge
   dirn2  is part of (dx2,dy2,dz2) that is orthog. to dirn1
elseif csysList contains 7 entries
   {inode, dx1, dy1, dz1, dx2, dy2, dz2}
   origin is at Node inode
   dirn1  is in (dx1,dy1,dz1) direction
   dirn2  is part of (dx1,dy2,dz2) that is orthog. to dirn1
else
   error
semicolon-sep lists can instead refer to
   multi-valued Parameter
dirn3 is formed by (dirn1)-cross-(dirn2)
does not create a Branch
```

In the `compositesys` parameter defined below 9 entries are given. Based on the documentation above this indicates the following.

Origin	0.0	5.5	0.↩ 0
Vector along x-Axis	79.3685	-0.65432	0.↩ 0
Vector along y-Axis	0.65432	79.3685	0.↩ 0

It should be noted that the vector along the y-axis may not be input perfectly perpendicular to the vector along the x-Axis. In this case ESP takes the projection of the input vector that is in the plane defined by both input vectors and perpendicular to the x-Axis. This is the case for all csystem input options defined above.

```
dimension compositesys   9   1   0
set       compositesys 0;5.5;0;79.3685;-0.65432;0;0.65432;79.3685;0
```

The geometry definition was generated by the ESP sketcher. Users are referred to ESP tutorials for information on how to create a sketch. The result is copy and pasted into the `*.csm` file snippet shown below.

```
skbeg     0   0   0   1
   skvar     xy
       -0.024750;0.051384;4.841311;-0.024750;-3.895337;0.000000;82.067045;-3.895337;0.000000;77.5
      94095;3.314007;0.000000;4.132462;13.891219;0.000000;-2.340160;13.680727;0.000000;-1.235078;10.891711;0.000000;
   skcon     X   1   -1  0
   skcon     Y   1   -1  0
   skcon     V   1   2  0
   skcon     L   3   4  8.5
   skcon     R   7   1  5.5
   skcon     H   2   3  0
   skcon     L   1   2  3.91
   skcon     L   6   7  3
   skcon     L   5   6  6.5
   skcon     L   4   5  74.2
   skcon     L   2   3  82.1
   skcon     P   1   -1  0
   skcon     A   5   -1  10
   skcon     A   6   -1  110
   skcon     A   4   -1  50
   linseg    ::x[2]   ::y[2]   0
   linseg    ::x[3]   ::y[3]   0
   linseg    ::x[4]   ::y[4]   0
   linseg    ::x[5]   ::y[5]   0
   linseg    ::x[6]   ::y[6]   0
   linseg    ::x[7]   ::y[7]   0
   arc       ::x[1]   ::y[1]   0   ::d[1]   xy
skend     0
```

The root edges are marked as `capsConstraint` locations. The overall surfaces is given a `capsGroup` and `capsLoad`attribute and a `csystem` definition is attached to it using the `compositesys` parameter previously discussed.

```
attribute capsGroup $wing
attribute capsLoad  $wing
attribute capsBound $wing
csystem   wing   compositesys
select    edge   7
attribute capsConstraint   $root
select    edge   6
attribute capsConstraint   $root
select    edge   1
attribute capsConstraint   $root
```

This model was created in centimeters and is converted to inches.

```
scale 1/2.54
```

Note if your Python major version is less than 3 (i.e. Python 2.7) the following statement should also be included so that print statements work correctly.

The following input defines a pyCAPS input that can be used along with the above ∗.csm input to create a nastran input. First the pyCAPS and os module needs to be imported.

Once the modules have been loaded the problem needs to be initiated.

Next the ∗.csm file is loaded. Though not shown in this example the user has access to the X, Y and Z depmtr inputs from this pyCAPS script.

The Nastran AIM is then loaded with the capsIntent set to STRUCTURE (this is consistent with the fidelity specified above in the ∗.csm file.

After the AIM is loaded some of the inputs to the AIM are defined. A full list of options can be found in the AIM Inputs section. In this case the `Proj_Name` is entered. The project name is becomes the Nastran input file names. Two are create `projectName.bdf` and `projectName.dat`. The `∗.bdf` file contains the grid and connectivity information. The data file contains the case control and other bulk data inputs required by Nastran. The input format is selected as `Small` and `Large` field format is used when the option is available. This is most likely in the GRID entries only. Additionally the analysis type selected is `Static`. The maximum points that can be placed along an edge is set to be 40.

In this example two materials are defined. This demonstrates how simple it is to change materials in a model. Both an aluminum and Graphite_expoxy material are defined in the `Material` AIM Inputs.

Again property information is defined for both an aluminum and composite stack version of the model. However, only the composite entry is defined in the `Property` AIM Inputs.

The `composite` definition brings together the materials in each layer of the stack, their thicknesses and orientations. For the sequence defined below the order of the sequence is given in the table below. The full sequence is given to point out that they symmetry condition is applied to the right side of the `compositeOrientation` input.

0	0	0	0	-45	45	-45	45	45	-45	45	-45	0	0	0	0

Finally the property is assigned to the regions with the `capsGroup $wing` attribute.

In this example the root edges are constrained in all degrees of freedom. This constraint references the `caps↩ Constraint $root` input defined in the ∗.csm file.

As previously shown in Modal Analysis Example Case information to define an eigen value problem is entered and the `Analysis` AIM Input is defined.

Finally preAnalysis is executed to generate all the required Nastran inputs.

Nastran is executed with a simple system call.

A post analysis command is entered allowing the user to access output data, if desired, from the application.

The caps session is closed.

19.6 Composite Wing Optimization Example

This section removes the frequency analysis and adds a pressure load to the previously introduced composite wing example case. Then An optimization problem allowing the thickness of each ply layer to change is performed.

References

[1] Michael Reymond and Mark Miller. *MSC NASTRAN Quick Reference Guide Version 68*, 1996. 1