

# Computational Aircraft Prototype Syntheses



## Training Session 9 Structures Analysis: ASTROS/NASTRAN/TACS ESP v1.22

**Marshall Galbraith**

[galbramc@mit.edu](mailto:galbramc@mit.edu)

Massachusetts Institute of Technology

**Bob Haimes**

[haimes@mit.edu](mailto:haimes@mit.edu)

**John F. Dannenhoffer, III**

[jfdannen@syr.edu](mailto:jfdannen@syr.edu)

- Structural analysis
  - ASTROS
  - NASTRAN
  - TACS
- Modal analysis using ASTROS
  - Cantilever
  - Support node
- Static analysis using ASTROS
  - Cantilever
  - Orthotropic materials and coordinate systems
- Flutter analysis with NASTRAN
- Static analysis using TACS (Sean Engelstad, Georgia Tech)
  - Sizing optimization
  - Shape optimization

## Automated Structural Optimization System

- ZONA Technology, Inc.
  - Originally developed by Northrop Corporation under contract with AF Wright Aeronautical Laboratories
- Structural Modal and Static Analysis
- Aerodynamic Loads (Vortex Lattice Method)
- Aeroelastic Stability and Trim
- Control System Interaction
- Structural Sizing Optimization
- Sensitivity Analysis

## micro-ASTROS (mASTROS)

- Limited mesh sizes
- No aerodynamic analysis

## NASA STRucture ANalysis

- MSC Software Corporation (MSC Nastran)
  - Originally developed for NASA in late 1960s by MSC
  - AutoDesk NEi Software (NEi Nastran)
  - Siemens PLM Software (NX Nastran)
  - Open source (<https://github.com/nasa/NASTRAN-95>)
- Structural Modal and Static Analysis
- Aerodynamic Loads (Vortex Lattice Method)
- Aeroelastic Stability and Trim
- Structural Assembly Modeling
- Automated Structural Optimization



## Toolkit for the Analysis of Composite Structures

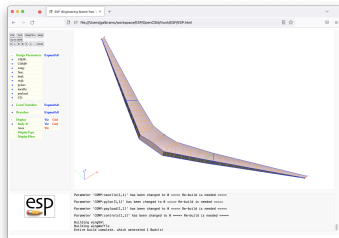
- Georgia Tech and University of Michigan
- Structural Static and Unsteady Analysis
- Structural Optimization Sensitivity Calculations
- Open source (<https://github.com/smdogroup/tacs>)
- Currently not available on Windows

## ASTROS AIM Documentation

- Cantilever root constraint
- Support node at root
- Box structure with spars and ribs

ESP/transport.csm

CFGPMTR	VIEW:Cantilever	0
CFGPMTR	VIEW:SimplySupport	0



- Use egadsTessAIM for quad surface tessellation
- Coarsest possible grid for expedience
- Consistent for all examples in this session

## session09/1\_astros\_ModalCantilever.py

---

```
# Create EGADS tess aim
tess = capsProblem.analysis.create(aim = "egadsTessAIM",
                                   name = "tess")

# No Tess vertexes on edges (minimal mesh)
tess.input.Edge_Point_Min = 2
tess.input.Edge_Point_Max = 2

# Use regularized quads
tess.input.Mesh_Elements = "Quad"
```

---

- Cantilever constraint on wing root rib  
FACE/EDGE/NODE using capsConstraint

ESP/viewCantilever.udc

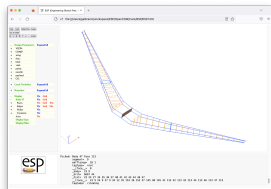
---

```
UDPRIM      editAttr  filename <<
FACE HAS    tagComp=leftWing tagType=root
SET         capsConstraint=rootConstraint

FACE HAS    tagComp=riteWing tagType=root
SET         capsConstraint=rootConstraint

EDGE ADJ2FACE tagComp=leftWing tagType=root
SET         capsConstraint=rootConstraint
```

---



session09/1\_astros\_ModalCantilever.py

---

```
# Set geometry variables to enable Box Structure with Clamped root
wing.cfgpmtr.VIEW.Concept      = 0
wing.cfgpmtr.VIEW.Cantilever   = 1
```

---

```
# Set constraints
constraint = {"dofConstraint" : 123456}
astros.input.Constraint = {"rootConstraint": constraint}
```

---

- Specify the type of analysis
- Build material database

## session09/1\_astros\_ModalCantilever.py

```
# Set analysis type
eigen = { "analysisType"      : "Modal",
          "extractionMethod"  : "SINV",
          "frequencyRange"    : [0, 10],
          "numEstEigenvalue"   : 1,
          "numDesiredEigenvalue" : 10,
          "eigenNormaliztion"  : "MASS"}

astros.input.Analysis = {"EigenAnalysis": eigen}

# Set materials
unobtainium = {"youngModulus" : 2.2E6 ,
               "poissonRatio" : .5,
               "density"       : 7850}

madeupium   = {"materialType" : "isotropic",
               "youngModulus"  : 1.2E5 ,
               "poissonRatio"  : .5,
               "density"       : 7850}

astros.input.Material = {"Unobtainium": unobtainium,
                         "Madeupium"  : madeupium}
```

- Define shell properties to associate with capsGroup
- Properties connected to materials via their name

## session09/1\_astros\_ModalCantilever.py

```
# Set properties
skinShell = {"propertyType" : "Shell",
             "material"      : "unobtainium",
             "bendingInertiaRatio" : 1.0,
             "shearMembraneRatio" : 0, # Turn of shear - no materialShear
             "membraneThickness" : 0.05}

ribShell = {"propertyType" : "Shell",
            "material"      : "unobtainium",
            "bendingInertiaRatio" : 1.0,
            "shearMembraneRatio" : 0, # Turn of shear - no materialShear
            "membraneThickness" : 0.1}

sparShell = {"propertyType" : "Shell",
             "material"      : "madeupium",
             "bendingInertiaRatio" : 1.0,
             "shearMembraneRatio" : 0, # Turn of shear - no materialShear
             "membraneThickness" : 0.2}

astros.input.Property = {"leftWingSkin" : skinShell,
                        "riteWingSkin" : skinShell,
                        "wingRib"      : ribShell,
                        "wingSpar1"    : sparShell,
                        "wingSpar2"    : sparShell}
```

- Associate shell properties with capsGroup

ESP/viewCantilever.udc

---

```

FACE    HAS    tagComp=leftWing    tagType=upper
SET      capsGroup=leftWingSkin  capsBound=upperWing

FACE    HAS    tagComp=riteWing    tagType=upper
SET      capsGroup=riteWingSkin  capsBound=upperWing

FACE    HAS    tagComp=leftWing    tagType=lower
SET      capsGroup=leftWingSkin  capsBound=lowerWing

FACE    HAS    tagComp=riteWing    tagType=lower
SET      capsGroup=riteWingSkin  capsBound=lowerWing

```

---

session09/1\_astros\_ModalCantilever.py

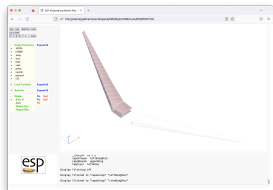
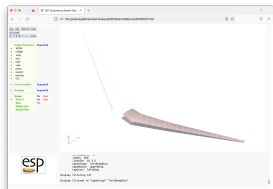
---

```

astros.input.Property = {"leftWingSkin": skinShell,
                        "riteWingSkin": skinShell,
                        "wingRib"      : ribShell,
                        "wingSpar1"    : sparShell,
                        "wingSpar2"    : sparShell}

```

---

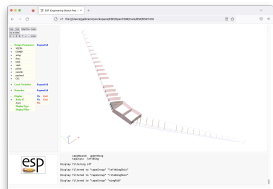




- Associate shell properties with capsGroup

ESP/viewCantilever.udc

FACE SET	HAS	tagType=rib capsGroup= <u>wingRib</u>	
FACE SET	HAS	tagComp=leftWing capsGroup= <u>wingRib</u>	tagType=tip capsBound=leftTip
FACE SET	HAS	tagComp=riteWing capsGroup= <u>wingRib</u>	tagType=tip capsBound=riteTip
FACE SET	HAS	tagComp=leftWing capsGroup= <u>wingRib</u>	tagType=fusespar
FACE SET	HAS	tagComp=riteWing capsGroup= <u>wingRib</u>	tagType=fusespar



session09/1\_astros\_ModalCantilever.py

```
astros.input.Property = {"leftWingSkin": skinShell,
                        "riteWingSkin": skinShell,
                        "wingRib"      : ribShell,
                        "wingSpar1"    : sparShell,
                        "wingSpar2"    : sparShell}
```

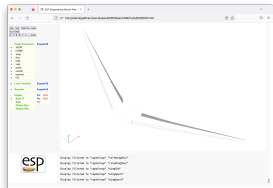
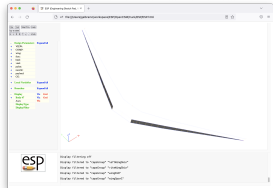
- Associate shell properties with capsGroup

ESP/viewCantilever.udc

FACE	HAS	tagType=spar	tagIndex=1
SET		capsGroup= <u>wingSpar1</u>	capsMesh= <u>wingSpar1</u>
<hr/>			
FACE	HAS	tagType=spar	tagIndex=2
SET		capsGroup= <u>wingSpar2</u>	capsMesh= <u>wingSpar2</u>

session09/1\_astros\_ModalCantilever.py

```
astros.input.Property = {"leftWingSkin": skinShell,  
                        "riteWingSkin": skinShell,  
                        "wingRib"      : ribShell,  
                        "wingSpar1"    : sparShell,  
                        "wingSpar2"    : sparShell}
```



- ASTROS requires “ASTRO.D01” and “ASTRO.IDX” in run directory

## session09/1\_astros\_ModalCantilever.py

```
# Declare ASTROS install directory
astrosInstallDir = os.environ['ESP_ROOT'] + os.sep + "bin" + os.sep

# Copy files needed to run astros
files = ["ASTRO.D01", "ASTRO.IDX"]
for file in files:
    try:
        shutil.copy2(astrosInstallDir + file, astros.analysisDir + os.sep + file)
    except:
        print ('Unable to copy ' + file + ' ')
        raise

# Run micro-ASTROS via system call
print ("\n==> Running MASTROS...")
astros.system("mastros.exe < " + astros.input.Proj_Name + ".dat > " + astros.input.Proj_Name + ".out")

# Remove temporary files
for file in files:
    if os.path.isfile(astros.analysisDir + os.sep + file):
        os.remove(astros.analysisDir + os.sep + file)
```

- Print the Eigen frequencies

session09/1\_astros\_ModalCantilever.py

---

```
# Get list of Eigen-frequencies
freqs = astros.output.EigenFrequency

print ("\n--> Eigen-frequencies:")
for i in range(len(freqs)):
    print ("      " + repr(i+1).ljust(2) + ": " + str(freqs[i]))
```

---

- Support connection on wing root rib FACE/EDGE/NODE using capsConnectLink and capsConnect

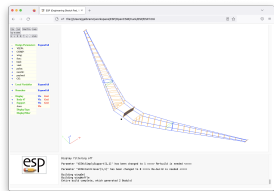
ESP/viewSimplySupport.udc

```
FACE HAS      tagComp=leftWing tagType=root
SET           capsConnectLink=ribRoot
```

```
FACE HAS      tagComp=riteWing tagType=root
SET           capsConnectLink=ribRoot
```

```
EDGE ADJ2FACE tagComp=leftWing tagType=root
SET           capsConnectLink=ribRoot
```

```
POINT wing:xroot+wing:chordr/4 wing:yroot wing:zroot
ATTRIBUTE capsConnect      $ribRoot
ATTRIBUTE capsGroup        $ribSupport
ATTRIBUTE capsConstraint    $ribRootPoint
```



session09/2\_astros\_ModalSupport.py

```
wing.cfcpmtr.VIEW.Concept      = 0
wing.cfcpmtr.VIEW.SimplySupport = 1
```

- Define connection and support types
- Define constraint on the support node

## ESP/viewSimplySupport.udc

```
POINT wing:xroot+wing:chordr/4 wing:yroot wing:zroot
ATTRIBUTE capsConnect      $ribRoot
ATTRIBUTE capsGroup        $ribSupport
ATTRIBUTE capsConstraint    $ribRootPoint
```

## session09/2\_astros\_ModalSupport.py

```
# Defined Connections
connection = {"dofDependent" : 123456,
              "connectionType" : "RigidBody"}
astros.input.Connect = {"ribRoot": connection}

# Set supports
support = {"dofSupport": 3}
astros.input.Support = {"ribRootPoint": support}

# Set constraints
constraint = {"dofConstraint" : 12456}
astros.input.Constraint = {"ribRootPoint": constraint}
```

- Concentrated mass with moments of inertia on support node

ESP/viewSimplySupport.udc

---

```
POINT wing:xroot+wing:chordr/4 wing:yroot wing:zroot
  ATTRIBUTE capsConnect      $ribRoot
  ATTRIBUTE capsGroup        $ribSupport
  ATTRIBUTE capsConstraint    $ribRootPoint
```

---

session09/2\_astros\_ModalSupport.py

---

```
mass = {"propertyType" : "ConcentratedMass",
        "mass"         : 1.0E5,
                     #I11 I12 I22 I31 I32 I33
        "massInertia"  : [0.0, 0.0, 1.0E5, 0.0, 0.0, 0.0]}

astros.input.Property = {"leftWingSkin" : skinShell,
                          "riteWingSkin" : skinShell,
                          "wingRib"      : ribShell,
                          "wingSpar1"    : sparShell,
                          "wingSpar2"    : sparShell,
                          "ribSupport"    : mass }
```

---

- Structural analysis
  - ASTROS
  - NASTRAN
  - TACS
- Modal analysis using ASTROS
  - Cantilever
  - Support node
- Static analysis using ASTROS
  - Cantilever
  - Orthotropic materials and coordinate systems
- Flutter analysis with NASTRAN
- Static analysis using TACS (Sean Engelstad, Georgia Tech)
  - Sizing optimization
  - Shape optimization

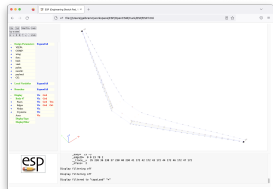


- Static loads can be applied to entities marked with capsLoad
- capsLoad on NODE → point load
- capsLoad on EDGE → multi-point load (future linear load)
- capsLoad on FACE → pressure load

ESP/viewCantilever.udc

```
UDPRIM      editAttr filename <<
NODE ADJ2FACE tagComp=leftWing tagType=tip
AND  ADJ2FACE tagType=lower
AND  ADJ2FACE tagType=spar      tagIndex=1
SET      capsLoad=leftPointLoad

NODE ADJ2FACE tagComp=riteWing tagType=tip
AND  ADJ2FACE tagType=lower
AND  ADJ2FACE tagType=spar      tagIndex=1
SET      capsLoad=ritePointLoad
```



session09/3\_astros\_StaticCantilever.py

```
wing.cfgpmtr.VIEW.Concept      = 0
wing.cfgpmtr.VIEW.Cantilever   = 1
```

- Define loads to apply, and set analysis to static

## ESP/viewCantilever.udc

```
UDPRIM      editAttr filename <<
  NODE ADJ2FACE tagComp=leftWing tagType=tip
  AND  ADJ2FACE tagType=lower
  AND  ADJ2FACE tagType=spar      tagIndex=1
  SET   capsLoad=leftPointLoad

  NODE ADJ2FACE tagComp=riteWing tagType=tip
  AND  ADJ2FACE tagType=lower
  AND  ADJ2FACE tagType=spar      tagIndex=1
  SET   capsLoad=ritePointLoad
```

## session09/3\_astros\_StaticCantilever.py

```
# Define loads
leftLoad = {"loadType"      : "GridForce",
            "forceScaleFactor" : 1.e6,
            "directionVector" : [0.0, 0.0, 1.0]}
riteLoad = {"loadType"      : "GridForce",
            "forceScaleFactor" : 2.e6,
            "directionVector" : [0.0, 0.0, 1.0]}
astros.input.Load = {"leftPointLoad": leftLoad,
                    "ritePointLoad": riteLoad}

# Set analysis type
astros.input.Analysis_Type = "Static"
```

- Print the displacements

session09/3\_astros\_StaticCantilever.py

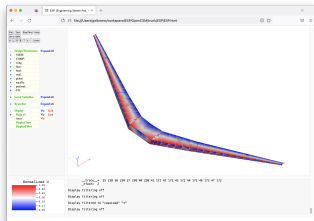
```
print ("\n--> Maximum displacements:")
print ("--> Tmax" , astros.output.Tmax )
print ("--> T1max", astros.output.T1max)
print ("--> T2max", astros.output.T2max)
print ("--> T3max", astros.output.T3max)
```

- Coordinate system name matches capsGroup

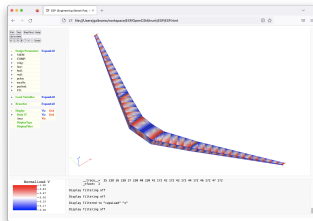
ESP/viewCantilever.udc

```

CSYSTEM leftWingSkin      # iface;ubar0;vbar0;du2;dv2
CSYSTEM leftWingSkinCsys;0;0;0;1
CSYSTEM riteWingSkin      riteWingSkinCsys;0;0;0;1
  
```



Normalized u-coordinates

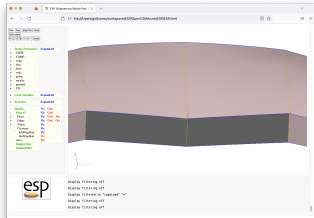


Normalized v-coordinates

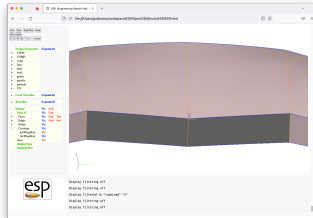
- Coordinate system name matches capsGroup

ESP/viewCantilever.udc

```
# iface;ubar0;vbar0;du2;dv2
CSYSTEM leftWingSkin leftWingSkinCsys;0;0;0;1
CSYSTEM riteWingSkin riteWingSkinCsys;0;0;0;1
```



leftWingSkin coordinate system



riteWingSkin coordinate system

- Define isotropic and orthotropic materials

session09/4\_astros\_Composite.py

```
# Set materials
Aluminum = {"youngModulus" : 10.5E6 ,
            "poissonRatio" : 0.3,
            "density"      : 0.1/386,
            "shearModulus" : 4.04E6}

Graphite_epoxy = {"materialType"      : "Orthotropic",
                  "youngModulus"      : 20.8E6 ,
                  "youngModulusLateral" : 1.54E6,
                  "poissonRatio"      : 0.327,
                  "shearModulus"      : 0.80E6,
                  "density"            : 0.059/386,
                  "tensionAllow"      : 11.2e-3,
                  "tensionAllowLateral" : 4.7e-3,
                  "compressAllow"      : 11.2e-3,
                  "compressAllowLateral" : 4.7e-3,
                  "shearAllow"         : 19.0e-3,
                  "allowType"          : 1}

astros.input.Material = {"Aluminum"      : Aluminum,
                        "Graphite_epoxy" : Graphite_epoxy}
```

- CompositeOrientation angles relative to CSYSTEM

session09/4\_astros\_Composite.py

---

```
skinShell = {"propertyType"      : "Composite",
             "shearBondAllowable" : 1.0e6,
             "bendingInertiaRatio" : 1.0,
             "shearMembraneRatio"  : 0, # Turn off shear - no materialShear
             "compositeMaterial"   : ["Graphite_epoxy"]*8,
             "compositeThickness"  : [0.00525]*8,
             "compositeOrientation" : [0, 0, 0, 0, -45, 45, -45, 45],
             "symmetricLaminate"   : True,
             "compositeFailureTheory" : "STRAIN" }
```

---

- Structural analysis
  - ASTROS
  - NASTRAN
  - TACS
- Modal analysis using ASTROS
  - Cantilever
  - Support node
- Static analysis using ASTROS
  - Cantilever
  - Orthotropic materials and coordinate systems
- Flutter analysis with NASTRAN
- Static analysis using TACS (Sean Engelstad, Georgia Tech)
  - Sizing optimization
  - Shape optimization

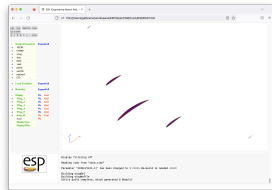


- Cantilever aeroelastic flutter analysis



Structural body with full skin

```
ATTRIBUTE capsAIM $nastranAIM
ATTRIBUTE capsDiscipline $Structure
```



VLM Aero bodies

```
ATTRIBUTE capsAIM $nastranAIM
ATTRIBUTE capsDiscipline $Aerodynamic
```

session09/5\_nastran\_Flutter.py

---

```
# Change to Structures and VLM
wing.cfgpmtr.VIEW.Concept      = 0
wing.cfgpmtr.VIEW.Vlm         = 1
wing.cfgpmtr.VIEW.Cantilever   = 1
```

---

- Aeroelastic analysis coupled via capsBound

ESP/viewCantilever.udc

```
FACE    HAS    tagComp=leftWing    tagType=upper
SET      capsGroup=leftWingSkin  capsBound=upperWing
```

session09/5\_nastran\_Flutter.py

```
# Note the surface name corresponds to the capsBound found in the *.csm file. This links
# the spline for the aerodynamic surface to the structural model
nastran.input.VLM_Surface = {"upperWing": wingVLM}
```

```
# Set analysis
flutter = { "analysisType"      : "AeroelasticFlutter",
            "extractionMethod"  : "Lanczos",
            "frequencyRange"    : [0.1, 300],
            "numEstEigenvalue"   : 4,
            "numDesiredEigenvalue" : 4,
            "eigenNormaliztion"  : "MASS",
            "aeroSymmetryXY"     : "ASYM",
            "aeroSymmetryXZ"     : "SYM",
            "analysisConstraint" : ["rootConstraint"],
            "machNumber"         : 0.1029,
            "dynamicPressure"    : 0.5* 1.2e-6 * 3500**2,
            "density"            : 1.2e-6,
            "reducedFreq"        : [0.001, 0.01, 0.1, 0.12, 0.14, 0.16, 0.18, 0.25],
            }
nastran.input.Analysis = {"Flutter": flutter}
```

- Structural analysis
  - ASTROS
  - NASTRAN
  - TACS
- Modal analysis using ASTROS
  - Cantilever
  - Support node
- Static analysis using ASTROS
  - Cantilever
  - Orthotropic materials and coordinate systems
- Flutter analysis with NASTRAN
- Static analysis using TACS (Sean Engelstad, Georgia Tech)
  - Sizing optimization
  - Shape optimization

- TACS setup identical to ASTROS
- TACS executes via Python API

## session09/6\_tacs\_StaticCantilever.py

---

```
#initialize pytacs with that data file (serial execution)
dat_file = os.path.join(tacs.analysisDir, tacs.input.Proj_Name+".dat")
fea_solver = pyTACS(dat_file)

# Set up TACS Assembler
fea_solver.initialize()

#read the bdf & dat file into pytacs FEAsolver
#SPs represents "StructuralProblems"
SPs = fea_solver.createTACSProbsFromBDF()

# add mass and aggregated stress constraint functions
function_names = ["mass", "ks_vmfailure"]
for caseID in SPs:
    SPs[caseID].addFunction('mass', functions.StructuralMass)
    SPs[caseID].addFunction('ks_vmfailure', functions.KSFailure, safetyFactor=1.5, ksWeight=1000.0)
```

---

- TACS setup identical to ASTROS
- TACS executes via Python API

## session09/6\_tacs\_StaticCantilever.py

```
# solve each structural analysis problem (in this case 1)
print ("\n==> Running TACS...")
tacs_funcs = {}
base_name = "tacs_output"
for caseID in SPs:
    SPs[caseID].solve()
    SPs[caseID].evalFunctions(tacs_funcs,evalFuncs=function_names)
    SPs[caseID].writeSolution(baseName=base_name, outputDir=tacs.analysisDir)

# Run AIM post-analysis
print ("\n==> Running TACS post-analysis...")
tacs.postAnalysis()

# print the output analysis functions and sensitivities
print("Tacs Analysis Outputs...")
for func_name in function_names:
    # find the associated tacs key (tacs key = (loadset) + (func_name))
    for tacs_key in tacs_funcs:
        if func_name in tacs_key:
            break

    func_value = tacs_funcs[tacs_key].real
    print(f"\tfunctional {func_name} = {func_value}")
```

- Optimize membrane thickness

session09/7\_tacs\_sizing.py

---

```
# make thickness variables
thickness = skinShell["membraneThickness"]
skinThick = {"initialValue" : thickness,
            "lowerBound"   : thickness*0.5,
            "upperBound"   : thickness*1.5,
            "maxDelta"     : thickness}

thickness = ribShell["membraneThickness"]
ribThick  = {"initialValue" : thickness,
            "lowerBound"   : thickness*0.5,
            "upperBound"   : thickness*1.5,
            "maxDelta"     : thickness}

thickness = sparShell["membraneThickness"]
sparThick = {"initialValue" : thickness,
            "lowerBound"   : thickness*0.5,
            "upperBound"   : thickness*1.5,
            "maxDelta"     : thickness}

tacs.input.Design_Variable = {"skinThick" : skinThick,
                              "ribThick"  : ribThick,
                              "sparThick" : sparThick}
```

---

- Optimize membrane thickness

## session09/7\_tacs\_sizing.py

```
tacs.input.Design_Variable = {"skinThick": skinThick,
                             "ribThick" : ribThick,
                             "sparThick": sparThick}

relLeftSkin = {"componentType": "Property",
               "componentName": "leftWingSkin",
               "fieldName"    : "T",
               "variableName" : "skinThick"}

relRiteSkin = {"componentType": "Property",
               "componentName": "riteWingSkin",
               "fieldName"    : "T",
               "variableName" : "skinThick"}

relRib      = {"componentType": "Property",
               "componentName": "wingRib",
               "fieldName"    : "T",
               "variableName" : "ribThick"}

relSpar      = {"componentType": "Property",
               "componentName": "wingSpar1",
               "fieldName"    : "T",
               "variableName" : "sparThick"}

tacs.input.Design_Variable_Relation = {"relLeftSkin": relLeftSkin,
                                       "relRiteSkin": relRiteSkin,
                                       "relRib"      : relRib,
                                       "relSpar"     : relSpar}
```

session09/7\_tacs\_sizing.py

---

```
tacs_funcs = {}
tacs_sens = {}
for caseID in self.SPs:

    # add functions for the struct problems
    self.SPs[caseID].addFunction('mass', functions.StructuralMass)
    self.SPs[caseID].addFunction('ks_vmfailure', functions.KSFailure, safetyFactor=1.5, ksWeight=50.0)

    # overwrite the thickness DVs
    xarray = self.SPs[caseID].x.getArray()
    for ithick, thick_dv in enumerate(self._thick_dvs):
        xarray[ithick] = float(inputs[thick_dv])
        print("-->",thick_dv,inputs[thick_dv])

    self.SPs[caseID].solve()
    self.SPs[caseID].evalFunctions(tacs_funcs, evalFuncs=self._func_names)
    self.SPs[caseID].evalFunctionsSens(tacs_sens, evalFuncs=self._func_names)
    self.SPs[caseID].writeSolution(baseName="tacs_output", outputDir=tacs.analysisDir)
    #print(tacs_sens)
```

---



- Create your own (optionally share it `galbramc@mit.edu`)