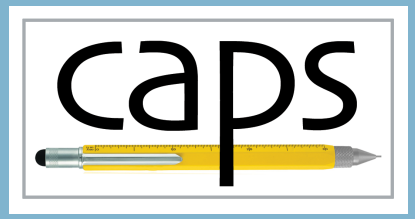
Computational Aircraft Prototype Syntheses



Training Session 7
Structures Analysis:
ASTROS/NASTRAN/TACS

ESP v1.26

Marshall Galbraith
galbramc@mit.edu

Bob Haimes
haimes@mit.edu

Massachusetts Institute of Technology

John F. Dannenhoffer, III
jfdannen@syr.edu

Overview

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

ASTROS Overview

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 Optmization
- Sensitivity Analysis

micro-ASTROS (mASTROS)

- Limited mesh sizes
- No aerodynamic analysis

NASTRAN Overview

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

TACS Overview

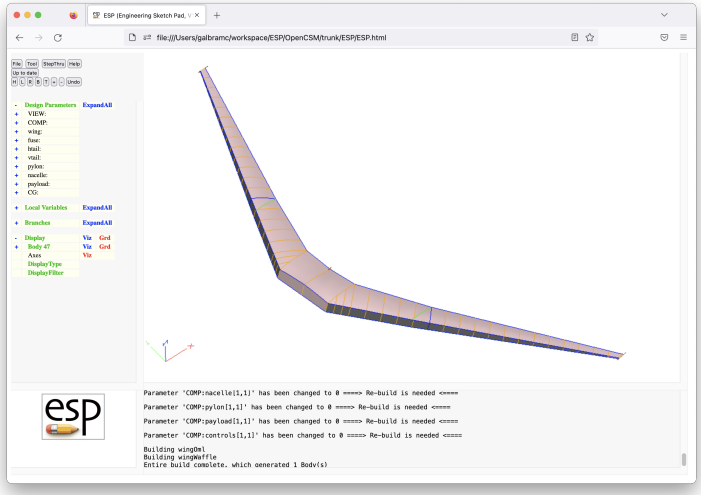
Toolkit for the Analysis of Composite Structures

- 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

transport.csm Geometry

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



ESP/transport.csm

```
CFGPMTR    VIEW:Cantilever        0
CFGPMTR    VIEW:SimplySupport     0
```

Structures Meshing

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

session07/1_astros_ModalCantilever.py

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

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

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

ASTROS Cantilever Modal Analysis: Setup

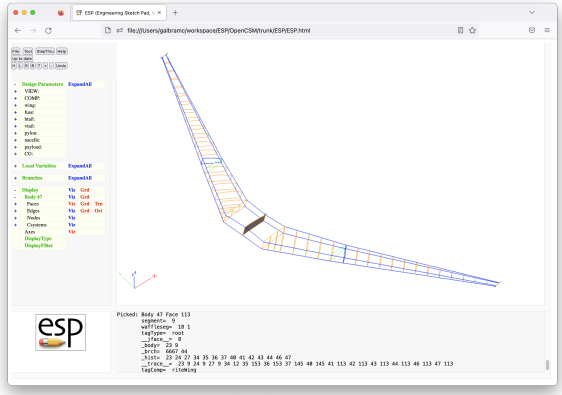
- Cantilever constraint on wing root rib `FACE/EDGE/NODE` using `capsConstraint`

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
```



session07/1_astros_ModalCantilever.py

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

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

ASTROS Cantilever Modal Analysis: Setup

- Specify the type of analysis
- Build material database

session07/1_astros_ModalCantilever.py

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

ASTROS Cantilever Modal Analysis: Setup

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

session07/1_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}
```

ASTROS Cantilever Modal Analysis: Setup

- Associate shell properties with `capsGroup`

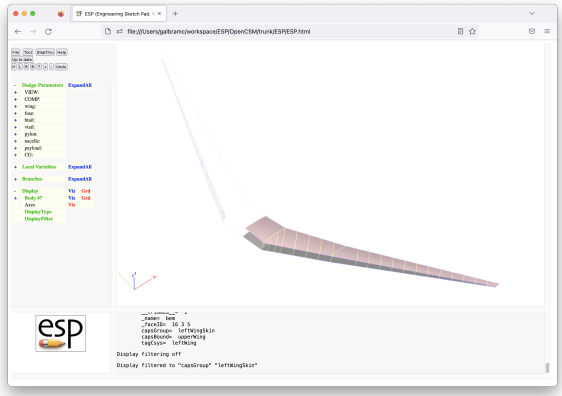
ESP/viewCantilever.udc

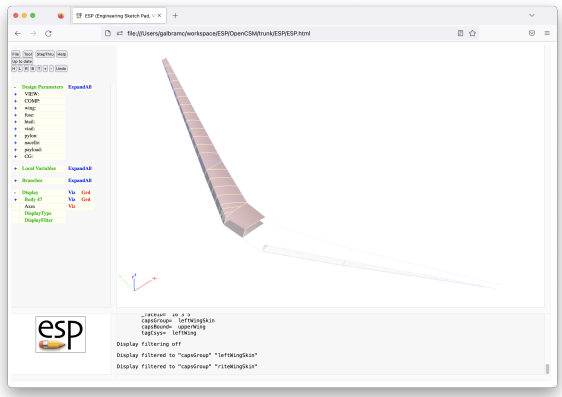
```
FACE    HAS  tagComp=leftWing       tagType=upper
SET          capsGroup=leftWingSkin capsBound=upperWing

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
```





session07/1_astros_ModalCantilever.py

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

ASTROS Cantilever Modal Analysis: Setup

- Associate shell properties with `capsGroup`

ESP/viewCantilever.udc

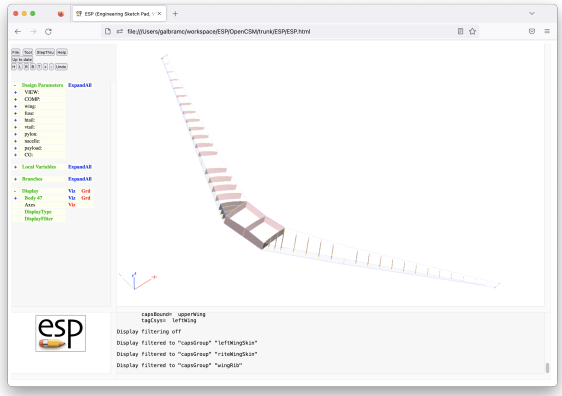
```
FACE    HAS  tagType=rib
SET          capsGroup=wingRib

FACE    HAS  tagComp=leftWing       tagType=tip
SET          capsGroup=wingRib      capsBound=leftTip

FACE    HAS  tagComp=riteWing       tagType=tip
SET          capsGroup=wingRib      capsBound=riteTip

FACE    HAS  tagComp=leftWing       tagType=fusespar
SET          capsGroup=wingRib

FACE    HAS  tagComp=riteWing       tagType=fusespar
SET          capsGroup=wingRib
```



session07/1_astros_ModalCantilever.py

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

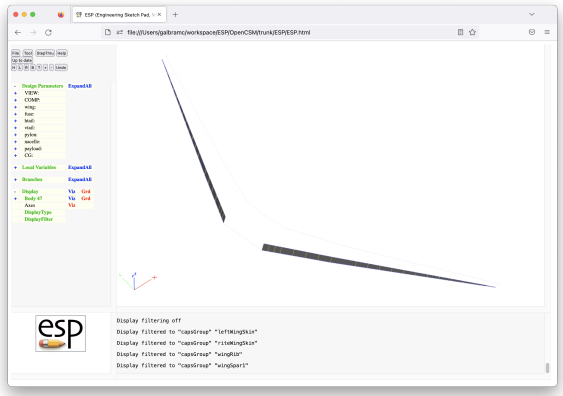
ASTROS Cantilever Modal Analysis: Setup

- Associate shell properties with `capsGroup`

ESP/viewCantilever.udc

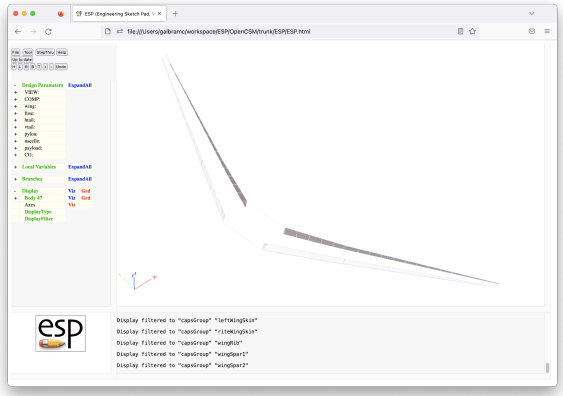
```
FACE    HAS   tagType=spar          tagIndex=1
SET           capsGroup=wingSpar1   capsMesh=wingSpar1

FACE    HAS   tagType=spar          tagIndex=2
SET           capsGroup=wingSpar2   capsMesh=wingSpar2
```



session07/1_astros_ModalCantilever.py

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



ASTROS Cantilever Modal Analysis: Execution

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

session07/1_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)
```

ASTROS Cantilever Modal Analysis: Output

- Print the Eigen frequencies

session07/1_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]))
```

ASTROS Support Modal Analysis: Setup

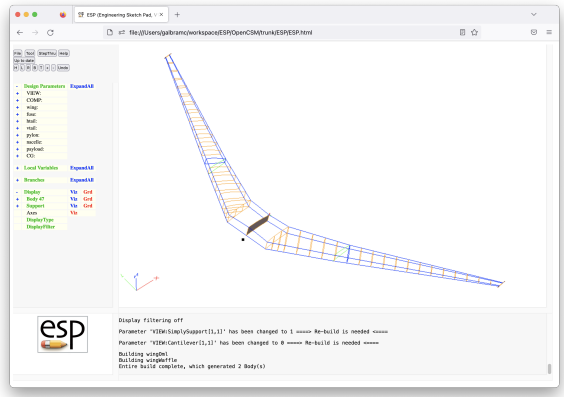
- Support connection on wing root rib `FACE/EDGE/NODE` using `capsConnectLink` and `capsConnect`

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
```

session07/2_astros_ModalSupport.py

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

ASTROS Support Modal Analysis: Setup

- 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
```

session07/2_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}
```

ASTROS Support Modal Analysis: Setup

- 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
```

session07/2_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     }
```

Overview

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

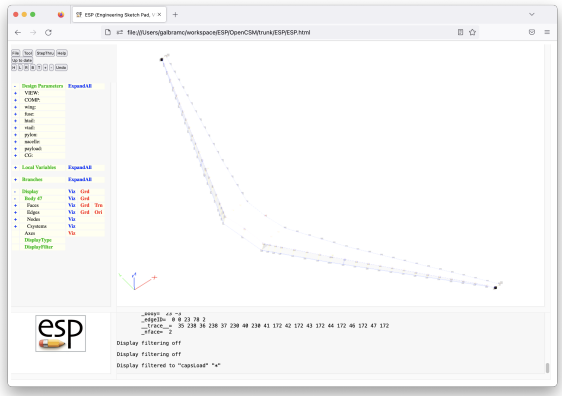
ASTROS Cantilever Static Analysis: Setup

- 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
```



session07/3_astros_StaticCantilever.py

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

ASTROS Cantilever Static Analysis: Setup

- 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
```

session07/3_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"
```

ASTROS Cantilever Static Analysis: Output

- Print the displacements

session07/3_astros_StaticCantilever.py

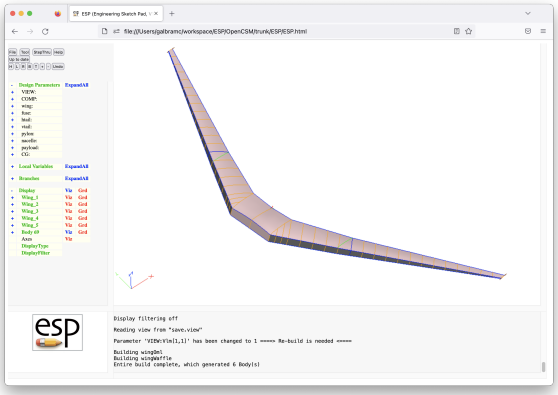
```
print ("\n--> Maximum displacements:")
print ("--> Tmax" , astros.output.Tmax )
print ("--> T1max", astros.output.T1max)
print ("--> T2max", astros.output.T2max)
print ("--> T3max", astros.output.T3max)
```

Overview

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

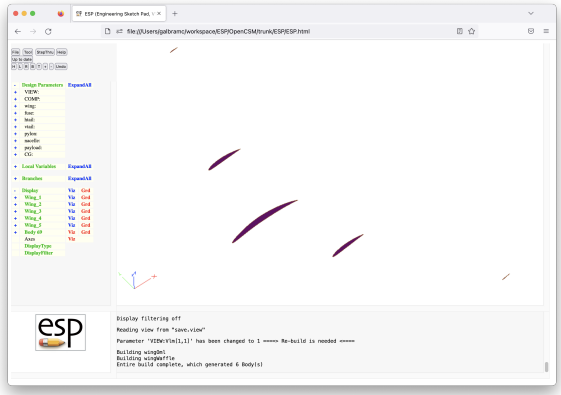
Flutter analysis with NASTRAN: Setup

- Cantilever aeroelastic flutter analysis



Structural body with full skin

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



VLM Aero bodies

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

session07/5_nastran_Flutter.py

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

Flutter analysis with NASTRAN: Setup

- Aeroelastic analysis coupled via `capsBound`

ESP/viewCantilever.udc

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

session07/5_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,
            "eigenNormalization"    : "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}
```

Overview

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

TACS Cantilever Static Analysis

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

session07/6_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 Cantilever Static Analysis

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

session07/6_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}")
```

TACS Size Optimization: Design_Variable

- Optimize membrane thickness

session07/7_tacs_sizing.py

```
# make thickness variables
thickness = skinShell["membraneThickness"]
skinThk = {"initialValue" : thickness,
           "lowerBound"   : thickness*0.5,
           "upperBound"   : thickness*1.5,
           "maxDelta"     : thickness}
thickness = ribShell["membraneThickness"]
ribThk  = {"initialValue" : thickness,
           "lowerBound"   : thickness*0.5,
           "upperBound"   : thickness*1.5,
           "maxDelta"     : thickness}
thickness = sparShell["membraneThickness"]
sparThk = {"initialValue" : thickness,
           "lowerBound"   : thickness*0.5,
           "upperBound"   : thickness*1.5,
           "maxDelta"     : thickness}
tacs.input.Design_Variable = {"skinThk": skinThk,
                              "ribThk" : ribThk,
                              "sparThk": sparThk}
```

TACS Size Optimization: Design_Variable_Relation

- Optimize membrane thickness

session07/7_tacs_sizing.py

```
tacs.input.Design_Variable = {"skinThk": skinThk,
                              "ribThk" : ribThk,
                              "sparThk": sparThk}

relLeftSkin = {"componentType": "Property",
               "componentName": "leftWingSkin",
               "fieldName"    : "T",
               "variableName" : "skinThk"}
relRiteSkin = {"componentType": "Property",
               "componentName": "riteWingSkin",
               "fieldName"    : "T",
               "variableName" : "skinThk"}
relRib      = {"componentType": "Property",
               "componentName": "wingRib",
               "fieldName"    : "T",
               "variableName" : "ribThk"}
relSpar     = {"componentType": "Property",
               "componentName": "wingSpar1",
               "fieldName"    : "T",
               "variableName" : "sparThk"}
tacs.input.Design_Variable_Relation = {"relLeftSkin": relLeftSkin,
                                       "relRiteSkin": relRiteSkin,
                                       "relRib"     : relRib,
                                       "relSpar"    : relSpar}
```

TACS Size Optimization: Execute

session07/7_tacs_sizing.py

```
        # solve each of the TACS struct problems, forward and adjoint analysis
        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)
```