

Engineering Sketch Pad (ESP)



Training Session 14 Structural Analyses and Data Transfers

John F. Dannenhoffer, III

jfdannen@syr.edu
Syracuse University

Marshall Galbraith

galbramc@mit.edu
Massachusetts Institute of Technology
updated for v1.25

- Updated `wing.csm`
 - Static structural deformations
 - Modal analysis
 - Fluid-structure interaction
-
- Final thoughts

- Previous `wing.csm` model was modified by adding:
 - `wing_Waffle.udc` — create waffle
 - `wing_Bem.udc` — create ribs, spars, and skins
 - `view_Bem.udc` — adds basic structures attributes
 - `view_Cantilever.udc` — adds CAPS-specific attributes

```
#####  
#                                                                 #  
# runModal --- run modal analysis through mASTROS              #  
#                                                                 #  
#           Written by John Dannenhoffer @ Syracuse University #  
#                   and Marshall Galbraith @ MIT                 #  
#                                                                 #  
#####  
  
# import pyCAPS module  
import pyCAPS  
  
# import os module  
import os  
import shutil
```

```
# load geometry [.csm] file
capsProblem = pyCAPS.Problem(problemName = "runModal",
                             capsFile    = "wing.csm",
                             outLevel    = 0)

# alias the geometry
wing = capsProblem.geometry

# reduce number of ribs for mASTROS
wing.cfcpmtr.wing.nrib1      = 1
wing.cfcpmtr.wing.nrib2      = 2
wing.cfcpmtr.wing.nrib3      = 4
```

```
# 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"
```

```
# create astros aim
astros = capsProblem.analysis.create(aim      = "astrosAIM",
                                     name      = "astros",
                                     autoExec  = False)

# link the EGADS tess Surface_Mesh to astros
astros.input["Mesh"].link(tess.output["Surface_Mesh"])

# set analysis specific variables
astros.input.Proj_Name      = "astros_modal"
astros.input.Mesh_File_Format = "Large"

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

```
# 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}
```



```
# 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}
```

```
# run AIM pre-analysis
print ("\n==> Running mASTROS pre-analysis...")
astros.preAnalysis()

##### Run mASTROS #####
# 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

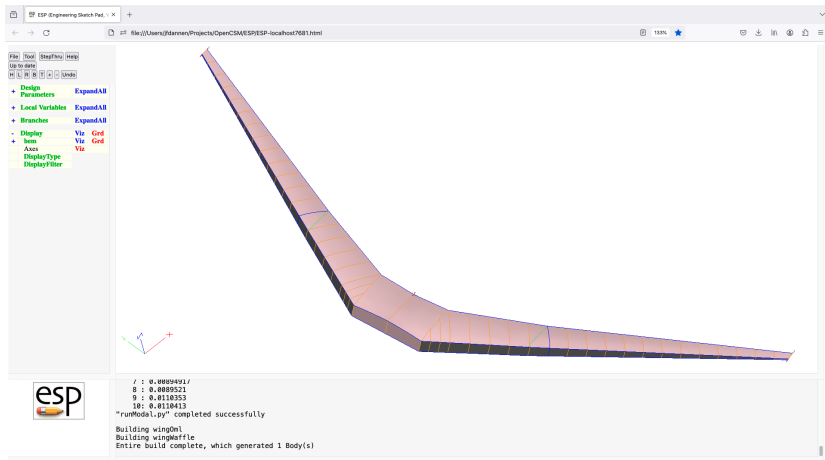
# run micro-ASTROS via system call
print ("\n==> Running mASTROS...")
astros.system("mastros.exe < " + astros.input.Proj_Name + ".dat > " + astros.input
```

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

# run AIM post-analysis
print ("\n==> Running mASTROS post-analysis...")
astros.postAnalysis()

# 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]))
```



```
#####  
#                                                                 #  
# runStatic --- run static structural deformations through mASTROS#  
#                                                                 #  
#           Written by John Dannenhoffer @ Syracuse University #  
#           and Marshall Galbraith @ MIT                        #  
#                                                                 #  
#####  
  
# import pyCAPS module  
import pyCAPS  
  
# import os module  
import os  
import shutil
```

```
# load geometry [.csm] file
capsProblem = pyCAPS.Problem(problemName = "runStatic",
                              capsFile    = "wing.csm",
                              outLevel    = 0)

# alias the geometry
wing = capsProblem.geometry

# reduce number of ribs for mASTROS
wing.cfpmtr.wing.nrib1      = 1
wing.cfpmtr.wing.nrib2      = 2
wing.cfpmtr.wing.nrib3      = 4
```

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

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

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

```
# create astros aim
astros = capsProblem.analysis.create(aim      = "astrosAIM",
                                     name     = "astros",
                                     autoExec = False)

# link the EGADS tess Surface_Mesh to astros
astros.input["Mesh"].link(tess.output["Surface_Mesh"])

# set analysis specific variables
astros.input.Proj_Name      = "astros_modal"
astros.input.Mesh_File_Format = "Large"

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



```
# 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"

# 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}
```

```
# 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}
```

```
# run AIM pre-analysis
print ("\n==> Running mASTROS pre-analysis...")
astros.preAnalysis()

##### Run mASTROS #####
# 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

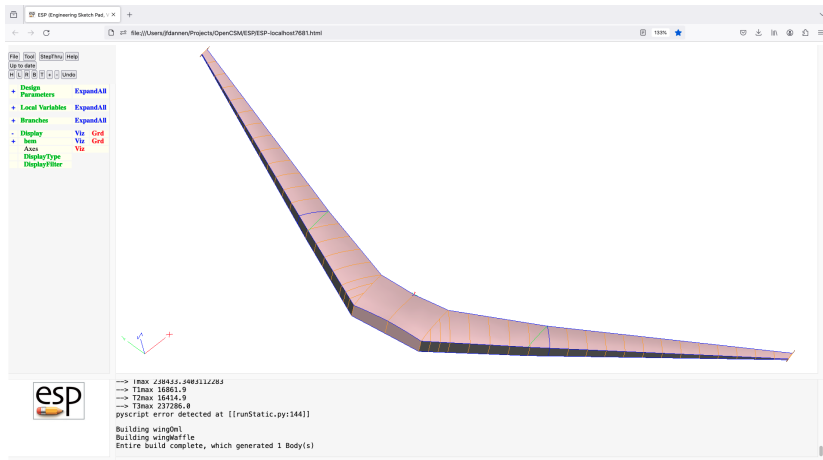
# remove temporary files
for file in files:
    if os.path.isfile(astros.analysisDir + os.sep + file):
        os.remove(astros.analysisDir + os.sep + file)
#####

# run AIM post-analysis
print ("\n==> Running mASTROS post-analysis...")
astros.postAnalysis()

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



runStatic.py Results



```
#####  
#                                                                 #  
# runFSI --- run fluid/struture interactions                      #  
#                                                                 #  
#           Written by John Dannenhoffer @ Syracuse University #  
#           and Marshall Galbraith @ MIT                        #  
#                                                                 #  
#####  
  
# import pyCAPS module  
import pyCAPS  
  
# import os module  
import os  
import shutil  
  
# import SU2 python environment  
from parallel_computation import parallel_computation as su2Run  
from mesh_deformation import mesh_deformation as su2Deform
```

```
# load CSM file
capsProblem = pyCAPS.Problem(problemName = "runFSI",
                              capsFile    = "wing.csm",
                              outLevel    = 0)

# create an alias for geometry
wing = capsProblem.geometry

# enable structures and CFD views
wing.cfgpmtr.VIEW.Cantilever = 1
wing.cfgpmtr.VIEW.CfdInviscid = 1

# reduce number of ribs for mASTROS
wing.cfgpmtr.wing.nrib1      = 1
wing.cfgpmtr.wing.nrib2      = 2
wing.cfgpmtr.wing.nrib3      = 4

wing.view()
```

```
# set the number of transfer iterations
numTransferIteration = 2

#-----

# create aflr4 AIM
aflr4 = capsProblem.analysis.create(aim = "aflr4AIM",
                                    name = "aflr4")

# farfield growth factor
aflr4.input.ff_cdfc = 1.4

# scaling factor to compute AFLR4 'ref_len' parameter
aflr4.input.Mesh_Length_Factor = 5

# edge mesh spacing discontinuity scaled interpolant and farfield BC
aflr4.input.Mesh_Sizing = {"leftWing": {"edgeWeight":1.0},
                           "riteWing": {"edgeWeight":1.0},
                           "Farfield": {"bcType":"Farfield"}}
```



```
# create AFLR3 AIM
aflr3 = capsProblem.analysis.create(aim = "aflr3AIM",
                                    name = "aflr3")

# link the surface mesh
aflr3.input["Surface_Mesh"].link(aflr4.output["Surface_Mesh"])
```

```
# create SU2 AIMs
su2 = capsProblem.analysis.create(aim    = "su2AIM",
                                name    = "su2")

# link the volume mesh
su2.input["Mesh"].link(aflr3.output["Volume_Mesh"])

# set inputs for su2
speedofSound = 340.0 # m/s
refVelocity   = 100.0 # m/s
refDensity    = 1.2 # kg/m**3

su2.input.Proj_Name           = "aeroelasticIterative"
su2.input.SU2_Version         = "Blackbird"
su2.input.Mach                = refVelocity/speedofSound
su2.input.Equation_Type       = "compressible"
su2.input.Num_Iter             = 5 # Way too few to converge the solver, but this
su2.input.Output_Format       = "Tecplot"
su2.input.Overwrite_CFG       = True
su2.input.Pressure_Scale_Factor = 0.5*refDensity*refVelocity**2
su2.input.Surface_Monitor     = "Wing"
```

```
inviscidBC = {"bcType" : "Inviscid"}  
su2.input.Boundary_Condition = {"Wing": inviscidBC,  
                                "Farfield": "farfield"}
```

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

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

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

```
# create ASTROS AIMS
astros = capsProblem.analysis.create(aim      = "astrosAIM",
                                     name      = "astros",
                                     autoExec  = False)

# link the surface mesh
astros.input["Mesh"].link(tess.output["Surface_Mesh"])

# set inputs for astros
astros.input.Proj_Name = su2.input.Proj_Name

# set analysis type
astros.input.Analysis_Type = "Static"

# external pressure load to astros that we will inherited from su2
load = {"loadType" : "PressureExternal"}
astros.input.Load = {"pressureAero": load}

# set constraints
constraint = {"groupName"      : ["rootConstraint"],
              "dofConstraint"  : 123456}
astros.input.Constraint = {"rootConstraint": constraint}
```

```
# set materials
unobtainium = {"youngModulus" : 2.2e13,
               "poissonRatio" : .5,
               "density"      : 7850}
madeupium   = {"materialType" : "isotropic",
               "youngModulus" : 1.2e14,
               "poissonRatio" : .5,
               "density"      : 7850}
astros.input.Material = {"Unobtainium": unobtainium,
                         "Madeupium"  : madeupium}
```

```
# 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}
```

```
# create the data transfer connections
boundNames = ["upperWing", "lowerWing", "leftTip", "riteTip"]
for boundName in boundNames:

    # create the bound
    bound = capsProblem.bound.create(boundName)

    # create the vertex sets on the bound for su2 and astros analysis
    su2Vset    = bound.vertexSet.create(su2)
    astrosVset = bound.vertexSet.create(astros)

    # create pressure data sets
    su2_Pressure    = su2Vset.dataSet.create("Pressure")
    astros_Pressure = astrosVset.dataSet.create("Pressure")

    # create displacement data sets
    su2_Displacement    = su2Vset.dataSet.create("Displacement", init=[0,0,0])
    astros_Displacement = astrosVset.dataSet.create("Displacement")
```



```
# link the data sets
astros_Pressure.link(su2_Pressure, "Conserve")
su2_Displacement.link(astros_Displacement, "Interpolate")

# close the bound as complete (cannot create more vertex or data sets)
bound.close()
```

```
# 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 SystemError
```

```
# aeroelastic iteration loop
for iter in range(numTransferIteration):

    ##### SU2 #####
    print ("\n==> Running SU2 pre-analysis...")
    su2.preAnalysis()

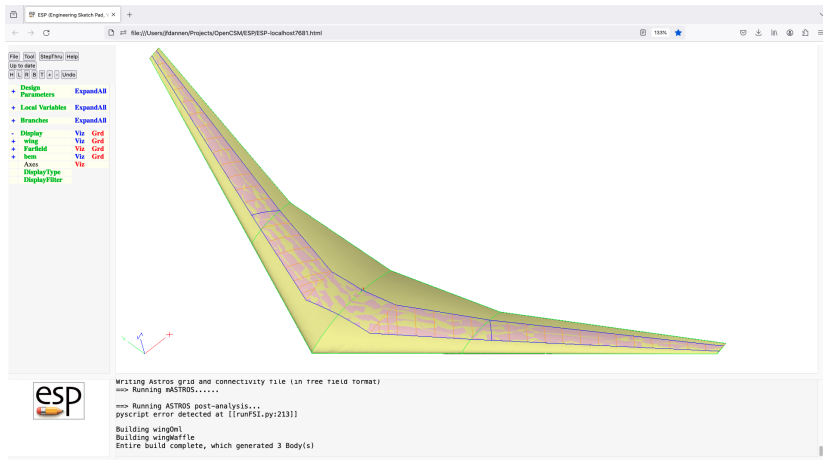
    #-----
    print ("\n==> Running SU2.....")
    currentDirectory = os.getcwd() # Get our current working directory
    os.chdir(su2.analysisDir)      # Move into test directory

    numberProc = 1
    # run SU2 mesh deformation
    if iter > 0:
        su2Deform(su2.input.Proj_Name + ".cfg", numberProc)

    # run SU2 solver
    su2Run(su2.input.Proj_Name + ".cfg", numberProc)

    shutil.copy("surface_flow_" + su2.input.Proj_Name + ".szplt", "surface_flow_" +
os.chdir(currentDirectory) # Move back to top directory
    #-----
```

```
##### ASTROS #####  
print ("\n==> Running ASTROS pre-analysis...")  
astros.preAnalysis()  
  
#-----  
# run mASTROS via system call  
print ("\n==> Running mASTROS.....")  
astros.system("mastros.exe < " + astros.input.Proj_Name + ".dat > " + astros.i  
#-----  
  
print ("\n==> Running ASTROS post-analysis...")  
astros.postAnalysis()  
#####
```



- ESP is freely available for download from `acd1.mit.edu/ESP`
- Based upon user requests, new and improved features are added continually
- Send bug reports to `jfdannen@syr.edu`
- Also send success stories to `jfdannen@syr.edu`

- Thank you for attending; send comments about the course to `jfdannen@syr.edu`