

Engineering Sketch Pad (ESP)



Training Session 11 Aerodynamic Analysis with CAPS

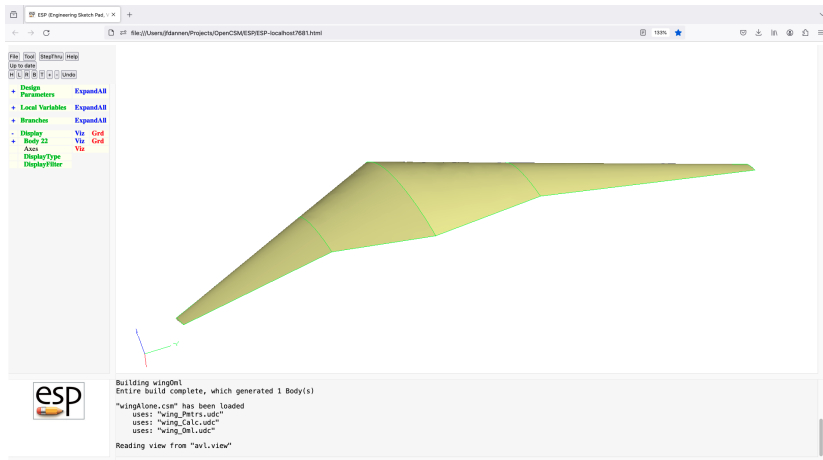
John F. Dannenhoffer, III

jfdannen@syr.edu
Syracuse University

Marshall Galbraith

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

- Basic wing model
- Vortex lattice method (AVL)
- Surface mesh generation (AFLR4)
- Volume mesh generation (AFLR3)
- Inviscid analysis (SU2)
- Homework exercises





Directory of Wing Files (1)

Top-level files

- `wingAlone.csm`
 - base geometry
- `wingAvl.csm`
 - geometry appropriate for AVL
- `wingCfdInv.csm`
 - geometry appropriate for CFD (inviscid)
- `wingCfdVis.csm`
 - geometry appropriate for CFD (viscous)



Directory of Wing Files (2)

Make geometry without CAPS attributes

- `wing_Pmtrs.udc`
 - DESPMTRs and CFGPMTRs that describe a wing
- `wing_Calc.udc`
 - calculations of global values
- `wing_Oml.udc`
 - make the OML of a wing
- `wing_Hinges.udc`
 - make the control surface hinges for a wing
- `wing_Vlm.udc`
 - make Bodys needed for AVL for a wing

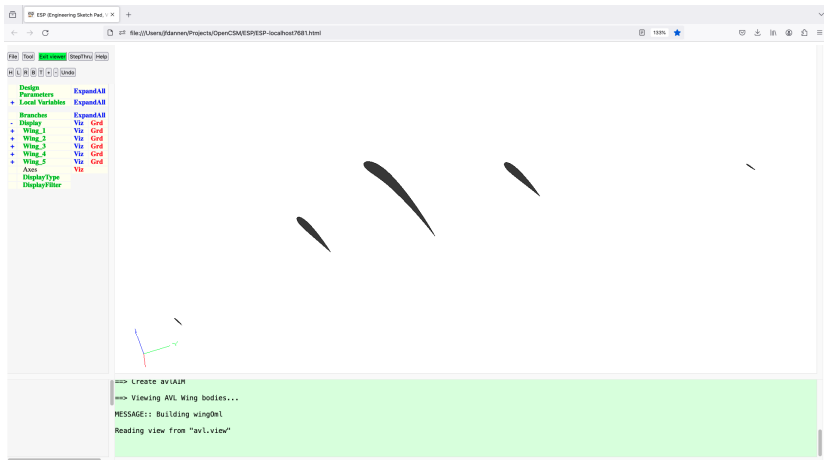


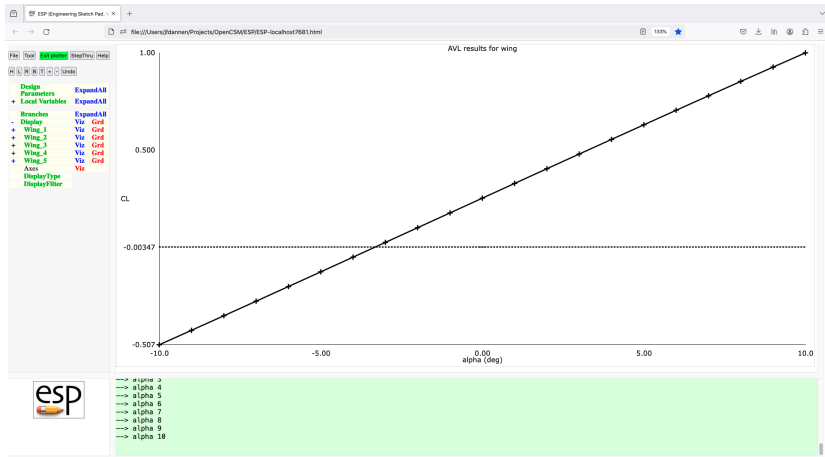
Directory of Wing Files (3)

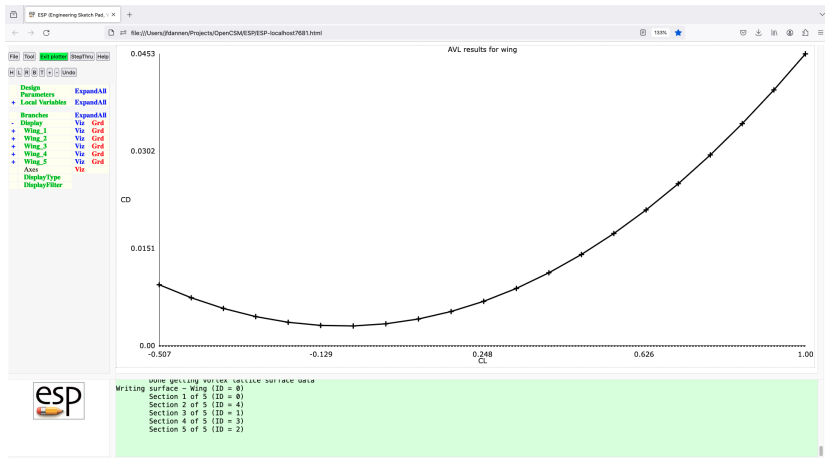
Add CAPS attributes to geometry

- `view_Oml.udc`
 - generate a “view” of the OML
- `view_Vlm.udc`
 - put CAPS attributes on AVL Bodys
- `view_CfdInviscid.udc`
 - put CAPS attributes on OML Bodys for inviscid calculations
- `view_CfdViscous.udc`
 - put CAPS attributes on OML Bodys for viscous calculations

- To run the examples in this session, do one of the following:
 - `serveESP wingAvl`
 - **Tool**→**Pyscript** `runAvl.py`
 - `serveESP runAvl.py`
 - `python runAvl.py`
 - Note: this starts separate **serveESP** to view geometry
 - Note: line plotting will not be available









runAvl.py (1)

```
#####  
#                                                                 #  
# runAvl --- run AVL on wing geometry; plot CL(alpha) and CL(CD) #  
#                                                                 #  
#           Written by John Dannenhoffer @ Syracuse University #  
#                   and Marshall Galbraith @ MIT                 #  
#                                                                 #  
#####  
  
# import pyCAPS module  
import pyCAPS  
from   pyOCSM import esp  
  
#-----  
  
# make a semi-colon-separated string from a list  
def makeString(array):  
    out = ""  
    for i in array:  
        out += str(i) + " ;"  
    return out
```

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

# make sure there is no tip treatment
capsProblem.geometry.despmtr["wing:tiptreat"].value = 0

# setup AIM for AVL
print ("\n==> Create avlAIM")
avl = capsProblem.analysis.create(aim = "avlAIM",
                                  name = "avl")

# view the geometry with ESP
print ("\n==> Viewing AVL Wing bodies...")
avl.geometry.view()
```

```
print ("\n==> Setting analysis values")
avl.input.Alpha = 1.0

# set meshing parameters for each surface
wing = {"numChord"      : 4,
        "numSpanTotal" : 24}

# Associate the surface parameters
# with capsGroups defined on the geometry
avl.input.AVL_Surface = {"Wing" : wing }

# initialize storage of results
Alpha = []
CL     = []
CD     = []
```

```
# run a sweep of angles of attack
for alpha in range(-10, 11):
    print("--> alpha", alpha)

    try:
        avl.input.Alpha = alpha

        CL.append(avl.output.CLtot)
        CD.append(avl.output.CDtot)
        Alpha.append(alpha)
    except pyCAPS.CAPSError:
        print("    *** did not work ***")
```

```
# load the plotter
esp.TimLoad("plotter", esp.GetEsp("pyscript"), "")

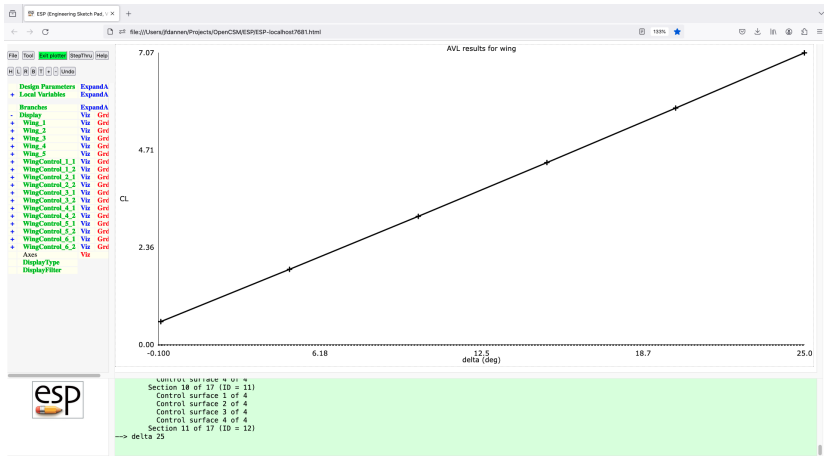
# plot the lift curve
esp.TimMesg("plotter", "new|AVL results for wing|alpha (deg)|CL|")
esp.TimMesg("plotter", "add|"+makeString(Alpha)+"|"+makeString(CL)+"|k-+|")
esp.TimMesg("plotter", "add|"+str(Alpha[0])+";"+str(Alpha[-1])+"|0;0|k:|")
esp.TimMesg("plotter", "add|-.1;+.1|0;0|k:|")
esp.TimMesg("plotter", "show")

# plot the drag polar
esp.TimMesg("plotter", "new|AVL results for wing|CL|CD|")
esp.TimMesg("plotter", "add|"+makeString(CL)+"|"+makeString(CD)+"|k-+|")
esp.TimMesg("plotter", "add|"+str(CL[0])+";"+str(CL[-1])+"|0;0|k:|")
esp.TimMesg("plotter", "show")

# exit the plotter
esp.TimQuit("plotter")

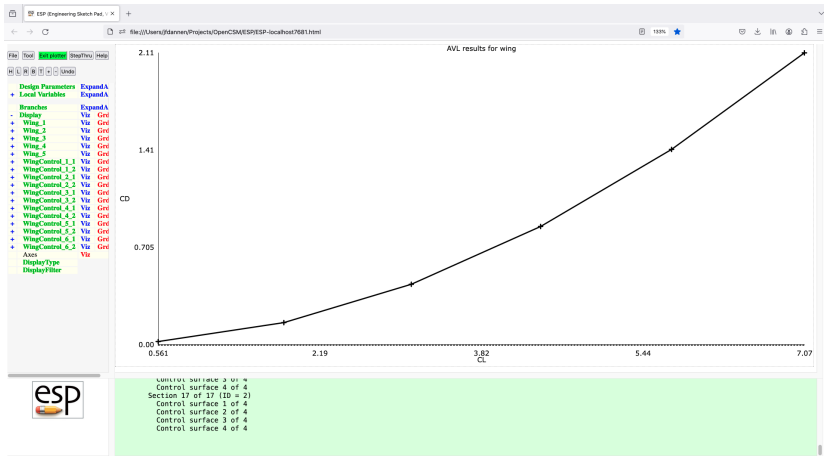
# close the capsProblem (required if you want to run another pyscript)
capsProblem.close()
```







runAvlFlaps (3)





runAvlFlaps.py (1)

```
#####  
#                                                                 #  
# runAvlFlaps --- run AVL on wing geometry; CL(flap) and CD(CL) #  
#                                                                 #  
#           Written by John Dannenhoffer @ Syracuse University #  
#                   and Marshall Galbraith @ MIT                 #  
#                                                                 #  
#####  
  
# Import pyCAPS module  
import pyCAPS  
from pyOCSM import esp  
  
#-----  
  
# make a semi-colon-separated string from a list  
def makeString(array):  
    out = ""  
    for i in array:  
        out += str(i) + ";  
    return out
```

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

# make sure there is no tip treatment
capsProblem.geometry.despmtr["wing:tiptreat"].value = 0

# make sure the controls are on
capsProblem.geometry.cfgpmtr["COMP:controls"].value = 1

# setup AIM for AVL
print ("\n==> Create avlAIM")
avl = capsProblem.analysis.create(aim  = "avlAIM",
                                  name = "avl")

# view the geometry with ESP
print ("\n==> Viewing AVL Wing bodies...")
avl.geometry.view()
```

```
print ("\n==> Setting analysis values")
avl.input.Alpha = 4.0

# set meshing parameters for each surface
wing = {"numChord"      : 4,
        "numSpanTotal" : 80}

# Associate the surface parameters
# with capsGroups defined on the geometry
avl.input.AVL_Surface = {"Wing" : wing }

# initialize storage of results
Delta = []
CL     = []
CD     = []
```

```
# run a sweep of flap deflections
for delta in range(0, 30, 5):
    print("--> delta", delta)

    try:
        # set up control surface deflections
        controls = {}

        hinge = capsProblem.geometry.despmtr["wing:hinge"].value
        for i in range(len(hinge)):
            controls["WingControl"+str(int(hinge[i][8]))] = {"deflectionAngle": del
        avl.input.AVL_Control = controls

        # explicitly trigger execution
        avl.runAnalysis()

        CL.append(avl.output.CLtot)
        CD.append(avl.output.CDtot)
        Delta.append(delta)
    except pyCAPS.CAPSError:
        print("    *** did not work ***")
```

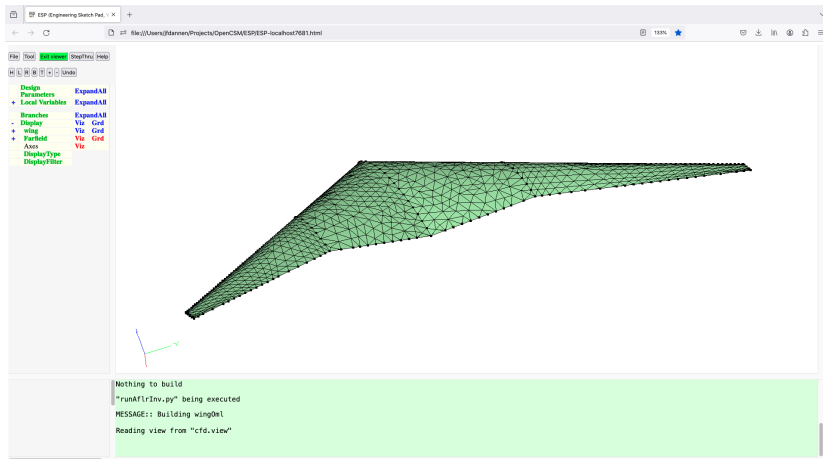
```
# load the plotter
esp.TimLoad("plotter", esp.GetEsp("pyscript"), "")

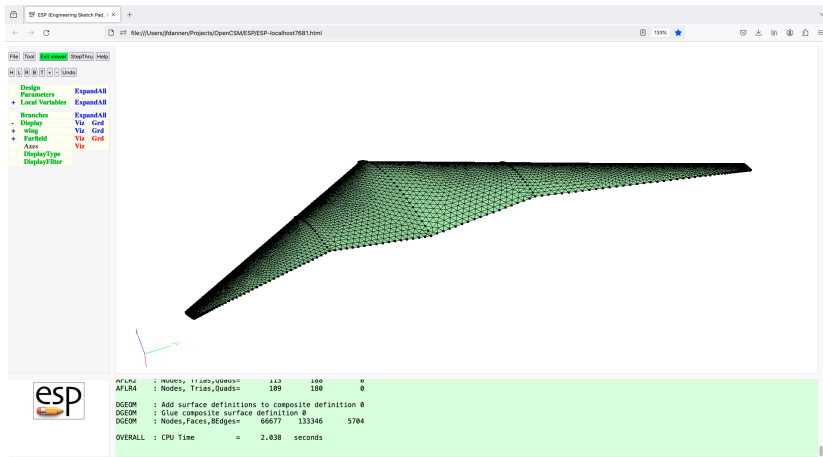
# plot the lift curve
esp.TimMesg("plotter", "new|AVL results for wing|delta (deg)|CL|")
esp.TimMesg("plotter", "add|"+makeString(Delta)+"|"+makeString(CL)+"|k-+|")
esp.TimMesg("plotter", "add|"+str(Delta[0])+"|"+str(Delta[-1])+"|0;0|k:|")
esp.TimMesg("plotter", "add|-.1;+.1|0;0|k:|")
esp.TimMesg("plotter", "show")

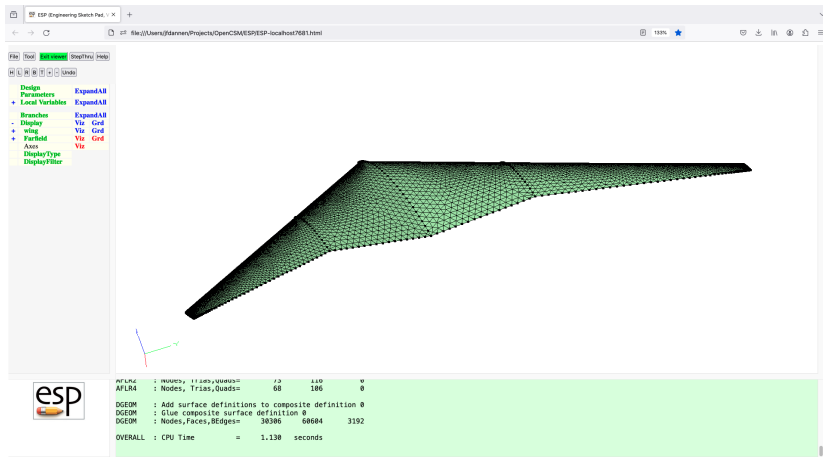
# plot the drag polar
esp.TimMesg("plotter", "new|AVL results for wing|CL|CD|")
esp.TimMesg("plotter", "add|"+makeString(CL)+"|"+makeString(CD)+"|k-+|")
esp.TimMesg("plotter", "add|"+str(CL[0])+"|"+str(CL[-1])+"|0;0|k:|")
esp.TimMesg("plotter", "show")

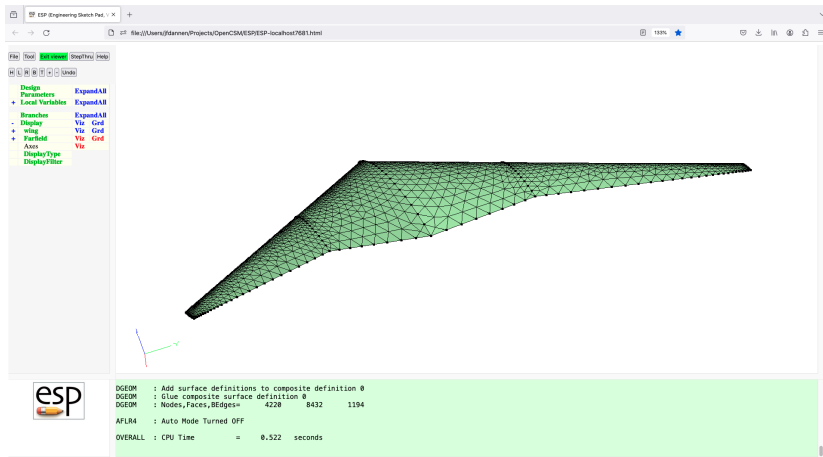
# exit the plotter
esp.TimQuit("plotter")

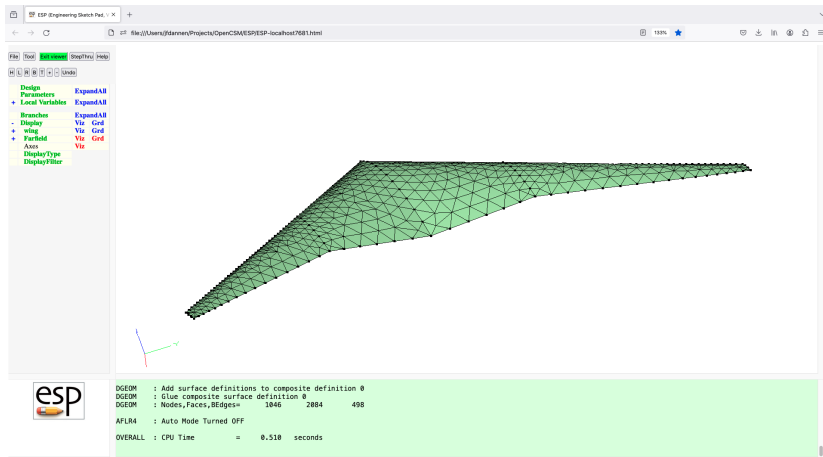
# close the capsProblem (required if you want to run another pyscript)
capsProblem.close()
```













runAflrInv.py (1)

```
#####  
#                                                                 #  
# runAflrInv --- run Aflr on wing geometry; plot surface grids #  
#                                                                 #  
#           Written by John Dannenhoffer @ Syracuse University #  
#           and Marshall Galbraith @ MIT                        #  
#                                                                 #  
#####  
  
# import pyCAPS module  
import pyCAPS  
from pyOCSM import esp  
  
#-----  
  
# load geometry [.csm] file  
capsProblem = pyCAPS.Problem(problemName = "runAflrInv",  
                             capsFile    = "wingCfdInv.csm",  
                             outLevel    = 0)
```

```
# setup AIM for AFLR
aflr4 = capsProblem.analysis.create(aim = "aflr4AIM",
                                   name = "aflr4")

# view the geometry with the EGADS tessellation
aflr4.geometry.view()

# mark capsMesh == Farfield with a Farfield bcType
aflr4.input.Mesh_Sizing = {"Farfield": {"bcType": "Farfield"}}

# run AIM analysis
aflr4.runAnalysis()

# view the AFLR4 surface tessellation
aflr4.geometry.view()
```

```
# farfield growth factor
aflr4.input.ff_cdfr = 1.4

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

# relative scale of maximum spacing bound relative to ref_len
# max_spacing = max_scale * ref_len
aflr4.input.max_scale = 0.1

# relative scale of minimum spacing bound relative to ref_len
# min_spacing = min_scale * ref_len
aflr4.input.min_scale = 0.01

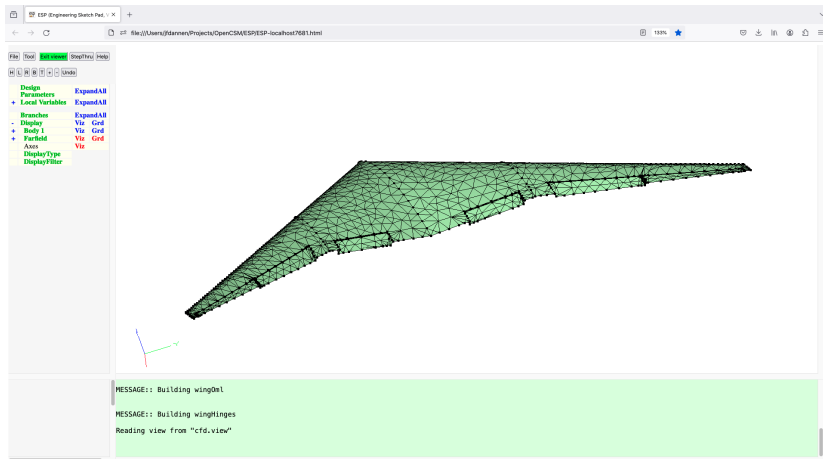
# absolute scale of minimum spacing bound for proximity
# abs_min_spacing = abs_min_scale * ref_len
aflr4.input.abs_min_scale = 0.001
```

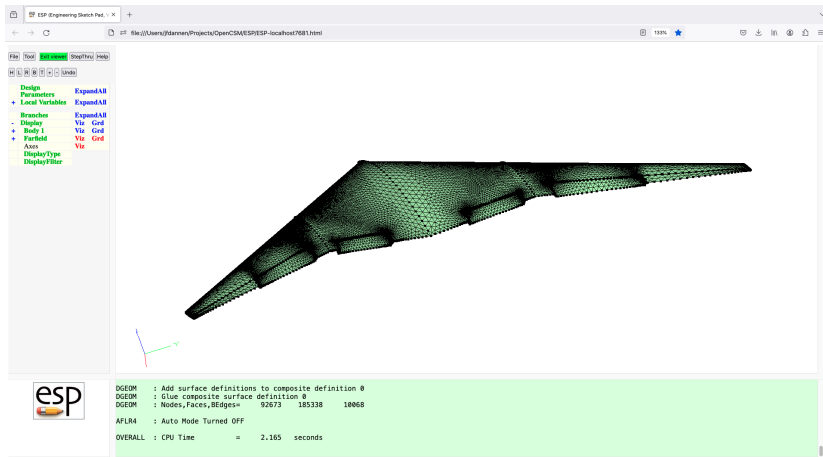
```
# use mesh length factor to make a series of meshes (not a family)
for Length_Factor in [1, 3, 9]:
    aflr4.input.Mesh_Length_Factor = Length_Factor

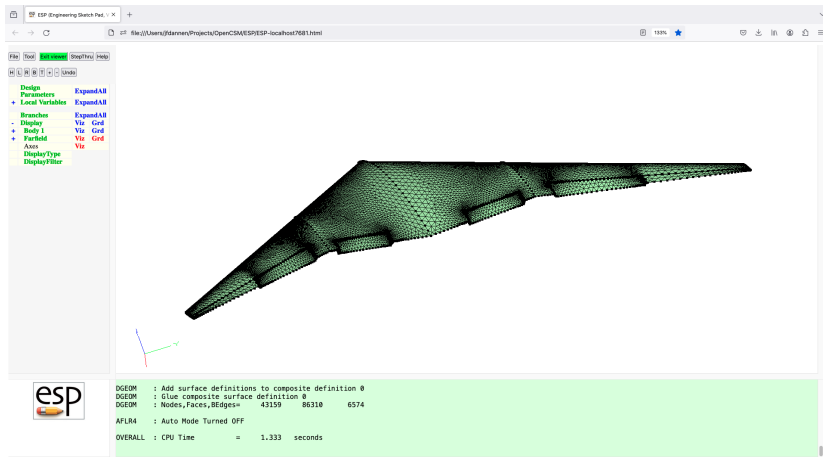
    # Run AIM
    aflr4.runAnalysis()

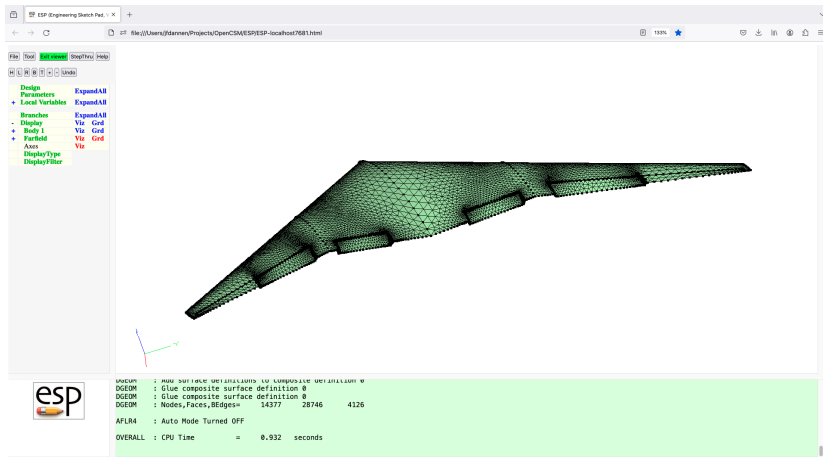
    # View the surface tessellation
    aflr4.geometry.view()

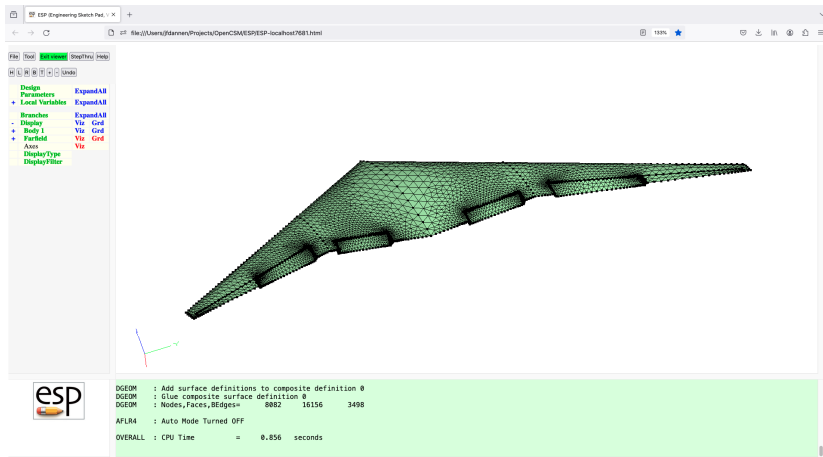
# close the capsProblem (required if you want to run another pyscript)
capsProblem.close()
```











runAflrFlapsInv.py (1)

```
#####  
#                                                                 #  
# runAflrFlapsInv --- run Aflr on wing geometry; plot surface grids #  
#                                                                 #  
#           Written by John Dannenhoffer @ Syracuse University #  
#           and Marshall Galbraith @ MIT                        #  
#                                                                 #  
#####  
  
# import pyCAPS module  
import pyCAPS  
from    pyOCSM import esp  
  
#-----  
  
# load geometry [.csm] file  
capsProblem = pyCAPS.Problem(problemName = "runAflrFlapsInv",  
                              capsFile    = "wingCfdInv.csm",  
                              outLevel    = 0)
```

```
# deflect the flaps
capsProblem.geometry.cfgpmtr["COMP:controls"].value = 1

hinges = capsProblem.geometry.despmtr["wing:hinge"].value
hinges[1][0] = 15          # left oflap
hinges[2][0] = 30          # left iflap
hinges[3][0] = 30          # rite iflap
hinges[4][0] = 15          # left iflap
capsProblem.geometry.despmtr["wing:hinge"].value = hinges
```

```
# setup AIM for AFLR
aflr4 = capsProblem.analysis.create(aim = "aflr4AIM",
                                   name = "aflr4")

# view the geometry with the EGADS tessellation
aflr4.geometry.view()

# mark capsMesh == Farfield with a Farfield bcType
aflr4.input.Mesh_Sizing = {"Farfield": {"bcType": "Farfield"}}

# run AIM analysis
aflr4.runAnalysis()

# view the AFLR4 surface tessellation
aflr4.geometry.view()
```



```
# farfield growth factor
aflr4.input.ff_cdfr = 1.4

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

# relative scale of maximum spacing bound relative to ref_len
# max_spacing = max_scale * ref_len
aflr4.input.max_scale = 0.1

# relative scale of minimum spacing bound relative to ref_len
# min_spacing = min_scale * ref_len
aflr4.input.min_scale = 0.01

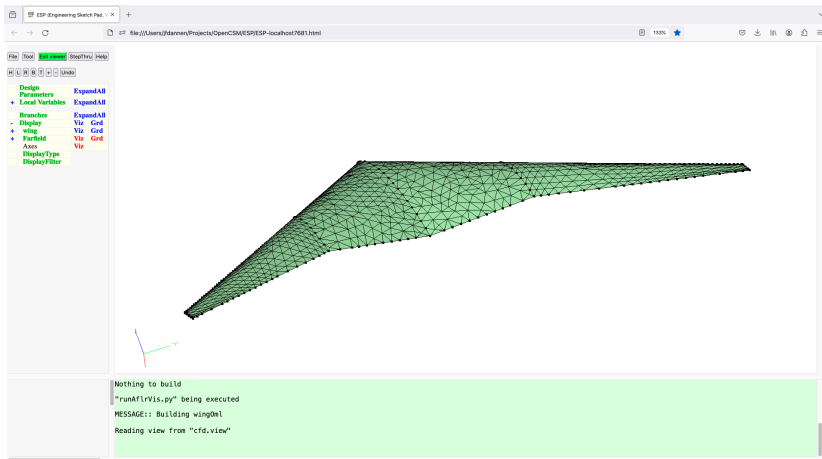
# absolute scale of minimum spacing bound for proximity
# abs_min_spacing = abs_min_scale * ref_len
aflr4.input.abs_min_scale = 0.001
```

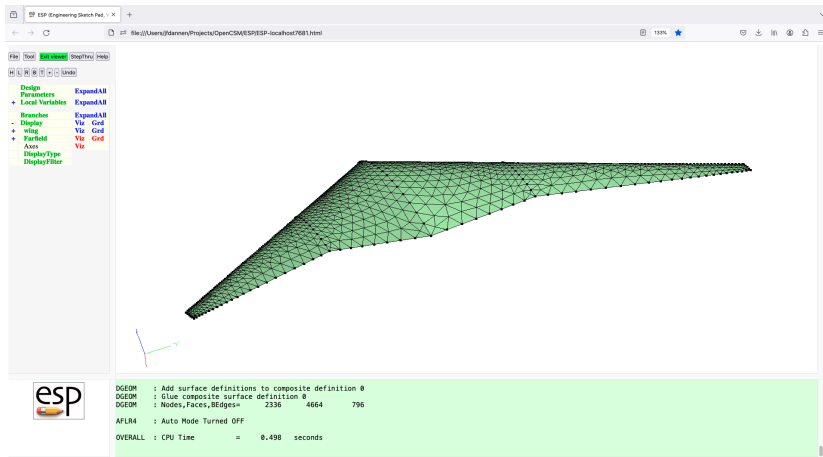
```
# use mesh length factor to make a series of meshes (not a family)
for Length_Factor in [1, 3, 9]:
    aflr4.input.Mesh_Length_Factor = Length_Factor

    # Run AIM
    aflr4.runAnalysis()

    # View the surface tessellation
    aflr4.geometry.view()

# close the capsProblem (required if you want to run another pyscript)
capsProblem.close()
```







runAflrVis.py (1)

```
#####  
#                                                                 #  
# runAflrVis --- run Aflr on wing geometry; plot surface grids  #  
#                                                                 #  
#           Written by John Dannenhoffer @ Syracuse University #  
#           and Marshall Galbraith @ MIT                        #  
#                                                                 #  
#####  
  
# import pyCAPS module  
import pyCAPS  
from   pyOCSM import esp  
  
#-----  
  
# load geometry [.csm] file  
capsProblem = pyCAPS.Problem(problemName = "runAflrVis",  
                              capsFile    = "wingCfdVis.csm",  
                              outLevel    = 0)
```

```
# setup AIM for AFLR
aflr4 = capsProblem.analysis.create(aim = "aflr4AIM",
                                   name = "aflr4")

# view the geometry with the EGADS tessellation
aflr4.geometry.view()

# mark capsMesh == Farfield with a Farfield bcType
aflr4.input.Mesh_Sizing = {"Farfield": {"bcType": "Farfield"}}

# farfield growth factor
aflr4.input.ff_cdfr = 1.4

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

# relative scale of maximum spacing bound relative to ref_len
# max_spacing = max_scale * ref_len
aflr4.input.max_scale = 0.1
```

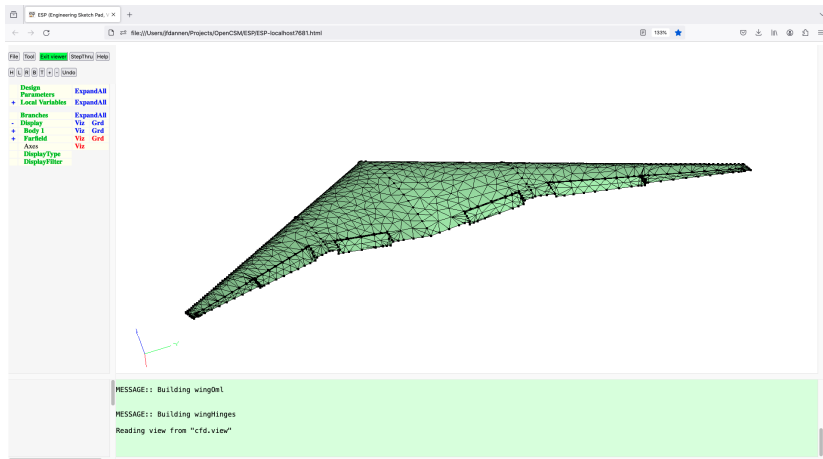
```
# relative scale of minimum spacing bound relative to ref_len
# min_spacing = min_scale * ref_len
aflr4.input.min_scale = 0.01

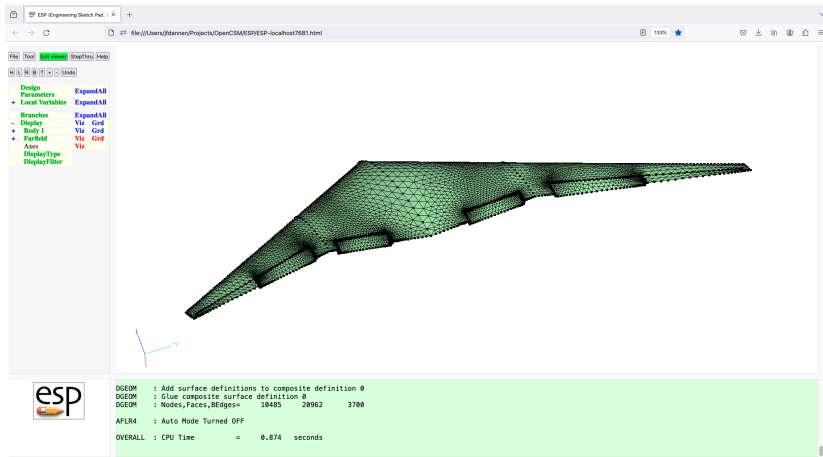
# absolute scale of minimum spacing bound for proximity
# abs_min_spacing = abs_min_scale * ref_len
aflr4.input.abs_min_scale = 0.001

# run AIM analysis
aflr4.runAnalysis()

# view the AFLR4 surface tessellation
aflr4.geometry.view()

# close the capsProblem (required if you want to run another pyscript)
capsProblem.close()
```







runAflrFlapsVis.py (1)

```
#####  
#                                                                 #  
# runAflrFlapsVis --- run Aflr on wing geometry; plot surface grids #  
#                                                                 #  
#           Written by John Dannenhoffer @ Syracuse University #  
#           and Marshall Galbraith @ MIT                        #  
#                                                                 #  
#####  
  
# import pyCAPS module  
import pyCAPS  
from    pyOCSM import esp  
  
#-----  
  
# load geometry [.csm] file  
capsProblem = pyCAPS.Problem(problemName = "runAflrFlapsVis",  
                             capsFile    = "wingCfdInv.csm",  
                             outLevel    = 0)
```

```
# deflect the flaps
capsProblem.geometry.cfgpmtr["COMP:controls"].value = 1

hinges = capsProblem.geometry.despmtr["wing:hinge"].value
hinges[1][0] = 15          # left oflap
hinges[2][0] = 30          # left iflap
hinges[3][0] = 30          # rite iflap
hinges[4][0] = 15          # left iflap
capsProblem.geometry.despmtr["wing:hinge"].value = hinges
```

```
# setup AIM for AFLR
aflr4 = capsProblem.analysis.create(aim = "aflr4AIM",
                                   name = "aflr4")

# view the geometry with the EGADS tessellation
aflr4.geometry.view()

# mark capsMesh == Farfield with a Farfield bcType
aflr4.input.Mesh_Sizing = {"Farfield": {"bcType": "Farfield"}}

# farfield growth factor
aflr4.input.ff_cdfc = 1.4

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

# relative scale of maximum spacing bound relative to ref_len
# max_spacing = max_scale * ref_len
aflr4.input.max_scale = 0.1
```

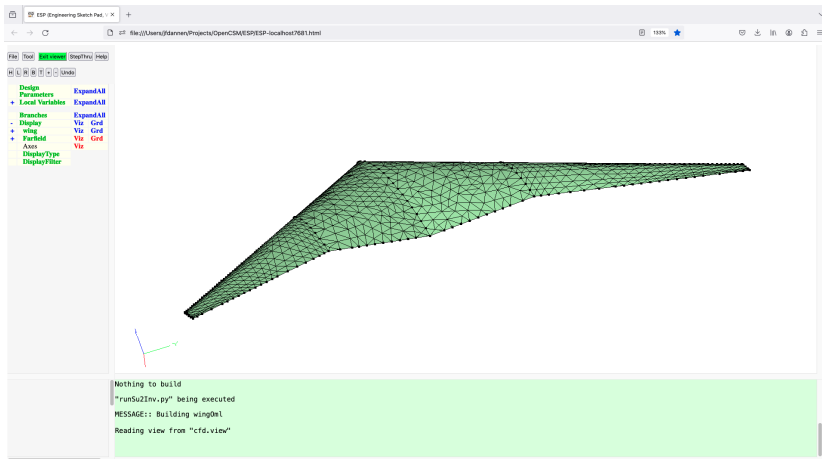
```
# relative scale of minimum spacing bound relative to ref_len
# min_spacing = min_scale * ref_len
aflr4.input.min_scale = 0.01

# absolute scale of minimum spacing bound for proximity
# abs_min_spacing = abs_min_scale * ref_len
aflr4.input.abs_min_scale = 0.001

# run AIM analysis
aflr4.runAnalysis()

# view the AFLR4 surface tessellation
aflr4.geometry.view()

# close the capsProblem (required if you want to run another pyscript)
capsProblem.close()
```





```
#####  
#                                                                 #  
# runSu2Inv --- run Su2 on wing geometry                        #  
#                                                                 #  
#           Written by John Dannenhoffer @ Syracuse University #  
#           and Marshall Galbraith @ MIT                        #  
#                                                                 #  
#####  
  
# import pyCAPS module  
import pyCAPS  
from pyOCSM import esp  
  
import os  
  
# import SU2 python environment  
from parallel_computation import parallel_computation as su2Run  
  
#-----  
  
# load geometry [.csm] file  
capsProblem = pyCAPS.Problem(problemName = "runSu2Inv",  
                             capsFile    = "wingCfdInv.csm",  
                             outLevel    = 0)
```



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

# farfield growth factor
aflr4.input.ff_cdfr = 1.4

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

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

# create AFLR3 AIM to generate the volume mesh
aflr3 = capsProblem.analysis.create(aim = "aflr3AIM",
                                   name = "aflr3")
```

```
# link the aflr4 Surface_Mesh as input to aflr3
aflr3.input["Surface_Mesh"].link(aflr4.output["Surface_Mesh"])

# show the surface meshes
aflr4.geometry.view()

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

# link the aflr3 Volume_Mesh as input to su2
su2.input["Mesh"].link(aflr3.output["Volume_Mesh"])

# set project name. Files written to analysisDir will have this name
su2.input.Proj_Name = "inviscidWing"
```

```
su2.input.Alpha          = 1.0          # AoA
su2.input.Mach           = 0.5          # Mach number
su2.input.Equation_Type  = "Compressible" # Equation type
su2.input.Num_Iter       = 10           # Number of iterations

# set number of inner iterations and convergence
su2.input.Input_String = ["INNER_ITER= 10",
                          "CONV_FIELD= RMS_DENSITY",
                          "CONV_RESIDUAL_MINVAL= -8"]

# set boundary conditions via capsGroup
inviscidBC = {"bcType" : "Inviscid"}
su2.input.Boundary_Condition = {"Wing"      : inviscidBC,
                                "Farfield": "farfield"}

# specify the boundaries used to compute forces
su2.input.Surface_Monitor = ["Wing"]

# set SU2 Version
su2.input.SU2_Version = "Blackbird"
```

```
# run AIM pre-analysis
su2.preAnalysis()

##### Run SU2 #####
print ("\n\nRunning SU2.....")
currentDirectory = os.getcwd() # Get our current working directory

os.chdir(su2.analysisDir) # Move into test directory

# run SU2 with specified number of partitions
su2Run(su2.input.Proj_Name + ".cfg", partitions = 1)

os.chdir(currentDirectory) # Move back to top directory
#####

# run AIM post-analysis
su2.postAnalysis()
```

```
# get Lift and Drag coefficients
print ("\n==> Total Forces and Moments")

print ("--> Cl = ", su2.output.CLtot,
        "Cd = ", su2.output.CDtot)

# Get Cmx, Cmy, and Cmx coefficients
print ("--> Cmx = ", su2.output.CMXtot,
        "Cmy = ", su2.output.CMYtot,
        "Cmx = ", su2.output.CMXtot)

# Get Cx, Cy, Cz coefficients
print ("--> Cx = ", su2.output.CXtot,
        "Cy = ", su2.output.CYtot,
        "Cz = ", su2.output.CZtot)
```

- Start with the files in
`$ESP_ROOT/training/ESP/data/session11`
- Run the various cases for `wing:aspect = 7.0`
- Make changes needed to get plot of $C_L(\alpha)$ and $C_D(C_L)$ from SU2