

# Computational Aircraft Prototype Syntheses



## Training Session 6

### Aero Modeling: AVL, friction, and masstran

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)

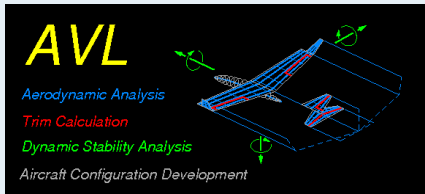
Syracuse University

- AVL Overview
- AVL Geometry Definition
  - Plane Vanilla
  - Control Surfaces
  - Reference Quantities
  - Transport
- Stability Derivatives
- AVL Eigenmode Analysis
  - Pure AVL
  - AVL and masstran
- Linked Analysis Parameters
  - AVL and FRICTION
- Suggested Exercises

- Aerodynamic and flight-dynamic analysis of rigid aircraft

## Extended Vortex-Lattice Model

- Aerodynamic Components
  - Lifting surfaces
  - Slender bodies
- Control deflections
  - Via normal-vector tilting
  - Leading edge or trailing edge flaps
- General freestream description
  - $\alpha$ ,  $\beta$  flow angles
  - $p$ ,  $q$ ,  $r$  aircraft rotation
- Aerodynamic outputs
  - Forces and moments, in body or stability axes
  - Force and moment derivatives w.r.t. angles, rotations, controls



## Trim Calculation

- Operating variables
  - $\alpha$ ,  $\beta$
  - $p$ ,  $q$ ,  $r$
  - control deflections
- Constraints
  - direct constraints on variables
  - indirect constraints via specified CL, moments
  - level or banked horizontal flight
  - steady pitch rate (looping) flight

## Eigenmode analysis

- Predicts flight stability characteristics
- Rigid-body, quasi-steady aero model
- Eigenvalue root progression with a parameter
- Display of eigenmode motion in real time

## AVL GUI on macOS

- Requires XQuartz  $\geq 2.8.1$

## AVL AIM Documentation

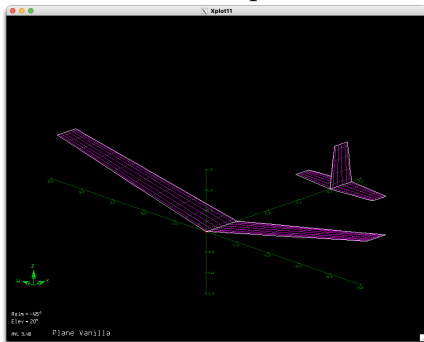
## VLM Geometry airfoil sections

```
#-----
SURFACE
WING
#Nchordwise  Cspace  Nspanwise  Sspace
1            1.0     16          -2.0
YDUPLICATE
0.0

SECTION
#Xle  Yle  Zle  Chord  Ainc  Nspanwise  Sspace
-0.25 0.   0.   1.000  0.   8          1.0
AIRFOIL
naca2412.dat

SECTION
#Xle  Yle  Zle  Chord  Ainc  Nspanwise  Sspace
-0.175 7.5 0.5  0.700  0.   0          0
AIRFOIL
naca0012.dat
```

## VLM mesh is flat panels



## session06/avlPlaneVanilla.csm

---

```
##### Vertical Tail #####
```

```
# Base
```

```
UDPRIM      naca Thickness vtail:thick
```

```
SCALE       vtail:croot
```

```
TRANSLATE   vtail:xroot    0    vtail:zroot
```

```
SELECT body
```

```
    ATTRIBUTE capsGroup      $Vtail
```

```
    ATTRIBUTE vlmControl_Rudder 0.75
```

```
    ATTRIBUTE capsAIM          $avlAIM
```

```
    ATTRIBUTE _name            $vtail:base
```

```
# Tip
```

```
UDPRIM      naca Thickness vtail:thick
```

```
SCALE       vtail:ctip
```

```
TRANSLATE   vtail:xtip     0    vtail:span
```

```
SELECT body
```

```
    ATTRIBUTE capsGroup      $Vtail
```

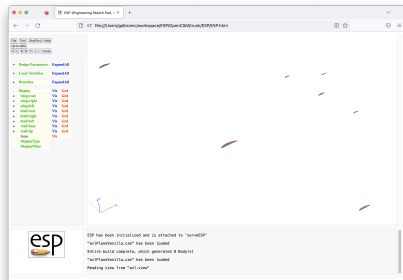
```
    ATTRIBUTE vlmControl_Rudder 0.75
```

```
    ATTRIBUTE capsAIM          $avlAIM
```

```
    ATTRIBUTE _name            $vtail:tip
```

---

## ESP geometry airfoil sections



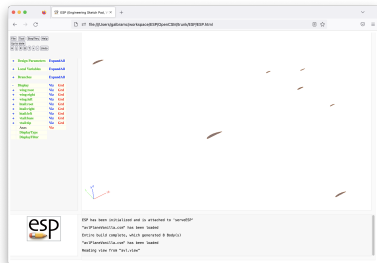
- capsGroup groups airfoils into surfaces

## session06/avlPlaneVanilla.csm

```
UDPRIM      naca Thickness wing:thick Camber wing:camber
SCALE       wing:croot
ROTATEX     90      0      0
TRANSLATE   wing:xroot 0      wing:zroot
SELECT body
  ATTRIBUTE capsGroup $Wing
```

```
UDPRIM      naca Thickness wing:thick Camber wing:camber
SCALE       htail:croot
ROTATEX     90      0      0
TRANSLATE   htail:xroot 0      htail:zroot
SELECT body
  ATTRIBUTE capsGroup $Htail
```

```
UDPRIM      naca Thickness vtail:thick
SCALE       vtail:croot
TRANSLATE   vtail:xroot 0      vtail:zroot
SELECT body
  ATTRIBUTE capsGroup $Vtail
```





- VLM meshing parameters defined via capsGroups

session06/1\_avl\_PlaneVanilla.py

---

```
# Set meshing parameters for each surface
```

```
wing = {"numChord"      : 4,  
        "numSpanTotal" : 24}
```

```
htail = {"numChord"      : 4,  
         "numSpanTotal" : 16}
```

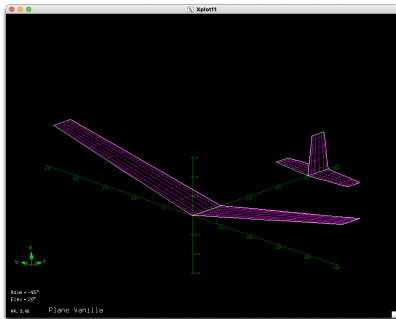
```
vtail = {"numChord"      : 4,  
         "numSpanTotal" : 10}
```

```
# Associate the surface parameters
```

```
# with capsGroups defined on the geometry
```

```
avl.input.AVL_Surface = {"Wing" : wing ,  
                         "Htail": htail,  
                         "Vtail": vtail}
```

---



- Controls specified at airfoil sections

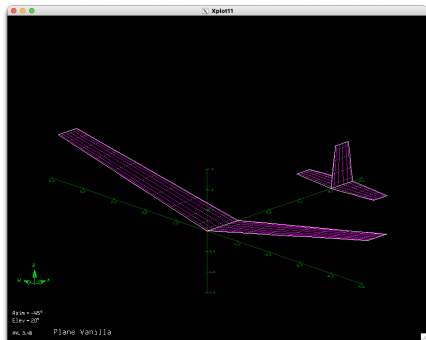
```

#-----
SURFACE
STAB
#Nchordwise Cspace Nspanwise Sspace
1 1.0 7 -2.0
YDUPLICATE
0.0

SECTION
#Xle Yle Zle Chord Ainc Nspanwise Sspace
6.0 0. 0.0 0.4 0. 7 -1.25
CONTROL
#name gain Xhinge XYZhvec SgnDup
elevator 1.0 0.0 0. 1. 0. 1

SECTION
#Xle Yle Zle Chord Ainc Nspanwise Sspace
-0.075 2.00 0.0 0.3 0. 0 0
CONTROL
#name gain Xhinge XYZhvec SgnDup
elevator 1.0 0.0 0. 1. 0. 1

```



- vlmControl\_“Name” specifies a section with a control surface
  - “Name” is the name of the control surface
  - Value is chord fraction of hinge line

[session06/avlPlaneVanilla.csm](#)

Wing Right Tip Airfoil

```

TRANSLATE wing:xtip      -wing:span/2      wing:ztip
SELECT body
    ATTRIBUTE capsGroup      $Wing
    ATTRIBUTE vlmControl_AileronRight 0.8 #Hinge line 80% chord

```

## Wing Root Airfoil

```

TRANSLATE wing:zroot      0      wing:zroot
SELECT  body
ATTRIBUTE capsGroup      $Wing
ATTRIBUTE vlmControl_AileronRight 0.8 #Hinge line 80% chord
ATTRIBUTE vlmControl_AileronLeft  0.8 #Hinge line 80% chord

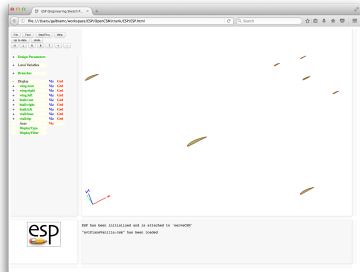
```

### Wing Left Tip Airfoil

```

TRANSLATE wing:xtip      wing:span/2      wing:ztip
SELECT  body
        ATTRIBUTE capsGroup      $Wing
        ATTRIBUTE vlmControl_AileronLeft 0.8 #Hinge line 80% chord

```



- vlmControl\_“Name” specifies a section with a control surface
  - “Name” is the name of the control surface
  - Value is chord fraction of hinge line

session06/avlPlaneVanilla.csm

```
UDPRIM      naca Thickness wing:thick Camber wing:camber
SCALE       wing:croot
ROTATEX     90      0      0
TRANSLATE   wing:xroot 0      wing:zroot
SELECT body
  ATTRIBUTE capsGroup      $Wing
  ATTRIBUTE vlmControl_AileronRight 0.8 #Hinge line 80% chord
  ATTRIBUTE vlmControl_AileronLeft  0.8 #Hinge line 80% chord
```

session06/2\_avl\_PlaneVanillaControl.py

```
# Set control surface parameters
aileronLeft = {"deflectionAngle" : -25.0}
aileronRight = {"deflectionAngle" : 25.0}
elevator     = {"deflectionAngle" : 5.0}
rudder       = {"deflectionAngle" : -2.0}

avl.input.AVL_Control = {"AileronLeft" : aileronLeft ,
                        "AileronRight" : aileronRight,
                        "Elevator"     : elevator   ,
                        "Rudder"       : rudder     }
```

## AVL Input Header

---

!Sref	Cref	Bref
12.0	1.0	15.0
!Xref	Yref	Zref
0.0	0.0	0.0

---

## session06/avlPlaneVanilla.csm

---

ATTRIBUTE	capsReferenceArea	wing:area
ATTRIBUTE	capsReferenceSpan	wing:span
ATTRIBUTE	capsReferenceChord	wing:croot
ATTRIBUTE	capsReferenceX	wing:xroot+wing:croot/4

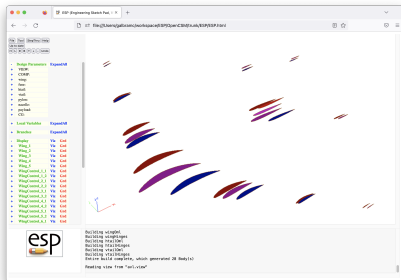
---

## capsReference\* attributes

Sref	capsReferenceArea	area for coefficients ( $C_L$ , $C_D$ , $C_m$ , etc)
Cref	capsReferenceChord	chord for pitching moment ( $C_m$ )
Bref	capsReferenceSpan	span for roll,yaw moments ( $C_l$ , $C_n$ )
Xref	capsReferenceX	location for moments, rotation rates

capsReference\* attributes on one or more bodies (consistent)

ESP geometry airfoil sections  
red/blue control surfaces



View Concept and VLM in ESP

---

```
cd data/ESP
serveESP transport.csm
```

```
VIEW:Concept    0
VIEW:Vlm        1
```

---

session06/3\_avl\_TransportGeom.py

---

```
# Load geometry [.csm] file
filename = os.path.join(".", "ESP", "transport.csm")
print ("\n==> Loading geometry from file \""+filename+"\"")
capsProblem = pyCAPS.Problem(problemName = "design_Transport",
                             capsFile = filename,
                             outLevel = 0)

# Alias the geometry
transport = capsProblem.geometry

# Change to VLM view
transport.cfgpmtr.VIEW.Concept = 0
transport.cfgpmtr.VIEW.Vlm     = 1

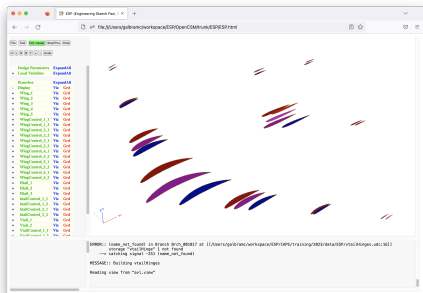
# view the geometry with ESP
print ("\n==> Viewing transport bodies...")
transport.view()

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

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

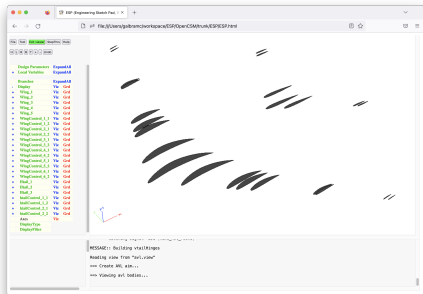
---

transport.view()





avl.geometry.view()



- Vertical tail is missing capsAIM attribute!
- ESP/viewVlm.udc must be fixed for session06 examples to work (uncomment line 68)

ESP/viewVlm.udc

---

```
SELECT      BODY  bodyList[ibody+nWingBody+nHtailBody+prevStack]  
#ATTRIBUTE capsAIM      $avlAIM;frictionAIM;astrosAIM;nastranAIM;egadsTessAIM
```

---

- Very rich input data set
  - Many geometric parameter
  - Multiple bodies
  - Many attributes on BODY/FACE/EDGE/NODE
- Not all error checking can be automated
- Significant user responsibility to check consistency
- Always check initial setup as much as possible

## ESP/wingPmtrs.udc

---

```

DIMENSION wing:hinge      6 9 1 # ymin          ymax
#                          theta  x/c   y/span  z/t      x/c   y/span  z/t   gap  grp
DESPMTR   wing:hinge      "-10.0; 0.75; -0.98; 0.50; 0.75; -0.70; 0.50; 0.25; 1; \ left aileron
+10.0; 0.75; -0.69; 0.00; 0.75; -0.43; 0.00; 0.25; 2; \ left oflap
+15.0; 0.85; -0.33; 0.00; 0.90; -0.14; 0.00; 0.25; 3; \ left iflap
+15.0; 0.90; 0.14; 0.00; 0.85; 0.33; 0.00; 0.25; 3; \ rite iflap
+10.0; 0.75; 0.43; 0.00; 0.75; 0.69; 0.00; 0.25; 2; \ rite oflap
+10.0; 0.75; 0.70; 0.50; 0.75; 0.98; 0.50; 0.25; 4" # rite aileron

```

---

## ESP/wingVlm.udc

---

```

# create the section at the beginning
BOX      xmin ybeg zmin xmax-xmin 0 xmax-zmin
RESTORE  wingOml
INTERSECT
    ATTRIBUTE _name                                !$WingControl_+ihinge+$_1
    ATTRIBUTE !$vlmControl_WingControl+tagIndex xoverc1

```

---

## session06/4\_avl\_TransportControl.py

---

```

controls = {}

hinge = transport.despmtr.wing.hinge
for i in range(len(hinge)):
    controls["WingControl_"+str(int(hinge[i][8]))] = {"deflectionAngle": hinge[i][0]}

```

---

- AVL Overview
- AVL Geometry Definition
  - Plane Vanilla
  - Control Surfaces
  - Reference Quantities
  - Transport
- **Stability Derivatives**
- AVL Eigenmode Analysis
  - Pure AVL
  - AVL and masstran
- Linked Analysis Parameters
  - AVL and FRICTION
- Suggested Exercises

- Stability derivatives available as analysis outputs

session06/2\_avl\_PlaneVanillaControl.py

---

```
# Print CL derivatives with old interface
```

```
print ("\n==> Some stability derivatives (old)")
```

```
print ("--> CLa  =", avl.output["CLa"].value) # - Alpha
```

```
print ("--> CLb  =", avl.output["CLb"].value) # - Beta
```

```
ControlStability = avl.output.ControlStability # Dictionary of dictionaries
```

```
print("--> dCL/dAileronRight =", ControlStability['AileronRight']['CLtot']) # - Quantities w.r.t. AileronRight
```

```
print("--> dCL/dAileronLeft  =", ControlStability['AileronLeft']['CLtot']) # - Quantities w.r.t. AileronLeft
```

```
print("--> dCL/dElevator     =", ControlStability['Elevator']['CLtot']) # - Quantities w.r.t. Elevator
```

```
print("--> dCL/dRudder       =", ControlStability['Rudder']['CLtot']) # - Quantities w.r.t. Rudder
```

---

- Stability derivatives available as analysis outputs

## session06/2\_avl\_PlaneVanillaControl.py

```
# Print CL derivatives with new interface
print ("\n==> Some stability derivatives (new)")
print ("--> dCL/dAlpha  =", avl.output["CLtot"].deriv("Alpha"))
print ("--> dCL/dBeta   =", avl.output["CLtot"].deriv("Beta"))

print ("--> dCL/dAileronRight =", avl.output["CLtot"].deriv("AileronRight"))
print ("--> dCL/dAileronLeft  =", avl.output["CLtot"].deriv("AileronLeft"))
print ("--> dCL/dElevator     =", avl.output["CLtot"].deriv("Elevator"))
print ("--> dCL/dRudder       =", avl.output["CLtot"].deriv("Rudder"))
```

session06/4\_avl\_TransportControl.py

---

```
# Print CL derivatives
print ("\n==> Some stability derivatives")
print ("--> CLa  =", avl.output["CLa"].value ) # - Alpha
print ("--> CLb  =", avl.output["CLb"].value ) # - Beta
print ("--> CLp' =", avl.output["CLp'"].value) # - Roll rate
print ("--> CLq' =", avl.output["CLq'"].value) # - Pitch rate
print ("--> CLr' =", avl.output["CLr'"].value) # - Yaw rate

# Get neutral point, CG and MAC
Xnp = avl.output.Xnp
Xcg = avl.output.Xcg
MAC = transport.outpmtr.wing.mac

StaticMargin = (Xnp - Xcg)/MAC
print ("--> Xcg = ", Xcg)
print ("--> Xnp = ", Xnp)
print ("--> StaticMargin = ", StaticMargin)

ControlStability = avl.output.ControlStability
for key in ControlStability.keys():
    print("-->", key, ControlStability[key])
```

---

- AVL Overview
- AVL Geometry Definition
  - Plane Vanilla
  - Control Surfaces
  - Reference Quantities
  - Transport
- Stability Derivatives
- AVL Eigenmode Analysis
  - Pure AVL
  - AVL and masstran
- Linked Analysis Parameters
  - AVL and FRICTION
- Suggested Exercises



## Eigenmode analysis

- Stable configuration: all negative real Eigen values
- Requires realistic:
  - Configuration
  - Mass, CG, and inertia data
  - Flight conditions
- AVL Eigenmode analysis requires dimensional units
  - Geometry units defined with `capsLength` BODY attribute

ESP/viewVlm.udc

```
SELECT      BODY  bodyList [ibody+prevStack]  
ATTRIBUTE  capsLength      $ft
```

- Define units consistent with udunits2 convection
- Define the unit system
- Analysis unitSystem triggers inputs/outputs with units

## session06/5\_avl\_TransportEigen.py

```
# Define units
m      = pyCAPS.Unit("meter")
kg     = pyCAPS.Unit("kg")
s      = pyCAPS.Unit("s")
K      = pyCAPS.Unit("Kelvin")
deg    = pyCAPS.Unit("degree")
ft     = pyCAPS.Unit("ft")
slug   = pyCAPS.Unit("slug")
lb     = pyCAPS.Unit("lb")

unitSystem={"mass":kg, "length":m, "time":s, "temperature":K}

avl = capsProblem.analysis.create(aim = "avlAIM",
                                name = "avl",
                                unitSystem = unitSystem)

# Set analysis specific variables
avl.input.Mach = 0.5
avl.input.Alpha = 1.5*deg
avl.input.Beta = 0.0*deg
```

- Specifying Eigenmode Analysis dimensional inputs

## session06/5\_avl\_TransportEigen.py

```
# Inspired by the b737.mass avl example file
#
#      mass      CGx  CGy  CGz      Ixx      Iyy      Izz
cockpit   = {"mass": 3000 * lb, "CG": [ 8, 0, 5] * ft, "massInertia": [0. , 0. , 0. ] * lb*ft**2}
wing      = {"mass": 19420 * lb, "CG": [ 78, 0, -1] * ft, "massInertia": [8.0e6, 0.1e6, 8.1e6] * lb*ft**2}
fuselage  = {"mass": 33720 * lb, "CG": [105, 0, 2] * ft, "massInertia": [0.7e6, 18.9e6, 19.6e6] * lb*ft**2}
tailcone  = {"mass": 310 * lb, "CG": [145, 0, 0] * ft, "massInertia": [0. , 0. , 0. ] * lb*ft**2}
Htail     = {"mass": 528 * lb, "CG": [160, 0, 2] * ft, "massInertia": [0.0e6, 0.0e6, 0.0e6] * lb*ft**2}
Vtail     = {"mass": 616 * lb, "CG": [100, 0, 8] * ft, "massInertia": [0.1e6, 0.0e6, 0.1e6] * lb*ft**2}
Main_gear = {"mass": 4500 * lb, "CG": [ 76, 0, -4] * ft, "massInertia": [0.5e6, 0.0 , 0.5e6] * lb*ft**2}
Nose_gear = {"mass": 1250 * lb, "CG": [ 36, 0, -5] * ft, "massInertia": [0. , 0. , 0. ] * lb*ft**2}

avl.input.MassProp = {"cockpit" : cockpit ,
                      "wing"     : wing     ,
                      "fuselage" : fuselage ,
                      "tailcone" : tailcone ,
                      "Htail"    : Htail    ,
                      "Vtail"    : Vtail    ,
                      "Main_gear": Main_gear,
                      "Nose_gear": Nose_gear}

avl.input.Gravity = 32.18 * ft/s**2
avl.input.Density = 0.002378 * slug/ft**3
avl.input.Velocity = 250.0 * m/s
```

- Check bodies passed to avl and masstran

## session06/6\_avl\_masstran\_Geom.py

---

```
transport.cfgeomtr.VIEW.Concept   = 0
transport.cfgeomtr.VIEW.Vlm      = 1
transport.cfgeomtr.VIEW.Skins    = 1

# Enable fuselage and lifting surfaces
transport.cfgeomtr.COMP.wing     = 1
transport.cfgeomtr.COMP.fuse     = 1
transport.cfgeomtr.COMP.htail    = 1
transport.cfgeomtr.COMP.vtail    = 1
transport.cfgeomtr.COMP.controls = 0

# Create AVL AIM
avl = capsProblem.analysis.create(aim = "avlAIM",
                                  name = "avl")

print ("\n==> Vewing AVL geometry")
avl.geometry.view()

masstran = capsProblem.analysis.create(aim = "masstranAIM",
                                       name = "masstran")

print ("\n==> Vewing Masstran geometry")
masstran.geometry.view()
```

---

- Get mass properties from masstran

session06/7\_avl\_masstran\_Eigen.py

---

```
# Set materials
unobtainium = { "density" : 200 * lb/ft**3 }
masstran.input.Material = {"Unobtainium": unobtainium}

# Set property
shell = {"propertyType"      : "Shell",
         "material"          : "Unobtainium",
         "membraneThickness" : 0.02 * ft}

masstran.input.Property = {"fuseSkin" : shell,
                           "wingSkin" : shell,
                           "htailSkin": shell,
                           "vtailSkin": shell}

masstran.input.Edge_Point_Min = 3

print ("\n==> Computing mass properties...")
aircraft_mass = masstran.output.Mass
aircraft_CG   = masstran.output.CG
aircraft_I    = masstran.output.I_Vector

aircraft_skin = {"mass":aircraft_mass, "CG":aircraft_CG, "massInertia":aircraft_I}

print ("\n==> Computed mass properties...")
print ("--> aircraft_skin = ", aircraft_skin)
```

- Pass mass properties to AVL

## session06/7\_avl\_masstran\_Eigen.py

---

```

#               mass               CGx  CGy  CGz               Ixx   Iyy  Izz
cockpit   = {"mass": 3000 * lb, "CG":[ 8,  0,  5] * ft, "massInertia":[0.   , 0.   , 0.   ] * lb*ft**2}
Main_gear = {"mass": 4500 * lb, "CG":[ 76,  0, -4] * ft, "massInertia":[0.5e6, 0.0   , 0.5e6] * lb*ft**2}
Nose_gear = {"mass": 1250 * lb, "CG":[ 36,  0, -5] * ft, "massInertia":[0.   , 0.   , 0.   ] * lb*ft**2}

avl.input.MassProp = {"aircraft_skin": aircraft_skin,
                      "cockpit"      : cockpit      ,
                      "Main_gear"    : Main_gear     ,
                      "Nose_gear"    : Nose_gear     }

avl.input.Gravity   = 32.18 * ft/s**2
avl.input.Density   = 0.002378 * slug/ft**3
avl.input.Velocity  = 250.0 * m/s

```

---

- AVL Overview
- AVL Geometry Definition
  - Plane Vanilla
  - Control Surfaces
  - Reference Quantities
  - Transport
- Stability Derivatives
- AVL Eigenmode Analysis
  - Pure AVL
  - AVL and masstran
- **Linked Analysis Parameters**
  - **AVL and FRICTION**
- Suggested Exercises

## FRICITION

- Virginia Tech Aerodynamics and Design Software Collection
  - Estimates subsonic skin friction and form drag
  - Suitable for aircraft preliminary design
- 
- Link FRICITION output drag to AVL input profile drag
  - Couples DIRTY/CLEAN process
  - Automatically transfers data
  - Data “shape” and “type” must be the same

session06/8\_avl\_friction\_Link.py

---

```
avl.input["CDp"].link(friction.output["CDtotal"])
```

---



- Both FRICTION and AVL have “Mach” input (must be consistent)
- Create a common analysis parameter and link to AVL/FRICTION input

## session06/8\_avl\_friction\_Link.py

```
# Create shared analysis input parameter
mach = capsProblem.parameter.create("Mach", 0.1)

friction.input["Mach"].link(mach)

avl.input["Mach"].link(mach)

machRange = [mach.value + i*0.1 for i in range(7)]
for M in machRange:
    # Set mission parameters - AIMS will be updated automatically due to link
    mach.value = M
```

## Multiple Shells

- Create multiple materials and shell properties for the transport components in `7_avl_masstran_Eigen.py`

## Multiple AIMs

- Create multiple masstran AIM instances for the transport components in `7_avl_masstran_Eigen.py`
  - Note: `viewOmlStructure` has same `capsIntent` as F-118

## Main Gear

- Use `wing:xroot` and `wing:mac` to position the main gear CGx in `7_avl_masstran_Eigen.py` as a fraction of `wing:mac` downstream of `wing:xroot`

## Stable Transport

- Resize tail and modify mass properties of 7\_avl\_masstran\_Eigen.py to make a stable transport (all negative real Eigen values)
- Create your own (optionally share it [galbramc@mit.edu](mailto:galbramc@mit.edu))