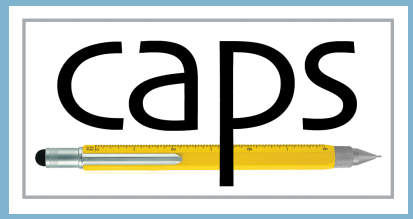
Computational Aircraft Prototype Syntheses



Training Session 3
CAPS with Gradient-based Optimization

ESP v1.26

Marshall Galbraith galbramc@mit.edu
Bob Haimes haimes@mit.edu
Massachusetts Institute of Technology

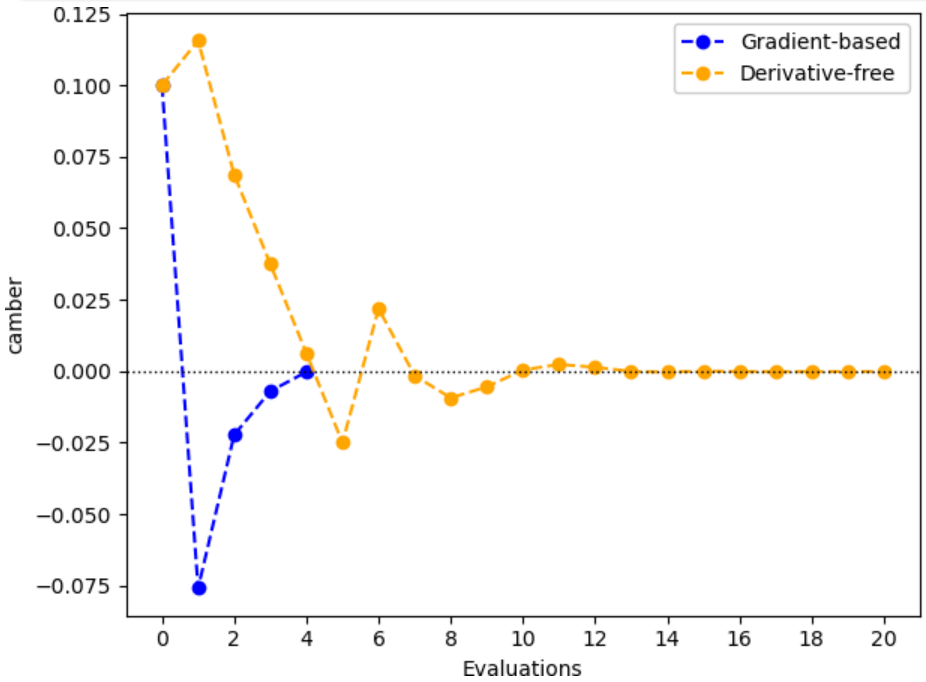
John F. Dannenhoffer, III
jfdannen@syr.edu
Syracuse University

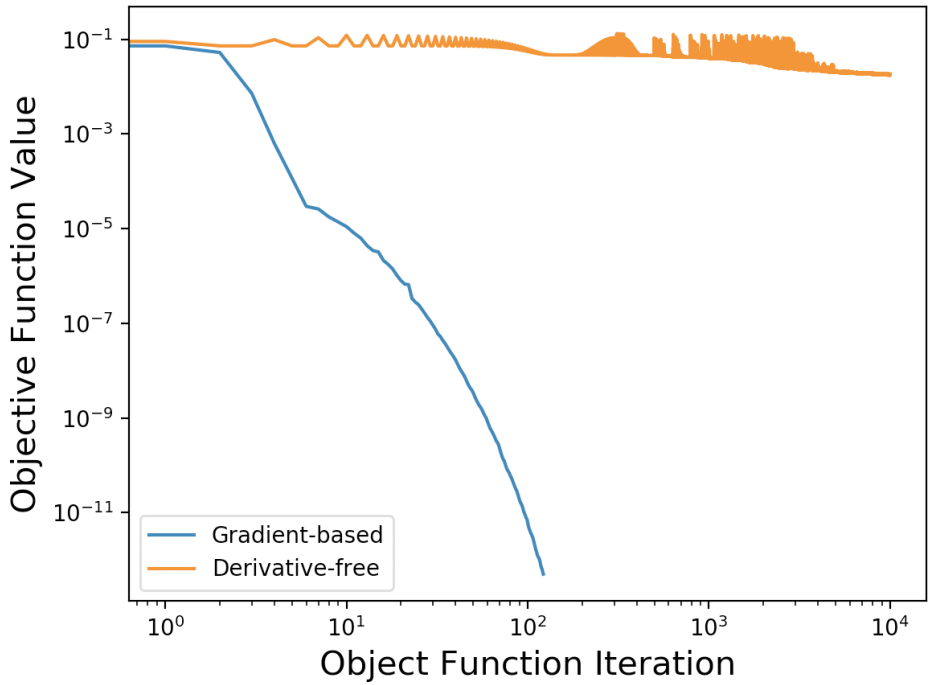
Gradient Based Geometric Parameter Optimization

NACA airfoil, $\alpha = 0°$

$$\min_{\text{camber}} \; C_D$$

L^2 minimization with
200 design parameters





Overview

- MSES Overview
- Accessing Functional Derivatives
- Gradient-Based Optimization with Analysis Input
- Gradient-Based Optimization with Geometric Inputs
- Gradient-based Optimization for Structures
- Suggested Exercises

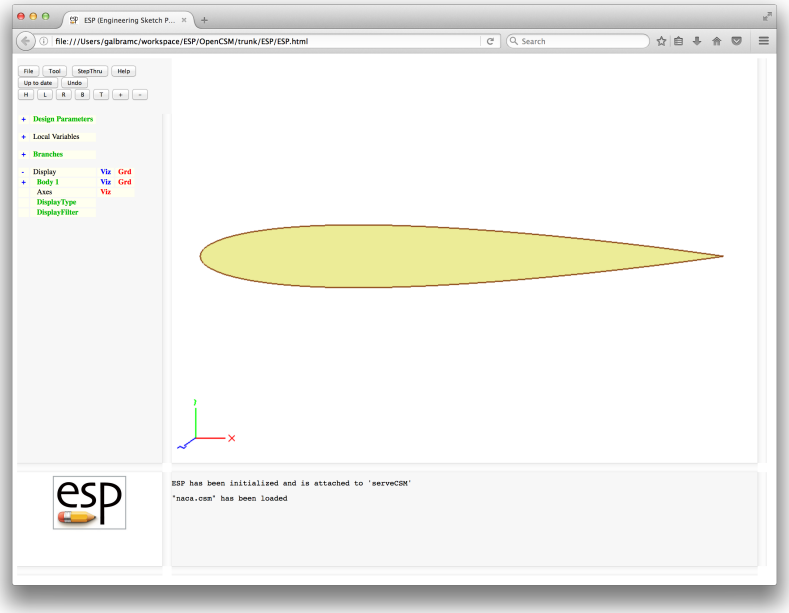
MSES and NACA Airfoil Geometry

MSES

- Multi-Element airfoil performance analysis
- Integral Boundary Layer Theory + Euler
  - Transonic flow
- e^N Transition Model
- Analysis and Geometric Sensitivities

session03/naca.csm

```
# NACA design paramters
DESPMTR   thick     0.12      #frac of local chord
DESPMTR   camber    0.00      #frac of local chord

# Construct the airfoil
UDPRIM  naca   Thickness thick   Camber  camber
    ATTRIBUTE capsAIM $xfoilAIM;msesAIM;tsfoilAIM

# output the area
OUTPMTR Area
SET Area @area
```



MSES AIM Documentation

Overview

- MSES Overview
- Accessing Functional Derivatives
- Gradient-Based Optimization with Analysis Input
- Gradient-Based Optimization with Geometric Inputs
- Gradient-based Optimization for Structures
- Suggested Exercises

Accessing output derivatives

- Derivatives or OUTPMTR w.r.t DESPMTR

session03/1_mses_Deriv.py

```
filename = "naca.csm"
print ("\n==> Loading geometry from file \""+filename+"\"...")
capsProblem = pyCAPS.Problem(problemName = "design_mses",
                             capsFile = filename,
                             outLevel = 0)

# Alias the geometry
naca = capsProblem.geometry

print('\n==> Area and Area derivatives...')
print("--> Area          ", naca.outpmtr["Area"].value)
print("--> dArea/dthick ", naca.outpmtr["Area"].deriv("thick"))
```

Accessing output derivatives

- Create MSES AIM and set analysis inputs

session03/1_mses_Deriv.py

```
# Setup MSES
mses = capsProblem.analysis.create(aim = "msesAIM", name="mses")

# Set flow condition
mses.input.Alpha = 0.0
mses.input.Mach  = 0.5
mses.input.Re = 5e6

# Set meshing parameters
mses.input.GridAlpha = 0
mses.input.Airfoil_Points = 201
```

Accessing output derivatives

- Desired derivative variables set with Design_Variable input
  - MSES always computes derivatives w.r.t. Alpha, Mach, Re,
- Output derivatives w.r.t. Analysis and Geometric inputs accessed via deriv method

session03/1_mses_Deriv.py

```
# Geometric derivative variables
mses.input.Design_Variable = {"camber":{}, "thick":{}}

# Print the functional and gradients
print('\n==> Drag and Drag derivatives...')
print( "--> CD          =", mses.output["CD"].value )
print( "--> dCD/dAlpha  =", mses.output["CD"].deriv("Alpha") )
print( "--> dCD/dMach   =", mses.output["CD"].deriv("Mach") )
print( "--> dCD/dcamber =", mses.output["CD"].deriv("camber") )
print( "--> dCD/dthick  =", mses.output["CD"].deriv("thick") )

print('\n==> All Drag derivatives...')
print( "--> CD derivs   =", mses.output["CD"].deriv() )
```

Overview

- MSES Overview
- Accessing Functional Derivatives
- Gradient-Based Optimization with Analysis Input
- Gradient-Based Optimization with Geometric Inputs
- Gradient-based Optimization for Structures
- Suggested Exercises

Plotting and verifying output α derivatives

- Create MSES AIM and set analysis inputs

session03/2_mses_DerivPlot_CD_Alpha.py

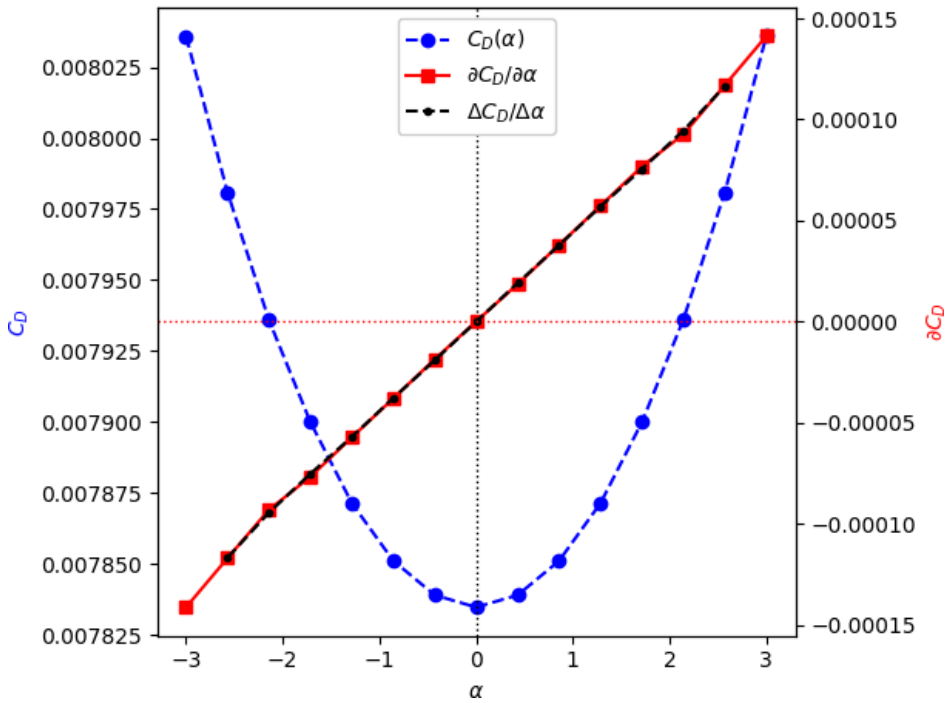
```
# Setup MSES
mses = capsProblem.analysis.create(aim = "msesAIM", name="mses")

# Set flow condition
mses.input.Alpha = 0.0
mses.input.Mach  = 0.5
mses.input.Re = 5e6

# Set meshing parameters
mses.input.GridAlpha = 0
mses.input.Airfoil_Points = 201

# Trip the flow near the leading edge to get smooth gradient
mses.input.xTransition_Upper = 0.1
mses.input.xTransition_Lower = 0.1
```

Plotting and verifying output α derivatives

- Analytic vs. Central Difference derivatives
- Plot vs. Alpha
  - C_d, $\partial C_d/\partial\alpha$, $\Delta C_d/\Delta\alpha$

session03/2_mses_DerivPlot_CD_Alpha.py

```
# Plot the functional and gradient
Alphas = npy.linspace(-3, 3, 15)
CD = []
CD_Alpha = []
for Alpha in Alphas:
    print("--> Alpha", Alpha)
    mses.input.Alpha = Alpha

    CD.append(      mses.output["CD"].value )
    CD_Alpha.append(mses.output["CD"].deriv("Alpha") )

# Compute central difference derivatives
dCD_Alpha = []
for i in range(1,len(Alphas)-1):
    dCD_Alpha.append( \
      (CD[i+1]-CD[i-1])/(Alphas[i+1]-Alphas[i-1]))
```

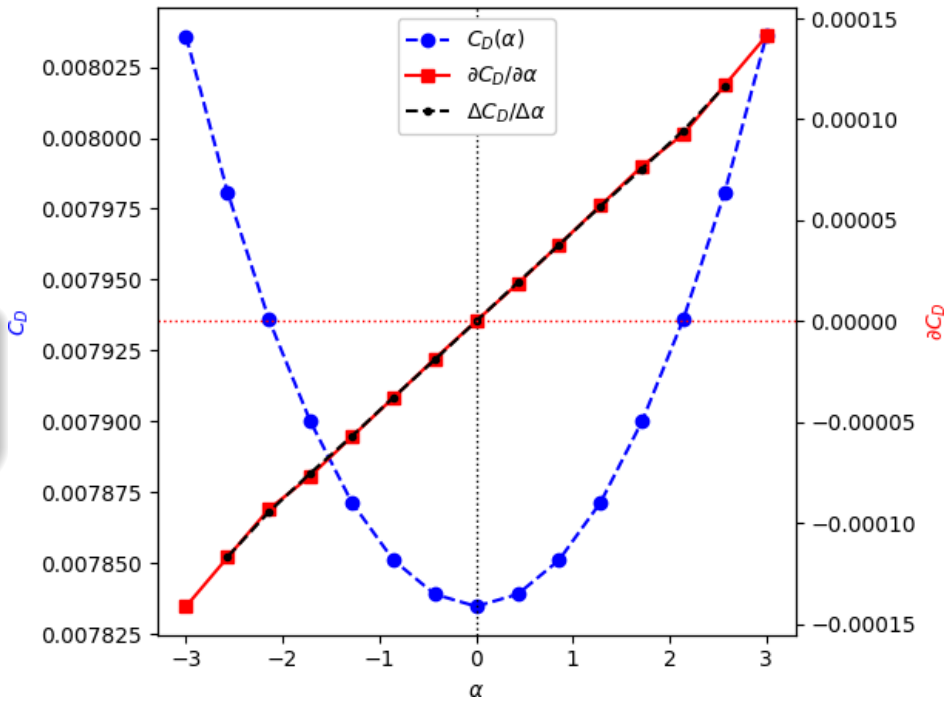


Optimization with Analysis Input Sensitivity

- Gradient based optimization with OpenMDAO[1]

$$\min_{\alpha} \; C_D \tag{1}$$

- Symmetric NACA 0012
- Exact solution: $\alpha = 0$



[1] https://openmdao.org

OpenMDAO: MSES AIM Inputs

- Basic analysis inputs

session03/3_mses_OpenMDAO_CD_Alpha.py

```
capsProblem = pyCAPS.Problem(problemName="design_mses",
                             capsFile="naca.csm",
                             outLevel=0)

# Setup MSES
mses = capsProblem.analysis.create(aim = "msesAIM", name="mses")

# Set flow condition
mses.input.Alpha = 3.0   # Initial guess away from solution Alpha == 0
mses.input.Mach  = 0.5
mses.input.Re = 5e6

# Set meshing parameters
mses.input.GridAlpha = 0
mses.input.Airfoil_Points = 201

# Trip the flow near the leading edge to get smooth gradient
mses.input.xTransition_Upper = 0.1
mses.input.xTransition_Lower = 0.1
```

OpenMDAO Component: Initialize and Setup

- Define output functional and partial derivatives

session03/3_mses_OpenMDAO_CD_Alpha.py

```
class msesAnalysis(om.ExplicitComponent):

    def initialize(self):
        # Setup arguments for the component.

        # CAPS Problem input
        self.options.declare('capsProblem', types=object)

    def setup(self):
        # Assign initial values to the variables.

        capsProblem = self.options['capsProblem']
        mses = capsProblem.analysis['mses']

        # attach parameters to OpenMDAO object
        self.add_input('Alpha', val=mses.input.Alpha)

        # Add output metric
        self.add_output('CD')

        # Declare and attach partials to self
        self.declare_partials('CD','Alpha')
```

OpenMDAO Component: Compute

- Set *alpha* based on "inputs" from OpenMDAO
- Return functional C_D with "outputs"

session03/3_mses_OpenMDAO_CD_Alpha.py

```
    def compute(self, inputs, outputs):
        # Compute functionals

        capsProblem = self.options['capsProblem']
        mses = capsProblem.analysis['mses']

        print("--> Alpha", inputs['Alpha'])

        # Update input values if changed
        if mses.input.Alpha != inputs['Alpha']:
            mses.input.Alpha = inputs['Alpha']

        # Grab objective and attach as an output
        outputs['CD'] = mses.output.CD
        print("--> CD", outputs['CD'])
```

OpenMDAO Component: Compute Partials

- Return $\partial C_D/\partial \alpha$ with "partials"

session03/3_mses_OpenMDAO_CD_Alpha.py

```
    def compute_partials(self,inputs,partials):
        # Compute functional partial derivatives

        capsProblem = self.options['capsProblem']
        mses = capsProblem.analysis['mses']

        print("--> Alpha", inputs['Alpha'])

        # Update input values if changed
        if mses.input.Alpha != inputs['Alpha']:
            mses.input.Alpha = inputs['Alpha']

        # Get derivatives and set partials
        partials['CD', 'Alpha'] = mses.output["CD"].deriv("Alpha")
        print("--> CD_alpha", partials['CD', 'Alpha'])
```

OpenMDAO: Execution

session03/3_mses_OpenMDAO_CD_Alpha.py

```
# Setup the openmdao problem object
omProblem = om.Problem()

# Create the OpenMDAO component
msesSystem = msesAnalysis(capsProblem = capsProblem)

# add subsystem to model
omProblem.model.add_subsystem('msesSystem', msesSystem)

# add design variables to model
omProblem.model.add_design_var('msesSystem.Alpha', lower=-5, upper=5)

# add objective to minimize CD
omProblem.model.add_objective('msesSystem.CD')

# setup the optimization
omProblem.driver = om.ScipyOptimizeDriver()
omProblem.driver.options['optimizer'] = "L-BFGS-B"
omProblem.driver.options['tol'] = 1.e-9
omProblem.driver.options['disp'] = True

# Press go
print ("\n==> Starting Optimization...")
omProblem.setup()
omProblem.run_driver()
omProblem.cleanup()

print("--> Optimized value:", omProblem.get_val("msesSystem.Alpha"))
```

Overview

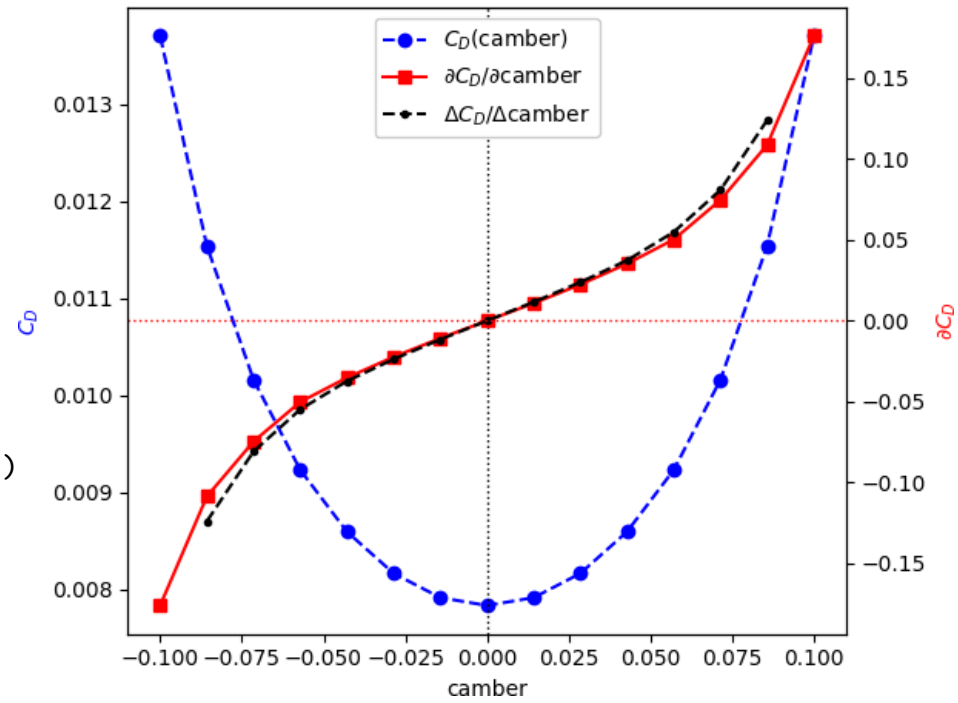
- MSES Overview
- Accessing Functional Derivatives
- Gradient-Based Optimization with Analysis Input
- Gradient-Based Optimization with Geometric Inputs
- Gradient-based Optimization for Structures
- Suggested Exercises

Plotting and verifying output "camber" derivatives

- Analytic vs. Central Difference derivatives
- Plot vs. camber
  - C_d, $\partial C_d/\partial$camber, $\Delta C_d/\Delta$camber

session03/4_mses_DerivPlot_CD_camber.py

```
# Use camber as the design variable
mses.input.Design_Variable = {"camber":{}}

# Plot the functional and gradient
cambers = npy.linspace(-0.1, 0.1, 15)
CD = []
CD_camber = []
for camber in cambers:
    print("--> camber", camber)
    capsProblem.geometry.despmtr.camber = camber

    CD.append(       mses.output["CD"].value )
    CD_camber.append(mses.output["CD"].deriv("camber") )

# Compute central difference derivatives
dCD_camber = []
for i in range(1,len(cambers)-1):
    dCD_camber.append( \
      (CD[i+1]-CD[i-1])/(cambers[i+1]-cambers[i-1]))
```

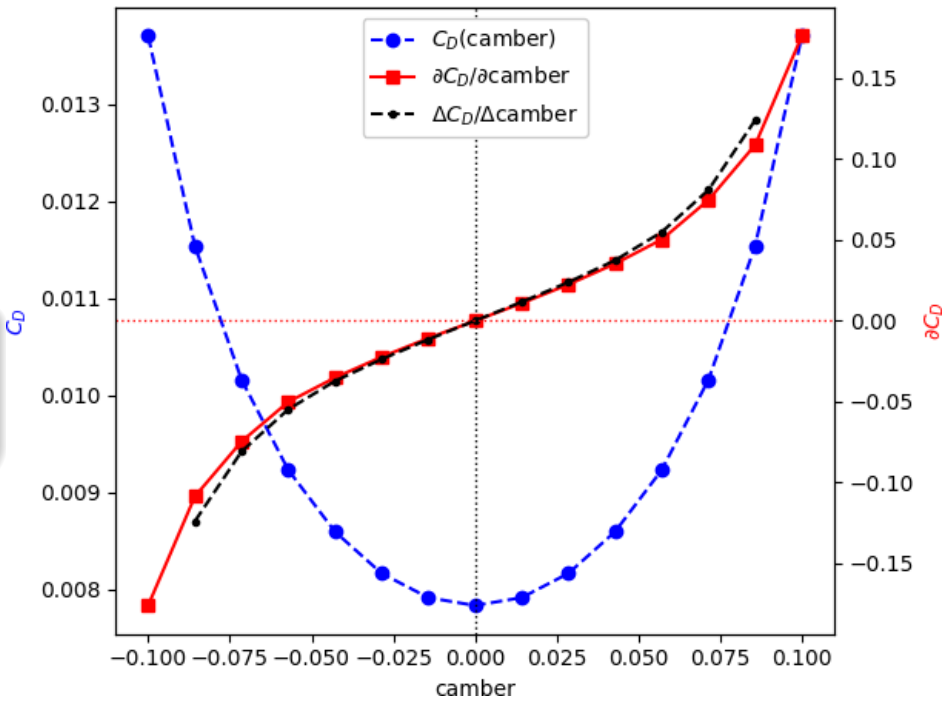


Optimization with Geometric Input Sensitivity

- Gradient based optimization with OpenMDAO

$$\min_{\text{camber}} \; C_D \tag{2}$$

- NACA Airfoil
- Exact solution: camber = 0



OpenMDAO: MSES AIM Inputs

- Basic analysis inputs

session03/5_mses_OpenMDAO_CD_camber.py

```
# Set a non-zero camber. The optimization should drive it back to zero
capsProblem.geometry.despmtr.camber = 0.1

# Setup MSES
mses = capsProblem.analysis.create(aim = "msesAIM", name="mses")

# Set flow condition
mses.input.Alpha = 0.0
mses.input.Mach  = 0.5
mses.input.Re = 5e6

# Set meshing parameters
mses.input.GridAlpha = 0
mses.input.Airfoil_Points = 201

# Trip the flow near the leading edge to get smooth gradient
mses.input.xTransition_Upper = 0.1
mses.input.xTransition_Lower = 0.1

# Compute derivatives w.r.t. camber
mses.input.Design_Variable = {"camber":{}}
```

OpenMDAO Component: Setup

- Define output functional and partial derivatives

session03/5_mses_OpenMDAO_CD_camber.py

```
    def setup(self):
        # Assign initial values to the variables.

        capsProblem = self.options['capsProblem']
        naca = capsProblem.geometry

        # attach parameters to OpenMDAO object
        self.add_input('camber', val=naca.despmtr.camber)

        # Add output metric
        self.add_output('CD')

        # Declare and attach partials to self
        self.declare_partials('CD','camber')
```

OpenMDAO Component: Compute

- Set camber based on "inputs" from OpenMDAO
- Return functional C_D with "outputs"

session03/5_mses_OpenMDAO_CD_camber.py

```
def compute(self, inputs, outputs):
    # Compute functionals

    capsProblem = self.options['capsProblem']
    naca = capsProblem.geometry
    mses = capsProblem.analysis['mses']

    print("--> camber", inputs['camber'])

    # Update input values if changed
    if naca.despmtr.camber != inputs['camber']:
        naca.despmtr.camber = inputs['camber']

    # Grab objective and attach as an output
    outputs['CD']  = mses.output.CD
```

OpenMDAO Component: Compute Partials

- Return $\partial C_D/\partial \text{camber}$ with "partials"

session03/5_mses_OpenMDAO_CD_camber.py

```
    def compute_partials(self,inputs,partials):
        # Compute functional partial derivatives

        capsProblem = self.options['capsProblem']
        naca = capsProblem.geometry
        mses = capsProblem.analysis['mses']

        print("--> camber    ", inputs['camber'])

        # Update input values if changed
        if naca.despmtr.camber != inputs['camber']:
            naca.despmtr.camber = inputs['camber']

        # Get derivatives and set partials
        partials['CD', 'camber'] = mses.output["CD"].deriv("camber")
        print("--> CD_camber", partials['CD', 'camber'])
```

OpenMDAO: Execution

session03/5_mses_OpenMDAO_CD_camber.py

```
# Create the OpenMDAO component
msesSystem = msesAnalysis(capsProblem = capsProblem)

# add subsystem to model
omProblem.model.add_subsystem('msesSystem', msesSystem)

# add design variables to model
omProblem.model.add_design_var('msesSystem.camber', lower=-0.1, upper=0.1)

# add objective to minimize CD
omProblem.model.add_objective('msesSystem.CD')

# setup the optimization
omProblem.driver = om.ScipyOptimizeDriver()
omProblem.driver.options['optimizer'] = "L-BFGS-B"
omProblem.driver.options['tol'] = 1.e-5
omProblem.driver.options['disp'] = True

# Press go
print ("\n==> Starting Optimization...")
omProblem.setup()
omProblem.run_driver()
omProblem.cleanup()

print("--> Optimized value:", omProblem.get_val("msesSystem.camber"))
```

Overview

- MSES Overview
- Accessing Functional Derivatives
- Gradient-Based Optimization with Analysis Input
- Gradient-Based Optimization with Geometric Inputs
- Gradient-based Optimization for Structures
- Suggested Exercises

Structural Optimization with Analsysis Variables

- Gradient based optimization with OpenMDAO
- Analysis Input: Wing shell membraneThickness (δ_w)

session03/6_masstran_OpenMDAO_thick.py

```
shellWing = {"propertyType"      : "Shell",
             "membraneThickness" : 0.1,
             "material"          : "unobtainium"}
masstran.input.Property = {"wing" : shellWing, "htail": shellTail,
                           "vtail": shellTail, "fuse" : shellFuse}
```

- Optimization statement

$$\min_{\delta_w} \left(X_{cg}(\delta_w) - X_{cg\,design}\right)^2 \tag{3}$$

- $X_{cg}(\delta_w)$ computed with masstran
- $X_{cg\,design}$ specified desired CG location (F-118D.csm)

Design Variable and Design Variable Relation

- CAPS uses the NASTRAN model for design variables
- Independent "Design Variable": x_1, x_2, x_3...
- Dependent "Design Variable Relation"
  - Links independent design variables to NASTRAN Inputs
  - constantCoeff: c_0
  - linearCoeff: c_1, c_2...

$$\delta_w(x_1, x_2, x_3...) = c_0 + \sum_{i=1} c_i x_i \tag{4}$$

Design Variable and Design Variable Relation

session03/6_masstran_OpenMDAO_thick.py

```
shellFuse = {"propertyType"      : "Shell",
             "membraneThickness" : 0.2,
             "material"          : "madeupium"}
shellWing = {"propertyType"      : "Shell",
             "membraneThickness" : 0.1,
             "material"          : "unobtainium"}
masstran.input.Property = {"wing" : shellWing, "htail": shellTail,
                           "vtail": shellTail, "fuse" : shellFuse}

# Specify design variables
masstran.input.Design_Variable = {
    "x1": {"initialValue":shellWing["membraneThickness"]},
    "x2": {"initialValue":0},
}

# Set design variable relations
relationWing = {"componentType": "Property",
                "componentName": "wing",
                "fieldName" : "membraneThickness",
                "constantCoeff" : 0.0,
                "variableName" : ["x1","x2"],
                "linearCoeff" : [1.0, 0.1]}
relationFuse = {"componentType": "Property",
                "componentName": "fuse",
                "fieldName" : "membraneThickness",
                "constantCoeff" : 0,
                "variableName" : "x1",
                "linearCoeff" : 1.0}
masstran.input.Design_Variable_Relation = {"wingRel":relationWing,"fuseRel":relationFuse}
```

OpenMDAO: masstran AIM Inputs

session03/6_masstran_OpenMDAO_thick.py

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

# Set mesh generation inputs
masstran.input.Edge_Point_Min = 20
masstran.input.Edge_Point_Max = 20
masstran.input.Quad_Mesh = True

# Set materials
madeupium    = {"materialType" : "isotropic",
                "density"      : 10}
unobtainium  = {"materialType" : "isotropic",
                "density"      : 25}
masstran.input.Material = {"madeupium": madeupium, "unobtainium": unobtainium}

# Set properties
shellTail = {"propertyType"      : "Shell",
             "membraneThickness" : 0.5,
             "material"          : "unobtainium"}
shellFuse = {"propertyType"      : "Shell",
             "membraneThickness" : 0.2,
             "material"          : "madeupium"}
shellWing = {"propertyType"      : "Shell",
             "membraneThickness" : 0.1,
             "material"          : "unobtainium"}
masstran.input.Property = {"wing" : shellWing, "htail": shellTail,
                           "vtail": shellTail, "fuse" : shellFuse}
```

OpenMDAO Component: Setup

- Define output functional and partial derivatives

session03/6_masstran_OpenMDAO_thick.py

```
    def setup(self):
        # Assign initial values to the variables.

        capsProblem = self.options['capsProblem']
        geometry = capsProblem.geometry
        masstran = capsProblem.analysis['masstran']

        # attach parameters to OpenMDAO object
        self.add_input('x1', val=masstran.input.Design_Variable["x1"]["initialValue"])
        self.add_input('x2', val=masstran.input.Design_Variable["x2"]["initialValue"])

        # Add output metric
        self.add_output('dCGx')

        # Declare and attach partials to self
        self.declare_partials('dCGx','x1')
        self.declare_partials('dCGx','x2')
```

OpenMDAO Component: Compute

- Return functional dCGx with "outputs"

session03/6_masstran_OpenMDAO_thick.py

```
    def compute(self, inputs, outputs):
        # Compute functionals

        capsProblem = self.options['capsProblem']
        geometry = capsProblem.geometry
        masstran = capsProblem.analysis['masstran']

        print("--> x1", inputs['x1'])
        print("--> x2", inputs['x2'])

        # Update input values
        if masstran.input.Design_Variable["x1"]["initialValue"] != inputs['x1'] or \
           masstran.input.Design_Variable["x2"]["initialValue"] != inputs['x2']:
            masstran.input.Design_Variable = {
                "x1": {"initialValue":float(inputs['x1'])},
                "x2": {"initialValue":float(inputs['x2'])},
            }

        x_cg_d  = geometry.despmtr.xcg
        x_cg    = masstran.output.CG[0]

        # Grab objective and attach as an output
        outputs['dCGx'] = (x_cg - x_cg_d)**2
        print("--> dCGx", outputs['dCGx'])
```

OpenMDAO Component: Compute Partials

session03/6_masstran_OpenMDAO_thick.py

```
    def compute_partials(self,inputs,partials):
        # Compute functional partial derivatives

        capsProblem = self.options['capsProblem']
        geometry = capsProblem.geometry
        masstran = capsProblem.analysis['masstran']

        print("--> x1", inputs['x1'])
        print("--> x2", inputs['x2'])

        # Update input values
        if masstran.input.Design_Variable["x1"]["initialValue"] != inputs['x1'] or \
           masstran.input.Design_Variable["x2"]["initialValue"] != inputs['x2']:
            masstran.input.Design_Variable = {
                "x1": {"initialValue":float(inputs['x1'])},
                "x2": {"initialValue":float(inputs['x2'])},
            }

        x_cg_d  = geometry.despmtr.xcg
        x_cg    = masstran.output.CG[0]

        cg_x1 = masstran.output["CG"].deriv("x1")[0]
        cg_x2 = masstran.output["CG"].deriv("x2")[0]

        # Get derivatives and set partials
        partials['dCGx', 'x1'] = 2*(x_cg - x_cg_d)*cg_x1
        partials['dCGx', 'x2'] = 2*(x_cg - x_cg_d)*cg_x2
        print("--> dCGx_x1", partials['dCGx', 'x1'])
        print("--> dCGx_x2", partials['dCGx', 'x2'])
```

Suggested Exercises

Thickness Derivatives

- Plot C_d and $\partial C_d / \partial \text{thick}$
  - Start from a copy of session03/4_mses_DerivPlot_CD_camber.py

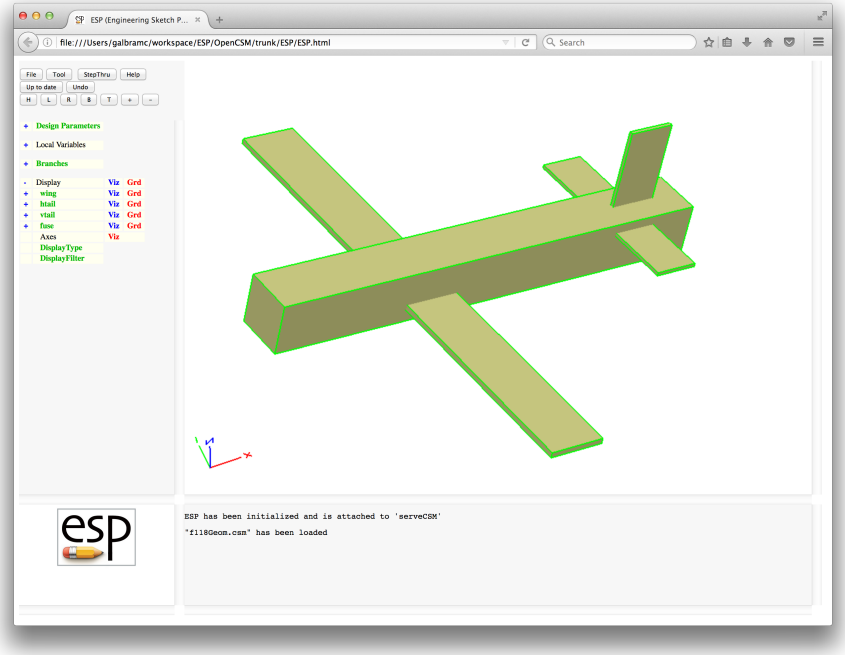
CG Optimization with Geometry Variables

- Optimize wing:xroot and htail:lh in F-118D such that the CG computed with masstran matches the design CG

F-118D Boxter Geometry

session03/f118-D.csm

```
# F-118D Boxster

DESPMTR   xcg                 9     # Desired x cg location

# wing design parameters
DESPMTR   wing:area          30     # area
DESPMTR   wing:aspect      9.00     # aspect ratio
DESPMTR   wing:thick       0.10     # thickness ratio
DESPMTR   wing:xroot        4.0     # xloc at root LE
DESPMTR   wing:zroot       -0.5     # zloc at root LE

# horizontal tail design parameters
DESPMTR   htail:lh            5     # htail moment length
DESPMTR   htail:aspect     4.15     # htail aspect ratio
DESPMTR   htail:thick      0.08     # htail thickness
DESPMTR   htail:Vh          0.8     # htail volume coeff
DESPMTR   htail:zroot       0.5     # zloc of root LE

# vertical tail design parameters
DESPMTR   vtail:aspect     1.80     # vtail aspect ratio
DESPMTR   vtail:tau        0.08     # vtail thickness
DESPMTR   vtail:Vv         0.03     # vtail volume coeff
DESPMTR   vtail:yroot         0     # yloc of root LE

# fuselage design parameters
DESPMTR   fuse:width        1.2     # vtail area
DESPMTR   fuse:height       1.2     # vtail aspect ratio
```



Move wing and tail s.t. CG from masstran matches design CG

F-118D Boxter Geometry

- Htail and Vtail sized by tail volume coefficient

$$V_h = \frac{S_h L_h}{S_w C_w} \quad \rightarrow \quad S_h = \frac{V_h S_w C_w}{L_h}$$

- L_h and X_w are free design parameters

session03/f118-D.csm

```
#================================================
# Htail
SET       htail:area      htail:Vh*wing:area*wing:chord/htail:lh
SET       htail:span      sqrt(htail:aspect*htail:area)
SET       htail:chord     htail:area/htail:span
SET       htail:xroot     xcg+htail:lh-htail:chord/4

BOX  htail:xroot  -htail:span/2  htail:zroot   htail:chord  htail:span  htail:chord*htail:thick
#================================================
# fuselage
SET   fuse:length   xcg+htail:lh+3*htail:chord/4

BOX  0  -fuse:width/2  -fuse:height/2   fuse:length  fuse:width  fuse:height
```

CG Optimization with Geometry Variables

- Gradient based optimization with OpenMDAO
- Geometry Input variables: wing:xroot, htail:lh

$$\min_{\text{wing:xroot,htail:lh}} \left(X_{cg} - X_{cg\,design}\right)^2 \tag{4}$$

- X_{cg} computed with masstran
- $X_{cg\,design}$ specified desired CG location (F-118D.csm)