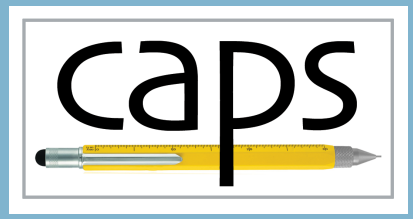
Computational Aircraft Prototype Syntheses



Training Session 6
CFD Analysis: Cart3D/Fun3D+refine
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

Overview

- CFD analysis with Cart3D
- Optimization with Cart3D
  - Geometric Parameter
- CFD analysis with Fun3D
- Fun3D with refine (sketch-to-solution)
- Optimization with Fun3D
  - Re-meshing
  - Mesh-morphing

Cart3D Analysis Outline

- Load geometry
- Create CFD AIM
- Set CFD analysis inputs
- Execute CFD
- Extract analysis outputs

session06/1_cart3d_InviscidWing.py

Cart3D AIM Analysis Inputs

session06/1_cart3d_InviscidWing.py

```
# Create Cart3D AIM
cart3d = capsProblem.analysis.create(aim = "cart3dAIM")

cart3d.geometry.view()

# Set inputs
cart3d.input.Mach          = 0.8
cart3d.input.alpha         = 1.5
cart3d.input.maxCycles     = 10
cart3d.input.nDiv          = 6
cart3d.input.maxR          = 9
cart3d.input.outer_box     = 8
cart3d.input.y_is_spanwise = True

cart3d.input.Model_X_axis = "-Xb"
cart3d.input.Model_Y_axis = "-Yb"
cart3d.input.Model_Z_axis = "-Zb"

span = transport.outpmtr.wing.span
cart3d.input.Yslices = [0, 0.5*span, span]

# Set inputs added directly to aero.csh
aerocsh = ["set it_fc = 150",
           "set it_ad = 150"]

if useMPI:
    aerocsh.append("set flowCart = mpix_flowCart")
    aerocsh.append("set mpi_prefix = 'mpiexec -n 16'")

cart3d.input.aerocsh = aerocsh
```

Cart3D Execution

- Cart3D is excued via a modified aero.csh
  - aero.csh is copied from $CART3D/bin directory

session06/2_cart3d_InviscidWing.py

```
# Explicitly Run Cart3D via aero.csh (optional)
cart3d.runAnalysis()

# ----------------------------------------------------------------
# Print outputs
print ("C_A   ", cart3d.output.C_A)
print ("C_Y   ", cart3d.output.C_Y)
print ("C_N   ", cart3d.output.C_N)
print ("C_D   ", cart3d.output.C_D)
print ("C_S   ", cart3d.output.C_S)
print ("C_L   ", cart3d.output.C_L)
print ("C_l   ", cart3d.output.C_l)
print ("C_m   ", cart3d.output.C_m)
print ("C_n   ", cart3d.output.C_n)
print ("C_M_x ", cart3d.output.C_M_x)
print ("C_M_y ", cart3d.output.C_M_y)
print ("C_M_z ", cart3d.output.C_M_z)
```

Overview

- CFD analysis with Cart3D
- Optimization with Cart3D
  - Geometric Parameter
- CFD analysis with Fun3D
- Fun3D with refine (sketch-to-solution)
- Optimization with Fun3D
  - Re-meshing
  - Mesh-morphing

CAPS+Cart3D Development

- Collaborative effort by
  - Marshall Galbraith and Bob Haimes (MIT)
  - Marian Nemec and Micheal Aftosmis (NASA Ames)
- Features added to CAPS framework and Cart3D AIM
- Cart3D design framework scripts updated
  - c3d_objGrad.csh updated to support execution outside Cart3D design framework
  - Updated xddm C-library for Cart3D design xml files
- Updated Cart3D design framework released with v1.5.9

Two Example Optimization Problems

- Minimize C_L^2
  - Geometry Input wing:alphar (root twist) design variables
  - Analysis Input α design variable (exercise)

Cart3D Optimization with Geometric Sensitivities

- Gradient based optimization

$$\min_{\text{wing:alphar}} \quad C_L^2 \tag{1}$$

- Hershey-bar wing with symmetric airfoil
  - Twist angle defined at root airfoil section
  - Exact solution: wing : alphar $= 0$
- Use Cart3D to compute C_L^2 and $\partial C_L^2 / \partial$(wing:alphar)
  - CAPS computes geometric sensitivities

Cart3D AIM Design Analysis Inputs

- Design analysis inputs
  - Design_Functional: Functionals defined in Cart3D design.xml
  - Design_Variable: Variables for partial derivatives

session06/2_cart3d_OpenMDAO_twist.py

```
# Setup Cl**2 as an output functional
cart3d.input.Design_Functional = {"CL2": {"function":"C_L", "power":2}}

# Declare design variables
cart3d.input.Design_Variable = {"wing:alphab":""}
```

ESP/wingPmtrs.udc

```
DESPMTR   wing:alphar      -1.0    # setting angle   at root
```

OpenMDAO Component: Setup

- Define output functional and partial derivatives for OpenMDAO

session06/2_cart3d_OpenMDAO_twist.py

```
class Cart3dAnalysis(om.ExplicitComponent):

    def initialize(self):
        # Setup variables so that arguments can be provided to the component.

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

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

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

        # Add free design parameters to OpenMDAO object
        self.add_input('wing:alphab', val=geometry.despmtr.wing.alphab)

        # Add outputs
        self.add_output('CL2')

        # Add partials
        self.declare_partials('CL2','wing:alphab')
```

OpenMDAO Component: Compute

- Compute the functional C_L^2
- Set wing:alphar based on "inputs" from OpenMDAO
- Disable Design_Sensitvity

session06/2_cart3d_OpenMDAO_twist.py

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

        capsProblem = self.options['capsProblem']
        cart3d = capsProblem.analysis['cart3d']

        # Update input values
        if capsProblem.geometry.despmtr.wing.alphab != inputs['wing:alphab']:
            capsProblem.geometry.despmtr.wing.alphab = inputs['wing:alphab']

        # Disable design sensitivity calculations
        if cart3d.input.Design_Sensitivity != False:
            cart3d.input.Design_Sensitivity = False

        # Return the functional output (triggers autoExecution of c3d_objGrad.csh)
        outputs['CL2'] = cart3d.dynout["CL2"].value
```

```
cart3d.input.Design_Functional = {"CL2": {"function":"C_L", "power":2}}
```

OpenMDAO Component: Compute Partials

- Compute the partial $\partial C_L^2/\partial$(wing:alphar)
- Enable Design_Sensitivity

session06/2_cart3d_OpenMDAO_twist.py

```
    def compute_partials(self,inputs,partials):
        # Compute partial derivatives of the functionals

        capsProblem = self.options['capsProblem']
        cart3d = capsProblem.analysis['cart3d']

        # Update input values
        if capsProblem.geometry.despmtr.wing.alphab != inputs['wing:alphab']:
            capsProblem.geometry.despmtr.wing.alphab = inputs['wing:alphab']

        # Compute analysis and/or geometric sensitivities for the Design_Functional
        if cart3d.input.Design_Sensitivity != True:
            cart3d.input.Design_Sensitivity = True

        # Get derivatives and set partials (triggers autoExecution of c3d_objGrad.csh)
        partials['CL2', 'wing:alphab']  = cart3d.dynout["CL2"].deriv("wing:alphab")
```

```
cart3d.input.Design_Variable = {"wing:alphab":""}
```

Cart3D AIM Geometric Sensitivity

- Cart3D AIM generates all tessellation sensitivity files

```
cart3d.analysisDir
└── design
    └── geometry_CAPSmodel
        └── geometry
            └── Model__CAPSmodel__Variable__wing:alphar_1_1_
                └── Components.i.tri
```

- Cart3D AIM executes c3d_objGrad.csh "restart" via system call

OpenMDAO: Execution

session06/2_cart3d_OpenMDAO_twist.py

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

# Add the fluid analysis - all the options are added as arguments
cart3dSystem = Cart3dAnalysis(capsProblem = capsProblem)

# setup the optimization
omProblem.driver = om.ScipyOptimizeDriver()
omProblem.driver.options['optimizer'] = "L-BFGS-B"
omProblem.driver.options['tol'] = 1.e-6
omProblem.driver.options['disp'] = True
omProblem.driver.options['maxiter'] = 10

# Add the Cart3D subsystem
omProblem.model.add_subsystem('cart3dSystem', cart3dSystem)

# Add design variables to model
omProblem.model.add_design_var('cart3dSystem.wing:alphab', lower=-10.0, upper=10.0)

# Set objective function to minimize Cl**2
omProblem.model.add_objective('cart3dSystem.CL2')

# Press go
omProblem.setup()
omProblem.run_driver()
omProblem.cleanup()

# Get the optimized value
opt_alphab = omProblem.get_val("cart3dSystem.wing:alphab")
print("Optimized wing:alphab:", opt_alphab)
```

Overview

- CFD analysis with Cart3D
- Optimization with Cart3D
  - Geometric Parameter
- CFD analysis with Fun3D
- Fun3D with refine (sketch-to-solution)
- Optimization with Fun3D
  - Re-meshing
  - Mesh-morphing

Fun3D Analysis Outline

- Load geometry
- Generate mesh (the most difficult CFD input to generate)
- Create CFD AIM
- Set CFD analysis inputs
- Execute CFD
- Extract analysis outputs

session06/3_fun3d_InviscidWing.py

Fun3D AIM Analysis Inputs

- Mesh generated with AFLR AIM

session06/3_fun3d_InviscidWing.py

```
# Create Fun3D AIM
fun3d = capsProblem.analysis.create(aim  = "fun3dAIM",
                                    name = "fun3d")

# Link the aflr3 Volume_Mesh as input to fun3d
fun3d.input["Mesh"].link(aflr3.output["Volume_Mesh"])

# Set project name. Files written to analysisDir will have this name
fun3d.input.Proj_Name = "inviscidWing"

fun3d.input.Alpha = 1.0                     # AoA
fun3d.input.Mach = 0.5                      # Mach number
fun3d.input.Equation_Type = "Compressible" # Equation type
fun3d.input.Num_Iter = 5                    # Number of iterations
fun3d.input.Restart_Read = 'off'            # Do not read restart
fun3d.input.Viscous = "inviscid"            # Inviscid calculation

# Set boundary conditions via capsGroup
inviscidBC = {"bcType" : "Inviscid"}
fun3d.input.Boundary_Condition = {"Wing"    : inviscidBC,
                                  "Farfield":"farfield"}
```

Fun3D NML Analysis Inputs

- fun3d.nml is very large (and changes with Fun3D versions)
- Not all inputs implemented in AIM
- f90nml used to write directly to fun3d.nml
  - NOTE: Circumvents CLEAN/DIRTY process
  - Always use AIM inputs when available

session06/3_fun3d_InviscidWing.py

```
# Use python to add inputs to fun3d.nml file
fun3d.input.Use_Python_NML = True

# Write boundary output variables to the fun3d.nml file directly
fun3dnml = f90nml.Namelist()
fun3dnml['boundary_output_variables'] = f90nml.Namelist()
fun3dnml['boundary_output_variables']['mach'] = True
fun3dnml['boundary_output_variables']['cp'] = True
fun3dnml['boundary_output_variables']['average_velocity'] = True

fun3dnml.write(os.path.join(fun3d.analysisDir,"fun3d.nml"), force=True)
```

Fun3D Execution

- Execute Fun3D using system call

session06/3_fun3d_InviscidWing.py

```
# Run AIM pre-analysis
fun3d.preAnalysis()

print ("\n==> Running FUN3D......")
# Run fun3d via system call
fun3d.system("nodet_mpi --animation_freq -1 --write_aero_loads_to_file > Info.out")

# Run AIM post-analysis
fun3d.postAnalysis()
```

Fun3D Extracting Analysis Output

- Retrieve forces and moments

session06/3_fun3d_InviscidWing.py

```
# Get force results
print ("\n==> Total Forces and Moments")
# Get Lift and Drag coefficients
print ("--> Cl = ", fun3d.output.CLtot,
           "Cd = ", fun3d.output.CDtot)

# Get Cmx, Cmy, and Cmz coefficients
print ("--> Cmx = ", fun3d.output.CMXtot,
           "Cmy = ", fun3d.output.CMYtot,
           "Cmz = ", fun3d.output.CMZtot)

# Get Cx, Cy, Cz coefficients
print ("--> Cx = ", fun3d.output.CXtot,
           "Cy = ", fun3d.output.CYtot,
           "Cz = ", fun3d.output.CZtot)
```

Overview

- CFD analysis with Cart3D
- Optimization with Cart3D
  - Geometric Parameter
- CFD analysis with Fun3D
- Fun3D with refine (sketch-to-solution)
- Optimization with Fun3D
  - Re-meshing
  - Mesh-morphing

Creating refine AIM

session06/4_fun3d_refine_InviscidWing.py

```
# Create refine AIM
refine = capsProblem.analysis.create(aim = "refineAIM",
                                     name= "refine")

# Limit the number of passes to speed up this example (generally this should be the default)
refine.input.Passes = 2

# Define executeable string
refine.input.ref="mpiexec -n 4 refmpifull";

# Link the aflr3 mesh as the initial mesh for adaptation
refine.input["Mesh"].link(aflr3.output["Volume_Mesh"]);

#-----------------------------------------------------------------------------#

# Create Fun3D AIM
fun3d = capsProblem.analysis.create(aim  = "fun3dAIM",
                                    name = "fun3d")

# Link the refine Mesh as input to fun3d
fun3d.input["Mesh"].link(refine.output["Mesh"])
```

Fun3D AIM Scalar and Restart Files

session06/4_fun3d_refine_InviscidWing.py

```
# additional FUN3D inputs to generate solb file with primitive variables for refine
fun3dnml['volume_output_variables'] = f90nml.Namelist()
fun3dnml['volume_output_variables']['mach'] = False
fun3dnml['volume_output_variables']['x']    = False
fun3dnml['volume_output_variables']['y']    = False
fun3dnml['volume_output_variables']['z']    = False
fun3dnml['volume_output_variables']['primitive_variables'] = True
fun3dnml['volume_output_variables']['export_to'] = 'solb'

fun3dnml['global'] = f90nml.Namelist()
fun3dnml['global']['volume_animation_freq'] = -1

fun3dnml.write(os.path.join(fun3d.analysisDir,"fun3d.nml"), force=True)

# Specify the file name for the primitive variable field use to compute multiscale metric
refine.input.ScalarFieldFile = os.path.join(fun3d.analysisDir, fun3d.input.Proj_Name + "_volume.solb");

# Indicate that solb file contains Fun3D primitive variables
# and generate a 'restart' file for Fun3d
refine.input.Fun3D = True
```

Fun3D and refine Adaptation Loop

session06/4_fun3d_refine_InviscidWing.py

```
# Specify number of adaptation iterations for each target complexity
nAdapt     = [   2,     2]
tar_Complex= [8000, 16000]
for ii,iComplex in enumerate(tar_Complex):
    for iadapt in range(0,nAdapt[ii]):
        # Set the current mesh size
        refine.input["Complexity"].value = iComplex

        if iadapt > 0 or iComplex > tar_Complex[0]:
            fun3dnml["flow_initialization"]=f90nml.Namelist()
            fun3dnml["flow_initialization"]['import_from'] = \
                os.path.join(refine.analysisDir, 'refine_out-restart.solb')
            fun3dnml.write(os.path.join(fun3d.analysisDir,"fun3d.nml"), force=True)

        ####### Run fun3d ####################
        fun3d.preAnalysis()
        print("\n==> Running FUN3D, complexity = %6d, iadapt = %2d....." % (iComplex,iadapt))
        fun3d.system("mpiexec -np 4 nodet_mpi --write_aero_loads_to_file > Info.txt")
        fun3d.postAnalysis()
        ######################################

        # Get force results
        print ("\n==> Total Forces and Moments")
        # Get Lift and Drag coefficients
        print ("--> Cl = ", fun3d.output.CLtot,
                   "Cd = ", fun3d.output.CDtot)

        # Unlink the refine mesh input
```

Overview

- CFD analysis with Cart3D
- Optimization with Cart3D
  - Geometric Parameter
- CFD analysis with Fun3D
- Fun3D with refine (sketch-to-solution)
- Optimization with Fun3D
  - Re-meshing
  - Mesh-morphing

Fun3D Optimization with Constraint

- Analysis and Geometric sensitivities
- Gradient based optimization

$$\min_{\alpha,\text{wing:aspect}} \quad C_D^2 \tag{2}$$

$$s.t.\ C_L \equiv 0.1 \tag{3}$$

- Use Fun3D to compute C_D^2, C_L and $\partial/\partial\alpha$, $\partial/\partial\text{aspect}$
  - CAPS computes geometric sensitivities
- New mesh generated with each shape change
  - Support large deformation and topological changes

Fun3D AIM Design Analysis Inputs

- Design analysis inputs
  - Design_Functional: Functionals in Fun3D rubber.data
  - Design_Variable: Variables for partial derivatives

session06/5_fun3d_OpenMDAO_remesh.py

```
# Setup Cd**2 as the objective functional to minimize
# and Cl for the constraint
fun3d.input.Design_Functional = {"Cd2": {"function":"Cd", "power":2},
                                 "Cl" : {"function":"Cl"}}

# Declare design variables
fun3d.input.Design_Variable = {"Alpha":"",
                               "wing:aspect":""}
```

```
fun3d.input.Alpha = 1.0                    # AoA
```

ESP/wingPmtrs.udc

```
DESPMTR   wing:aspect      9.00    # aspect ratio
```

OpenMDAO Component: Setup

- Define output functional and partial derivatives for OpenMDAO

session06/5_fun3d_OpenMDAO_remesh.py

```
    def setup(self):
        # Assign the actual values to the variables.
        capsProblem = self.options['capsProblem']
        fun3d = capsProblem.analysis['fun3d']

        # attach free design parameters to OpenMDAO fluid object
        self.add_input('Alpha' , val=fun3d.input.Alpha)
        self.add_input('wing:aspect', val=capsProblem.geometry.despmtr.wing:aspect)

        # Add outputs for both objective and constraints
        self.add_output('Cd2')
        self.add_output('Cl')

        # Add partials
        self.declare_partials('Cd2','Alpha')
        self.declare_partials('Cd2','wing:aspect')

        self.declare_partials('Cl','Alpha')
        self.declare_partials('Cl','wing:aspect')
```

OpenMDAO Component: Compute

- Disable Design_Sensitvity, run nodet_mpi

session06/5_fun3d_OpenMDAO_remesh.py

```
    def compute(self, inputs, outputs):
        # Compute functionals
        cores = self.options['cores']
        capsProblem = self.options['capsProblem']
        fun3d = capsProblem.analysis['fun3d']

        # Update input values
        fun3d.input.Alpha = inputs['Alpha']
        capsProblem.geometry.despmtr.wing.aspect = inputs['wing:aspect']

        # Disable design sensitivity calculations
        fun3d.input.Design_Sensitivity = False

        # Run forward analysis
        fun3d.preAnalysis()
        fun3d.system(f"mpirun -np {cores} nodet_mpi --design_run | tee nodet.out", 'Flow')
        fun3d.postAnalysis()

        # Grab functional and attach as an output
        outputs['Cd2'] = fun3d.dynout.Cd2
        outputs['Cl']  = fun3d.dynout.Cl
```

```
fun3d.input.Design_Functional = {"Cd2": {"function":"Cd", "power":2},
                                 "Cl" : {"function":"Cl"}}
```

OpenMDAO Component: Compute Partials

- Enable Design_Sensitivity, run dual_mpi

session06/5_fun3d_OpenMDAO_remesh.py

```
    def compute_partials(self,inputs,partials):
        # The actual value of the partial derivative is assigned here.
        cores = self.options['cores']
        capsProblem = self.options['capsProblem']
        fun3d = capsProblem.analysis['fun3d']

        # Request analysis and/or geometric design sensitivities for the Design_Functional
        fun3d.input.Design_Sensitivity = True

        # Run adjoint
        fun3d.preAnalysis()
        fun3d.system(f"mpirun -np {cores} dual_mpi --getgrad --outer_loop_krylov | tee dual.out",'Adjoint')
        fun3d.postAnalysis()

        # Get derivatives and set partials
        partials['Cd2', 'Alpha'      ] = fun3d.dynout["Cd2"].deriv("Alpha")
        partials['Cd2', 'wing:aspect'] = fun3d.dynout["Cd2"].deriv("wing:aspect")

        partials['Const', 'Alpha'      ] = fun3d.dynout["Cl"].deriv("Alpha")
        partials['Const', 'wing:aspect'] = fun3d.dynout["Cl"].deriv("wing:aspect")
```

```
fun3d.input.Design_Variable = {"Alpha":"",
                               "wing:aspect":""}
```

OpenMDAO: Execution

session06/5_fun3d_OpenMDAO_remesh.py

```
# setup the optimization
omProblem.driver = om.ScipyOptimizeDriver()
omProblem.driver.options['optimizer'] = 'SLSQP'
#omProblem.driver.options['optimizer'] = "L-BFGS-B"
omProblem.driver.options['tol'] = 1.e-7
omProblem.driver.options['disp'] = True
omProblem.driver.options['maxiter'] = 10

# add design variables to model
aspect_limits = transport.despmtr["wing:aspect"].limits

omProblem.model.add_design_var('fun3dSystem.Alpha' , lower=-25.0, upper=25.0)
omProblem.model.add_design_var('fun3dSystem.wing:aspect', lower=aspect_limits[0], upper=aspect_limits[1])

# minimize Cd**2 subject to Cl == 0.1
omProblem.model.add_objective('fun3dSystem.Cd2')
omProblem.model.add_constraint('fun3dSystem.Cl', equals = 0.1)

# Press go
omProblem.setup()
omProblem.run_driver()
omProblem.cleanup()

# Get the optimized value
opt_Cd2    = omProblem.get_val("fun3dSystem.Cd2")
opt_Cl     = omProblem.get_val("fun3dSystem.Cl")
opt_Alpha  = omProblem.get_val("fun3dSystem.Alpha")
opt_aspect = omProblem.get_val("fun3dSystem.wing:aspect")
```

Fun3D Optimization with Mesh Morphing

- Mesh morphing might have smoother sensitivities
- Limited to smaller deformations
- No geometric topological changes

session06/6_fun3d_OpenMDAO_morph.py

```
# Enable Mesh_Morph to save off a reference mesh
fun3d.input.Mesh_Morph = True
```

- Compute with additional arguments
- Unlink mesh to dissable re-generation

```
        # Run forward analysis
        fun3d.preAnalysis()
        fun3d.system(f"mpirun -np {cores} nodet_mpi --design_run " +
                     f"--read_surface_from_file --write_mesh {fun3d.input.Proj_Name} | tee nodet.out", 'Flow')
        fun3d.postAnalysis()

        # Grab functional and attach as an output
        outputs['Cd2'] = fun3d.dynout.Cd2
        outputs['Cl']  = fun3d.dynout.Cl

        # Unlink the mesh to trigger morphing
        fun3d.input["Mesh"].unlink()
```

Suggested Exercises

- Use Cart3D to optimize analysis input α