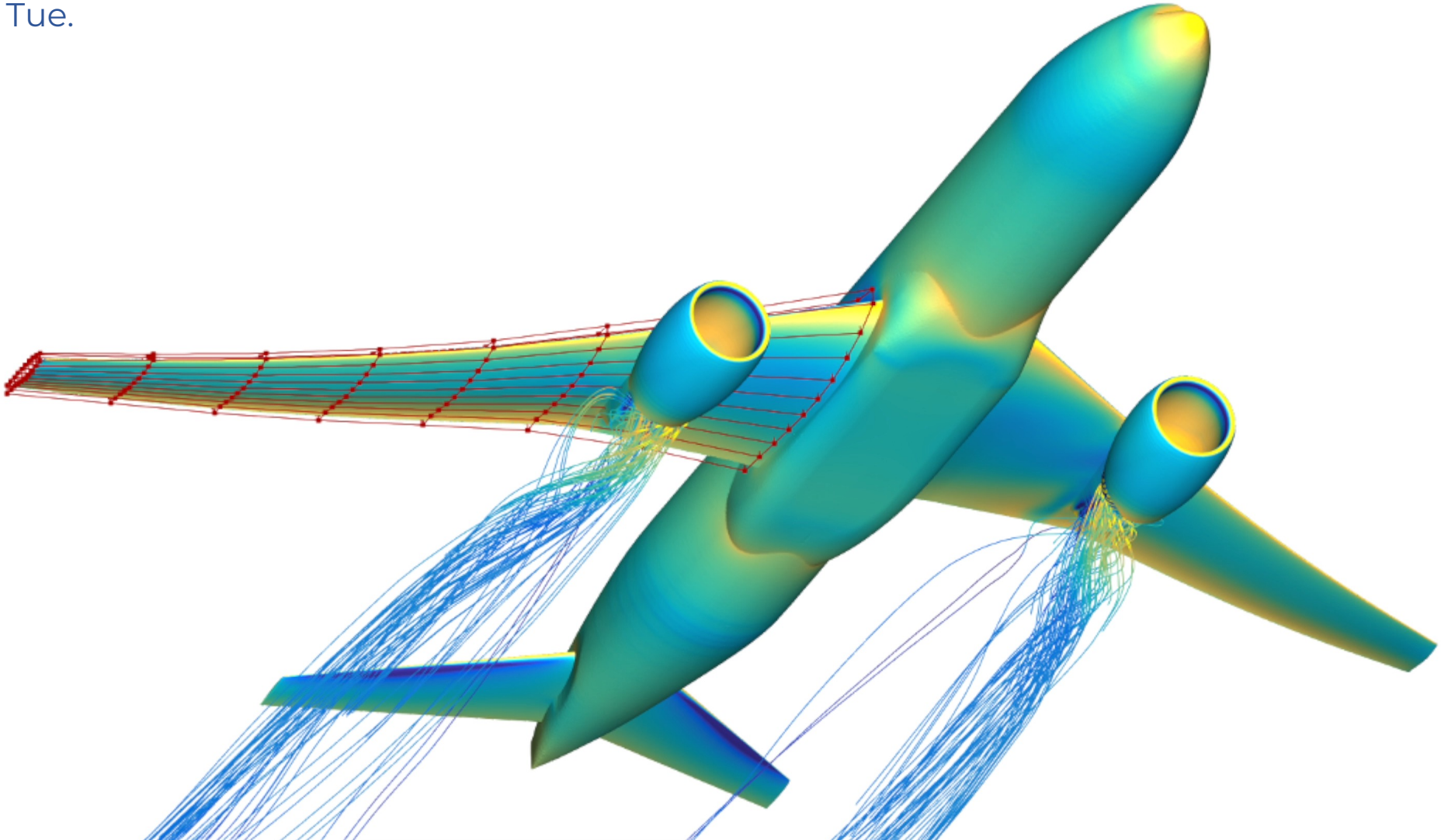
DAFoam: A Discrete-Adjoint for OpenFOAM to Enable Gradient-Based Multidisciplinary Design Optimization

Bernardo Pacini, Ping He, Joaquim R. R. A. Martins

University of Michigan + Iowa State University

July 11th, 2023 Tue.



MDO Lab

Getting Started

Download the Docker Image

```
docker pull dafoam/opt-packages:v3.0.6
```

Git Clone the Run Files

```
git clone git@github.com:bernardopacini/OpenFOAMWorkshop2023-DAFoamTutorial.git
```

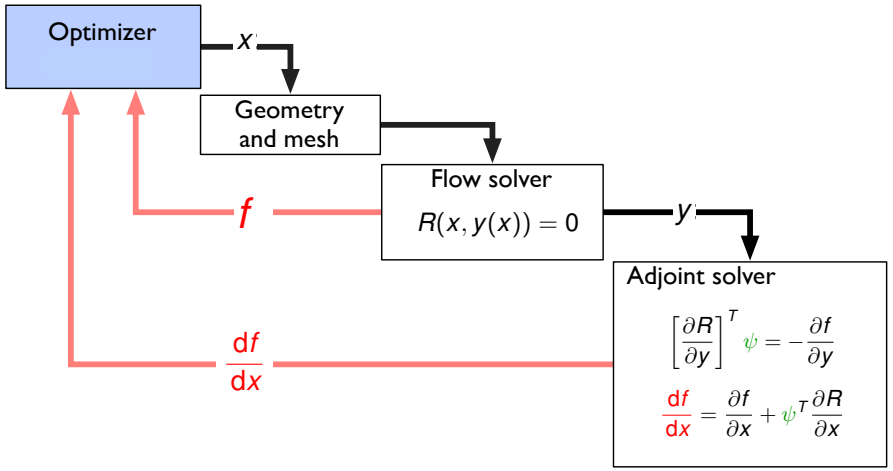
OR

Download the Run Files

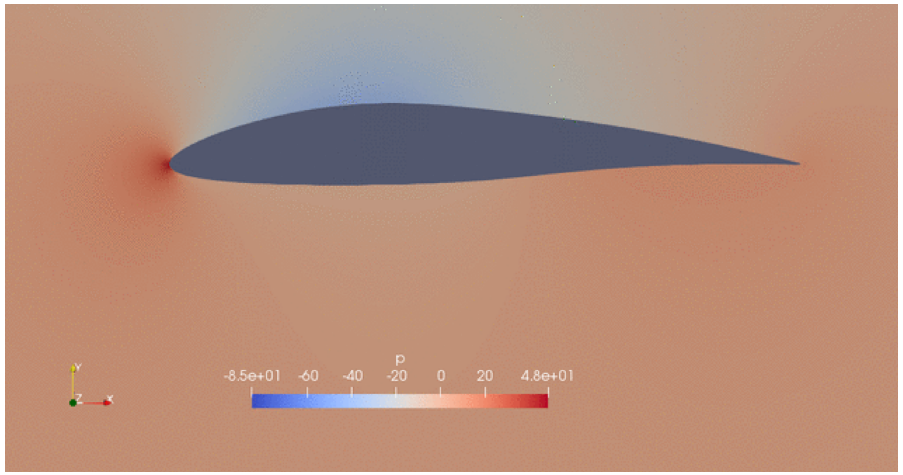
https://github.com/bernardopacini/OpenFOAMWorkshop2023-DAFoamTutorial

The Agenda

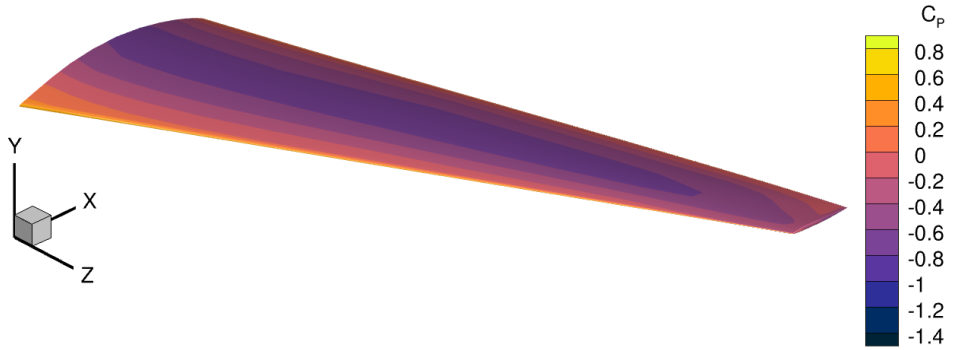
Theory



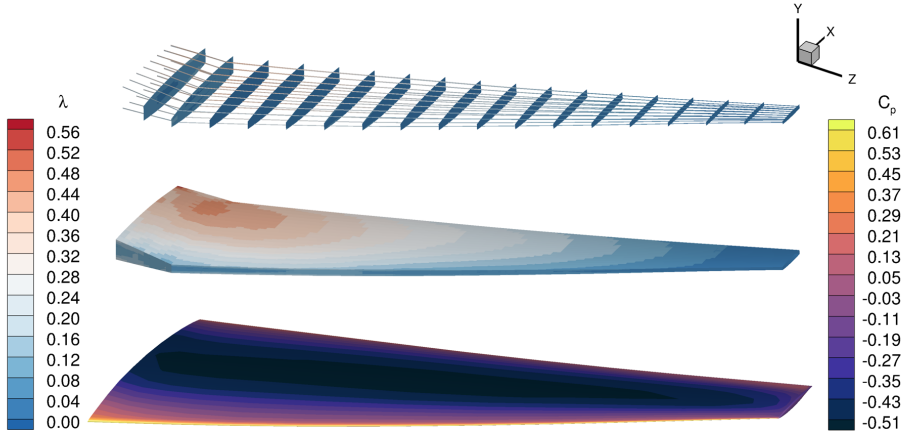
Airfoil Optimization



Wing Aerodynamic Optimization

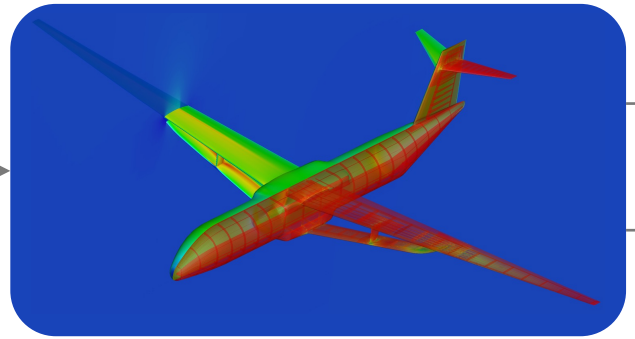


Wing Aerostructural Optimization



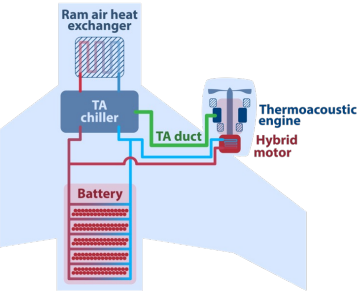
What is multidisciplinary design optimization and what is it being used for?

In the MDO Lab:

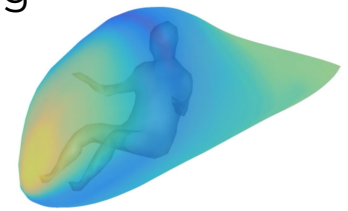


minimize	f(x)	objective
with respect to	x	design variables
subject to	c(x) ≤ 0	constraints

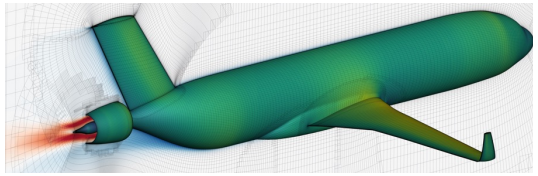
- Electro-propulsive design optimization



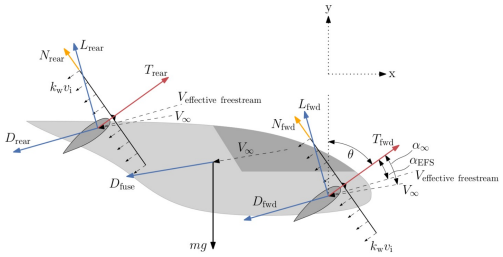
- Aerodynamic optimization with packaging



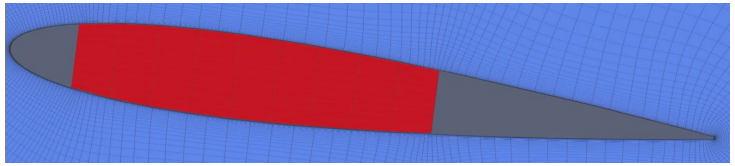
- Aero-propulsive design optimization



- Trajectory optimization



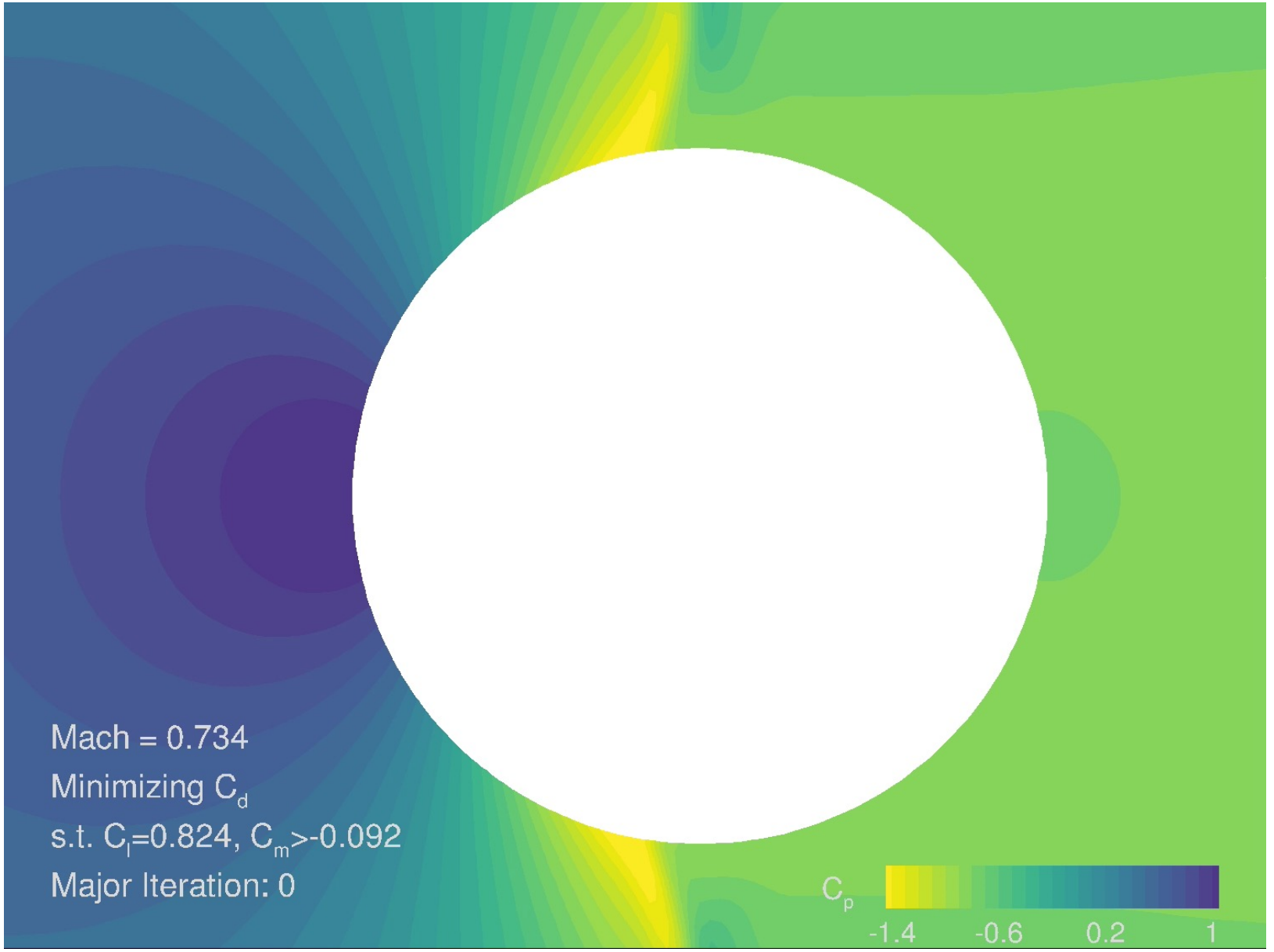
- Aero-thermal shape optimization



Slide adapted from: *Martins et al, Exploiting Aircraft Electrification via Multidisciplinary Design Optimization*

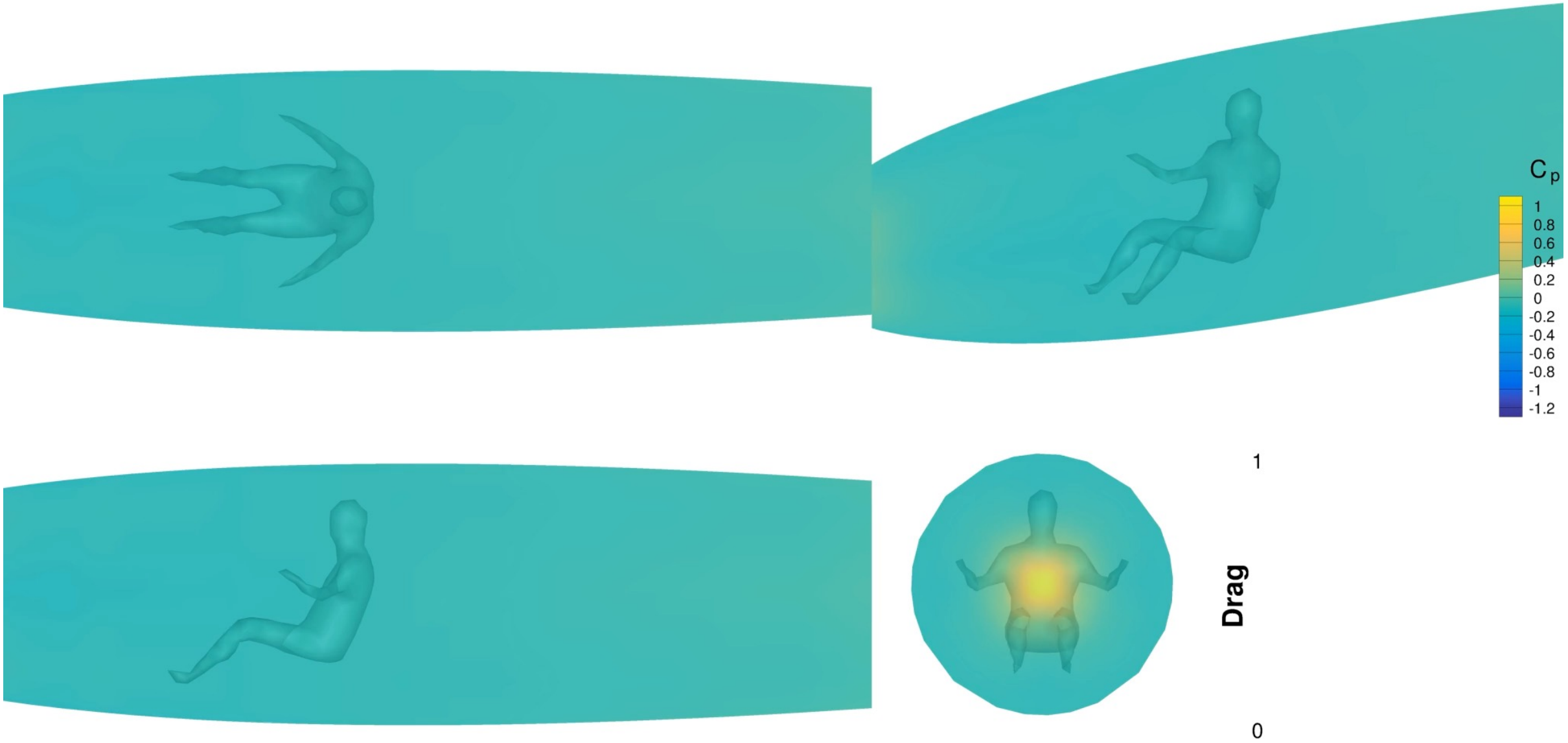
Aerodynamic modeling is central to design optimization

Circle to Airfoil Optimization



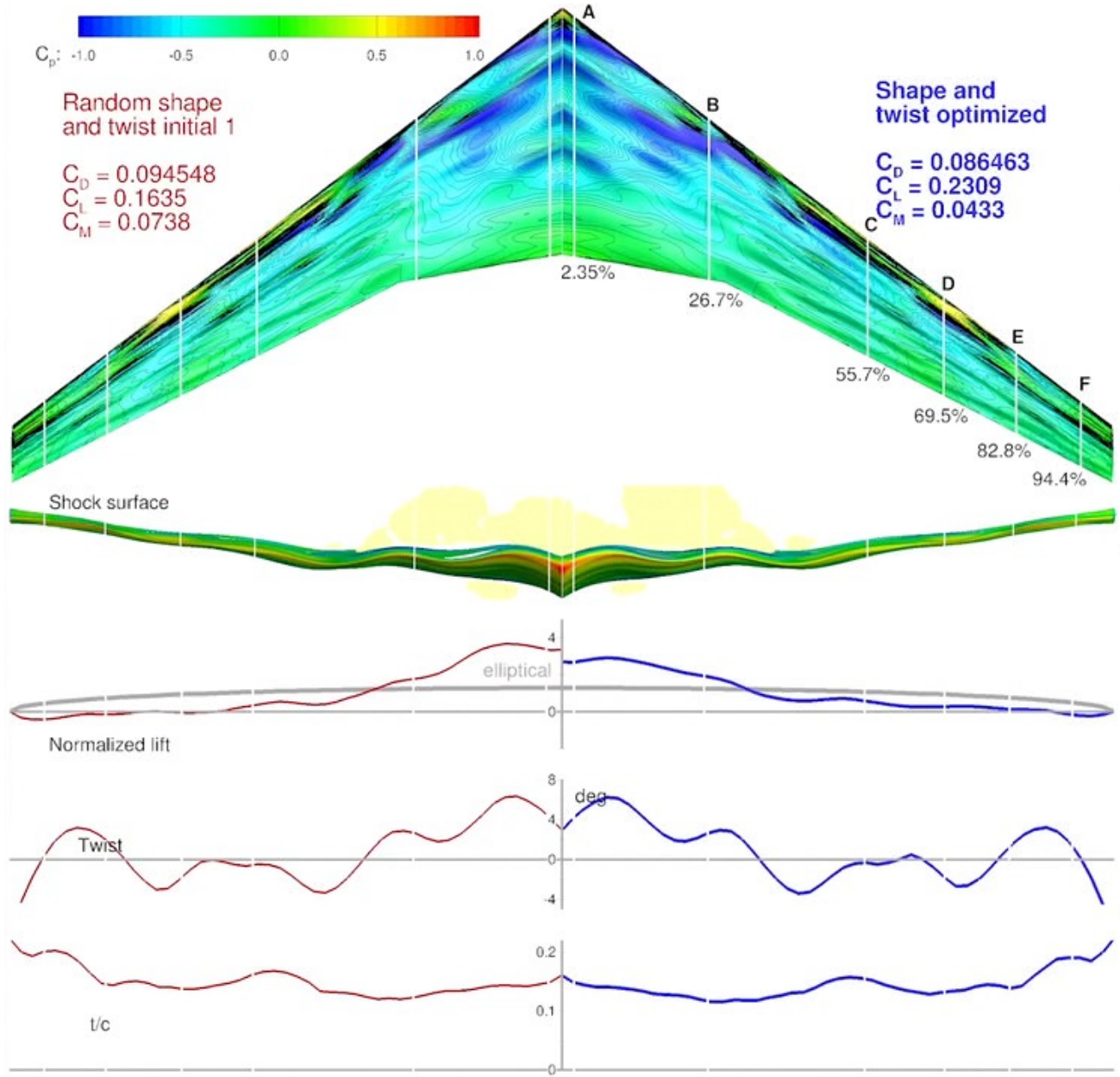
He, Li, Mader, Yildirim, Martins. Robust aerodynamic shape optimization— from a circle to an airfoil. *Aerospace Science and Technology*, 2019

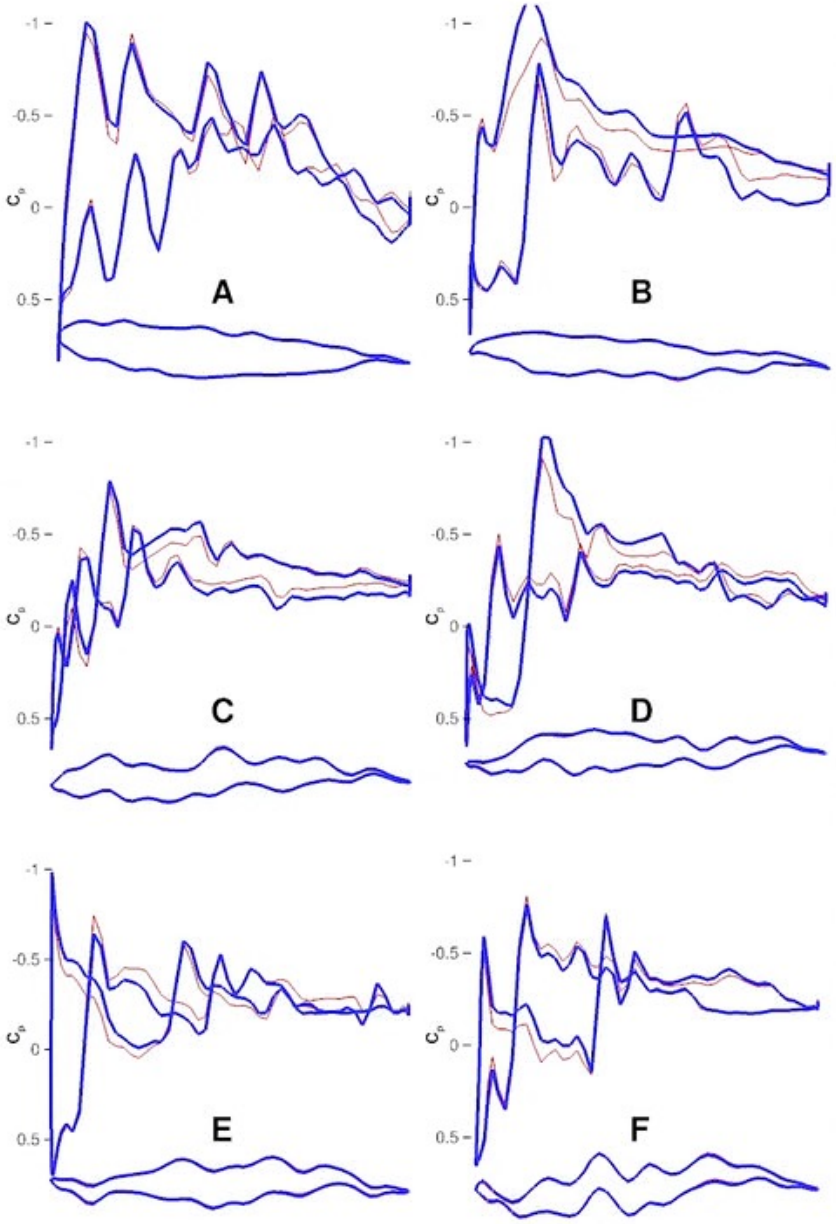
Vehicle Shape Optimization



Brelje, Anibal, Yildirim, Mader, Martins. Flexible formulation of spatial integration constraints in aerodynamic shape optimization. *AIAA Journal, 2020*.

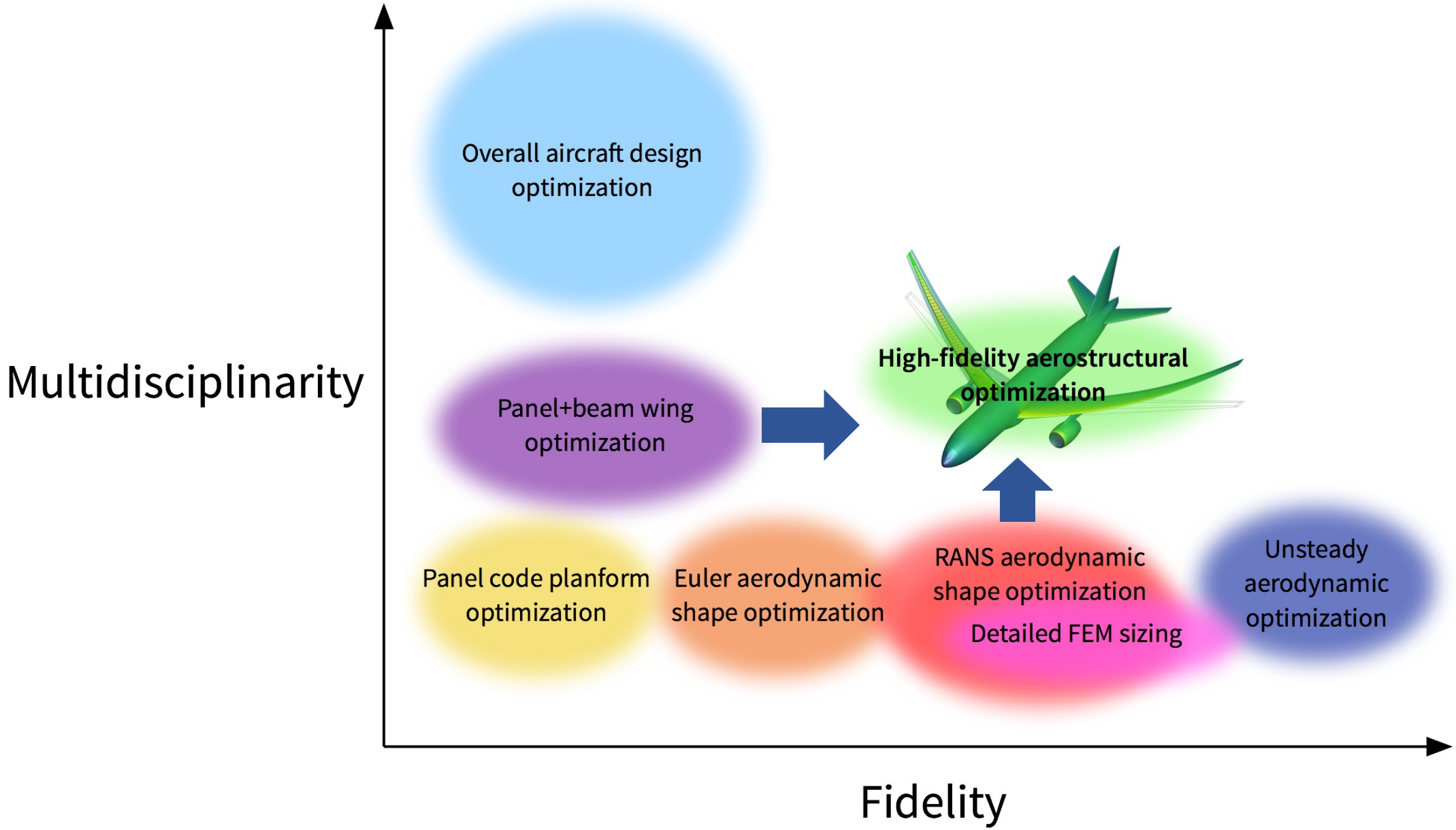
Wing Optimization





Lyu, Kenway, and Martins. Aerodynamic shape optimization investigations of the Common Research Model wing benchmark. *AIAA Journal*, 2015.

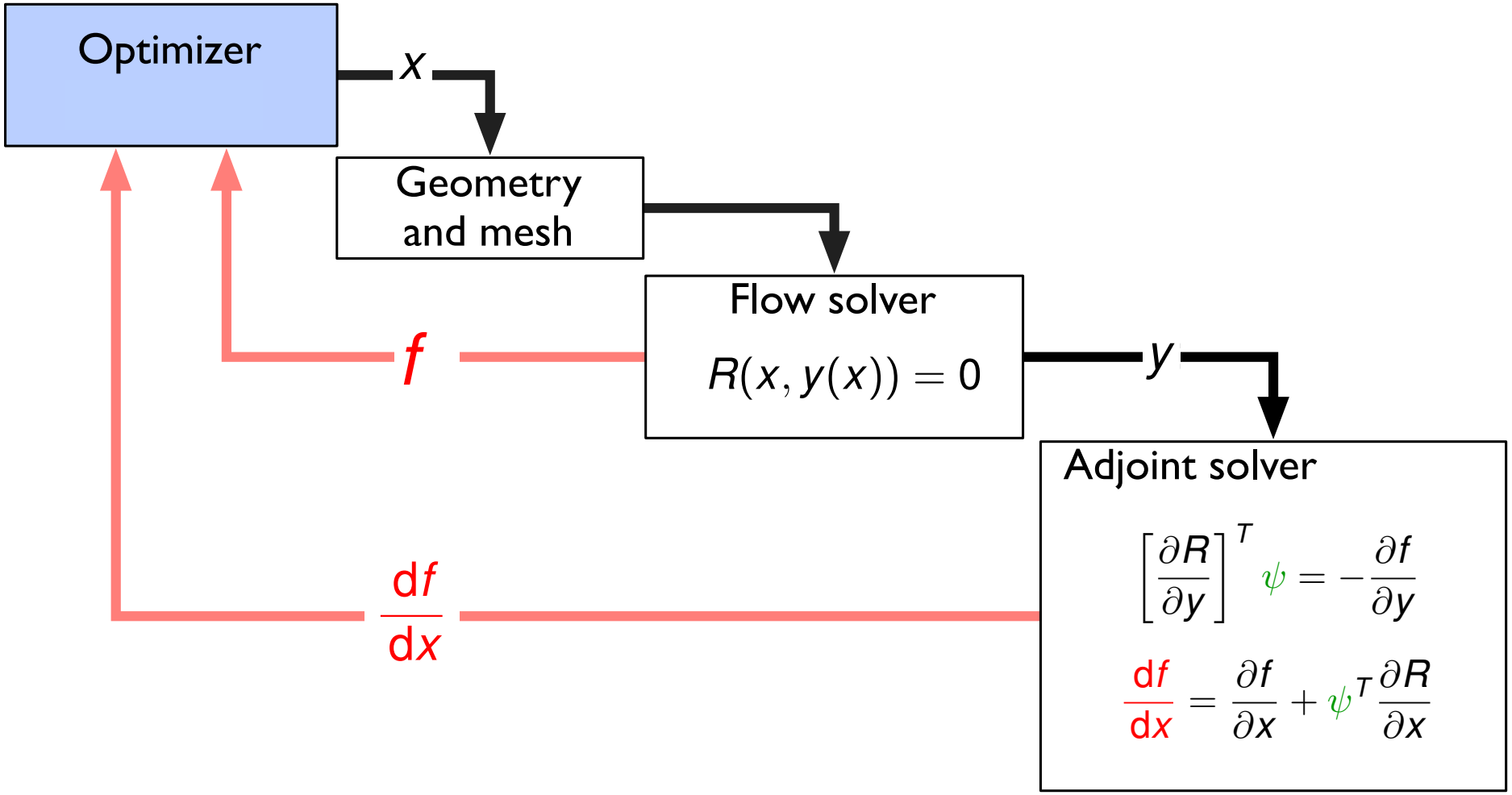
Multidisciplinary Design Optimization Requires a Balance of Fidelities



Slide adapted from: *Martins et al, Multidisciplinary Design Optimization of Engineering Systems*

How does aerodynamic shape optimization work?

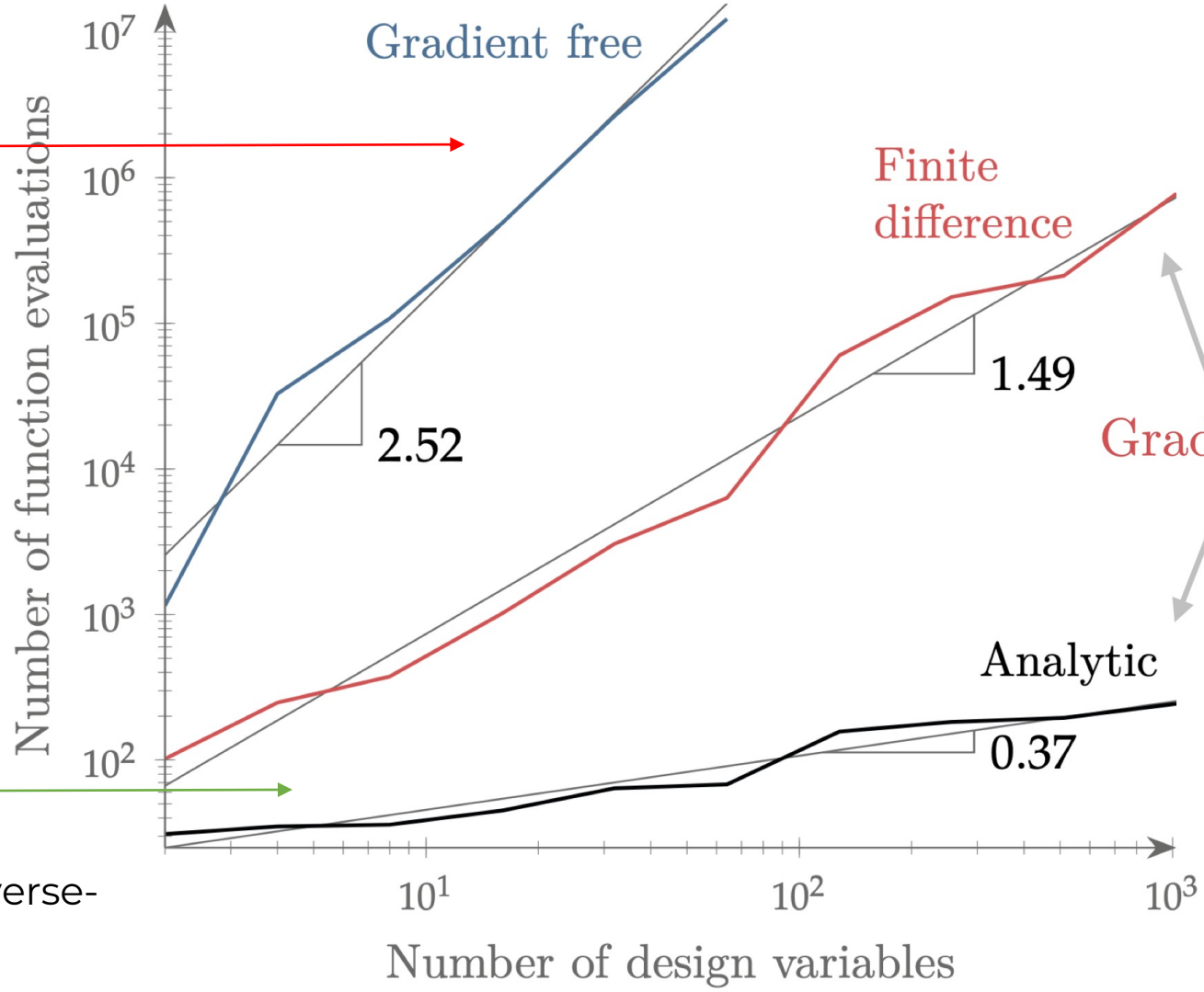
Optimization Subsystems



Slide adapted from: *Martins et al, Multidisciplinary Design Optimization of Engineering Systems*

Why do we need gradients specifically?

Gradient Free Methods



Gradient Based Methods

We use a discrete-adjoint with reverse-mode algorithmic differentiation

Martins and Ning. Engineering Design Optimization. Cambridge University Press, 2021.

How can we compute gradients?

Monolithic ***Black boxes*** ***input and outputs***	Finite-differences	$\frac{df}{dx_j} = \frac{f(x_j + h) - f(x)}{h} + \mathcal{O}(h)$	Inaccurate; requires step-size tuning
	Complex-step	$\frac{df}{dx_j} = \frac{\text{Im}\left[f(x_j + ih)\right]}{h} + \mathcal{O}(h^2)$	Accurate, requires source-code transformation
Analytic ***Governing eqns*** ***state variables***	Direct Adjoint	$\frac{df}{dx} = \frac{\partial f}{\partial x} - \underbrace{\frac{\partial f}{\partial y}\overbrace{\left[\frac{\partial R}{\partial y}\right]^{-1}}^{-\,dy/dx}\frac{\partial R}{\partial x}}_{\psi}$	Does not require re-converging solver
Algorithmic differentiation ***Lines of code*** ***code variables***	Forward Reverse	$\begin{bmatrix} 1 & 0 & \dots & 0 \\ -\frac{\partial T_2}{\partial t_1} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ -\frac{\partial T_n}{\partial t_1} & \dots & -\frac{\partial T_n}{\partial t_{n-1}} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \dots & 0 \\ \frac{dt_2}{dt_1} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \frac{dt_n}{dt_1} & \dots & \frac{dt_n}{dt_{n-1}} & 1 \end{bmatrix} = \boldsymbol{I} = \begin{bmatrix} 1 & -\frac{\partial T_2}{\partial t_1} & \dots & -\frac{\partial T_n}{\partial t_1} \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\frac{\partial T_n}{\partial t_{n-1}} \\ 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{dt_2}{dt_1} & \dots & \frac{dt_n}{dt_1} \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & 1 & \frac{dt_n}{dt_{n-1}} \\ 0 & \dots & 0 & 1 \end{bmatrix}$	Accurate, requires source-code transformation Forward-mode scales with number of model inputs; reverse-mode scales with number of model outputs

Martins and Hwang, *AIAA Journal*, 2013 | Martins et al., *ACM TOMS*, 2003

The Discrete Adjoint

$f = f(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x}))$: Function of Interest

$\mathcal{R}(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x})) = 0$: Solution Residual

$$\underbrace{\frac{df}{d\boldsymbol{x}}}_{n_f \text{ x } n_x} = \underbrace{\frac{\partial f}{\partial \boldsymbol{x}}}_{n_f \text{ x } n_x} + \underbrace{\frac{\partial f}{\partial \boldsymbol{u}}}_{n_f \text{ x } n_u} \underbrace{\frac{d\boldsymbol{u}}{d\boldsymbol{x}}}_{n_u \text{ x } n_x}$$

$$\underbrace{\frac{d\mathcal{R}}{d\boldsymbol{x}}}_{n_\mathcal{R} \text{ x } n_x} = \underbrace{\frac{\partial \mathcal{R}}{\partial \boldsymbol{x}}}_{n_\mathcal{R} \text{ x } n_x} + \underbrace{\frac{\partial \mathcal{R}}{\partial \boldsymbol{u}}}_{n_\mathcal{R} \text{ x } n_u} \underbrace{\frac{d\boldsymbol{u}}{d\boldsymbol{x}}}_{n_u \text{ x } n_x} = 0$$

$$\Rightarrow \frac{d\boldsymbol{u}}{d\boldsymbol{x}} = -\left[\frac{\partial \mathcal{R}}{\partial \boldsymbol{u}}\right]^{-1} \frac{\partial \mathcal{R}}{\partial \boldsymbol{x}}$$

$$\frac{df}{d\boldsymbol{x}} = \frac{\partial f}{\partial \boldsymbol{x}} - \underbrace{\frac{\partial f}{\partial \boldsymbol{u}} \overbrace{\left[\frac{\partial \mathcal{R}}{\partial \boldsymbol{u}}\right]^{-1}}^{\phi}}_{\psi^T} \frac{\partial \mathcal{R}}{\partial \boldsymbol{x}}$$

$$\underbrace{\frac{\partial \mathcal{R}}{\partial \boldsymbol{u}}}_{n_\mathcal{R} \text{ x } n_u} \underbrace{\phi}_{n_u \text{ x } n_x} = \underbrace{\frac{\partial \mathcal{R}}{\partial \boldsymbol{x}}}_{n_\mathcal{R} \text{ x } n_x} \text{ : Direct Method}$$

$$\underbrace{\left[\frac{\partial \mathcal{R}}{\partial \boldsymbol{u}}\right]^T}_{n_u \text{ x } n_\mathcal{R}} \underbrace{\psi}_{n_u \text{ x } 1} = \underbrace{\left[\frac{\partial f}{\partial \boldsymbol{u}}\right]^T}_{n_u \text{ x } 1} \text{ : Adjoint Method}$$

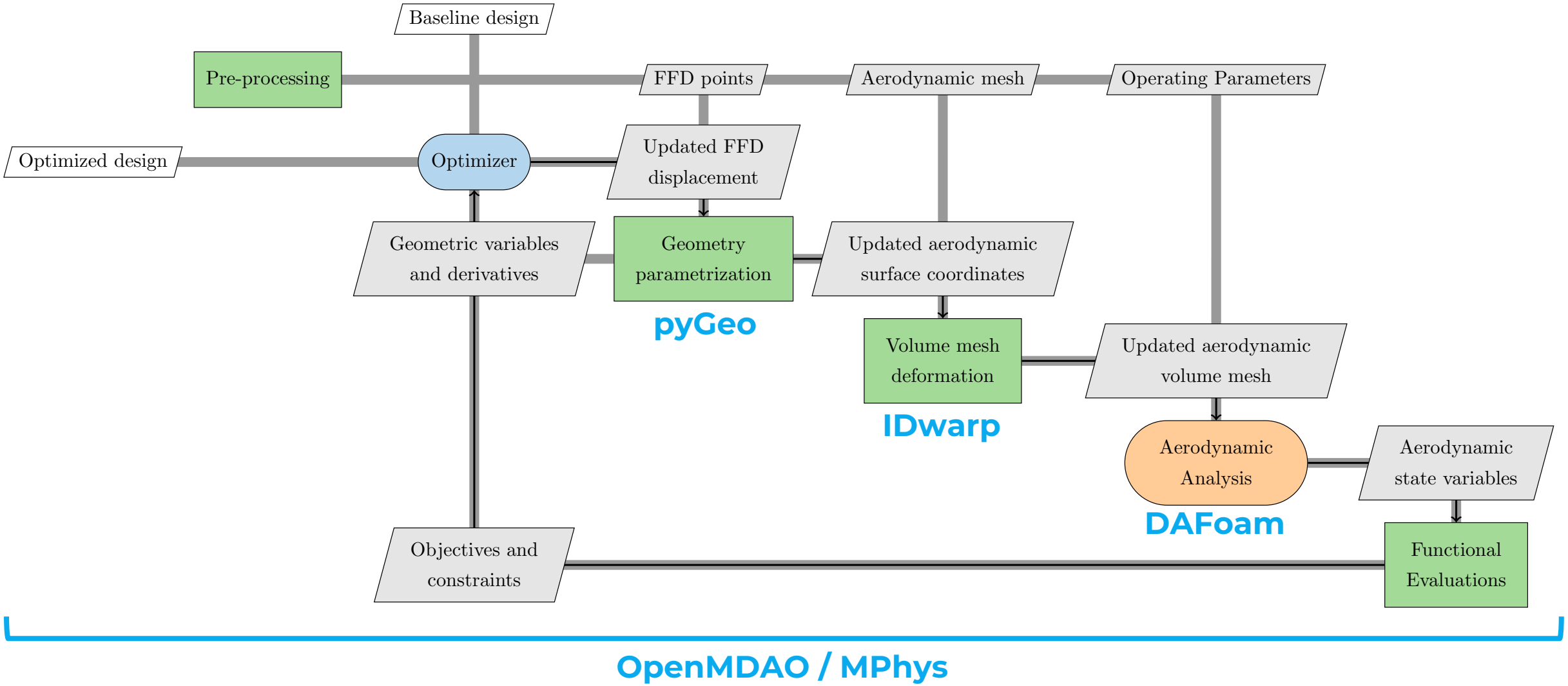
Direct method scales with n_x, the number of design variables

Adjoint method scales with n_f, the number of functionals

→ For CFD-based optimization, $n_x \gg n_f$

How is this optimization functionality implemented?

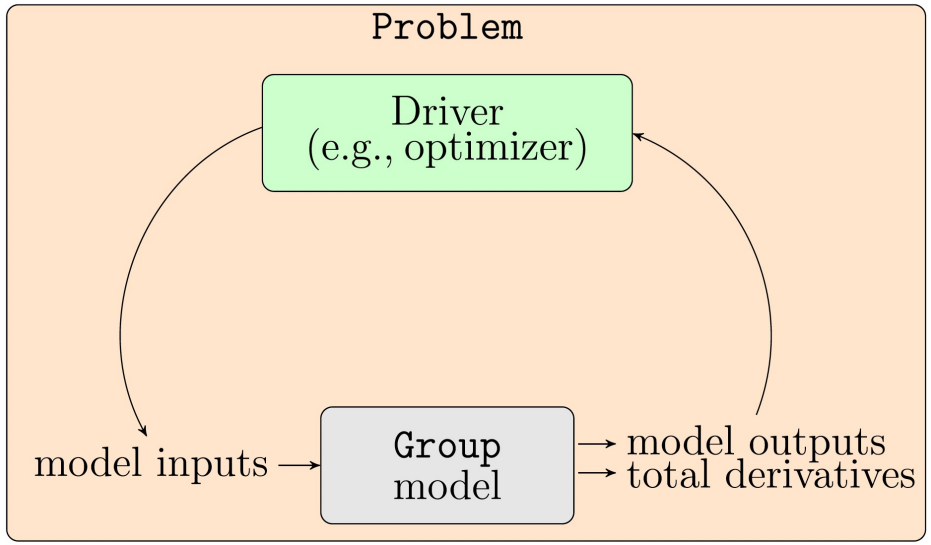
Aerodynamic Shape Optimization

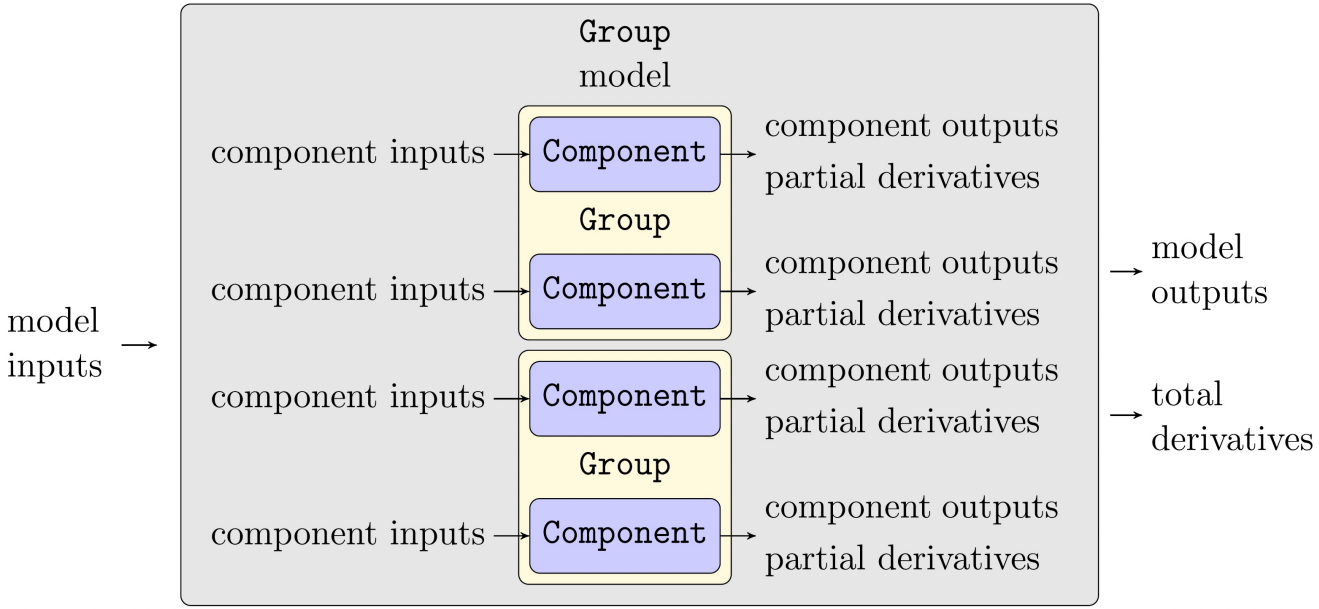


- Opensource platform for systems analysis and multidisciplinary design optimization

- Built specifically for gradient-based optimization of coupled disciplines with analytic derivatives

- Modular, making it easy to build complex coupled models with efficient numerical methods

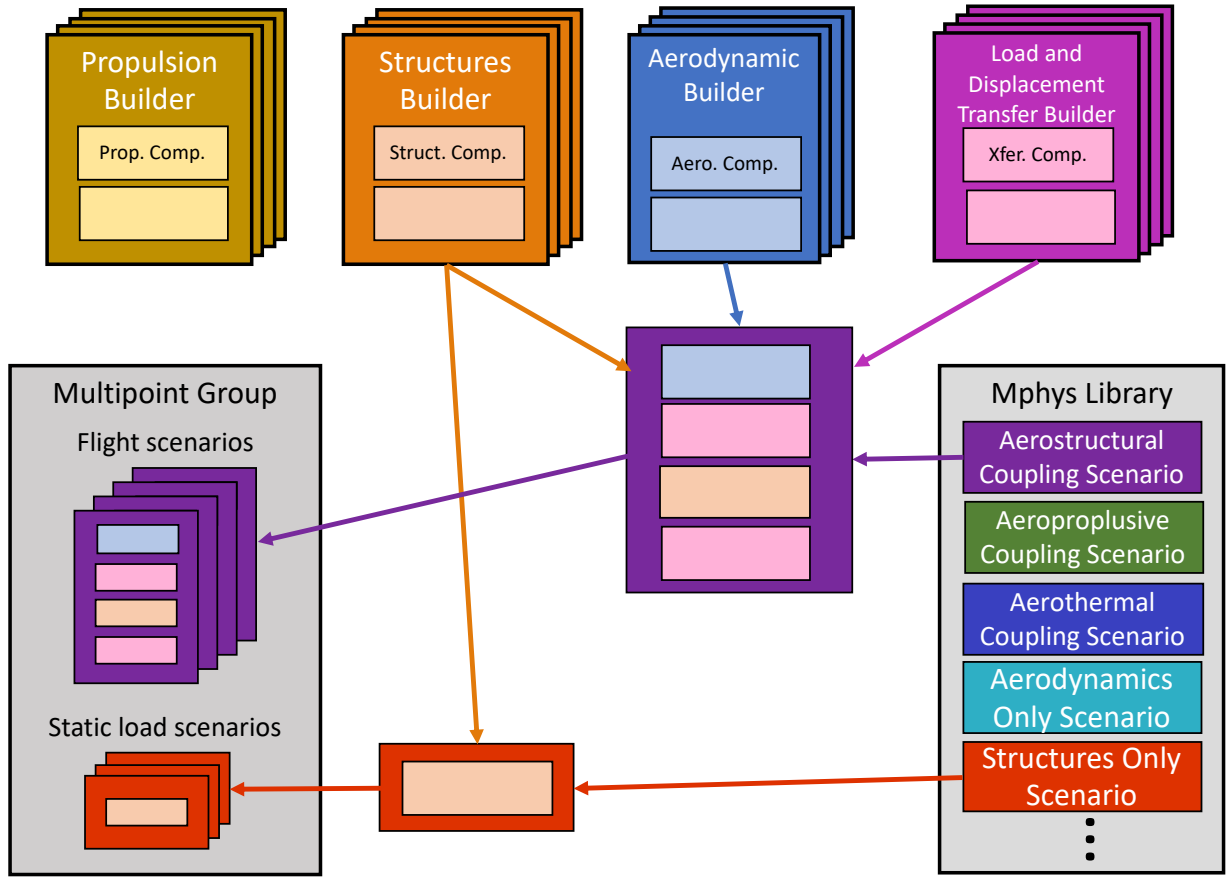




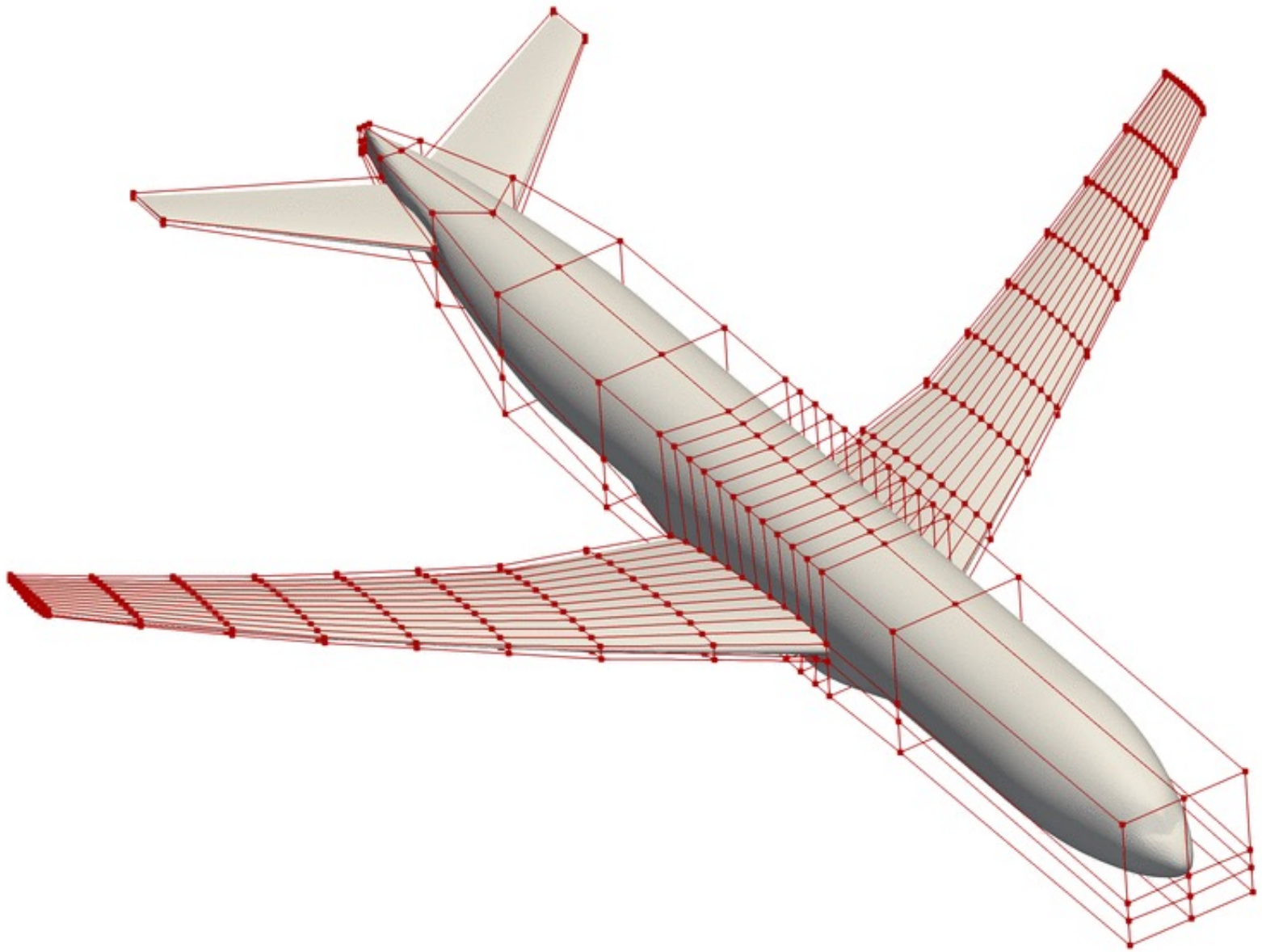
Gray, Hwang, Martins, Moore, and Naylor. OpenMDAO: An open- source framework for multidisciplinary design, analysis, and optimization. *Structural and Multidisciplinary Optimization*, 2019

MPhys

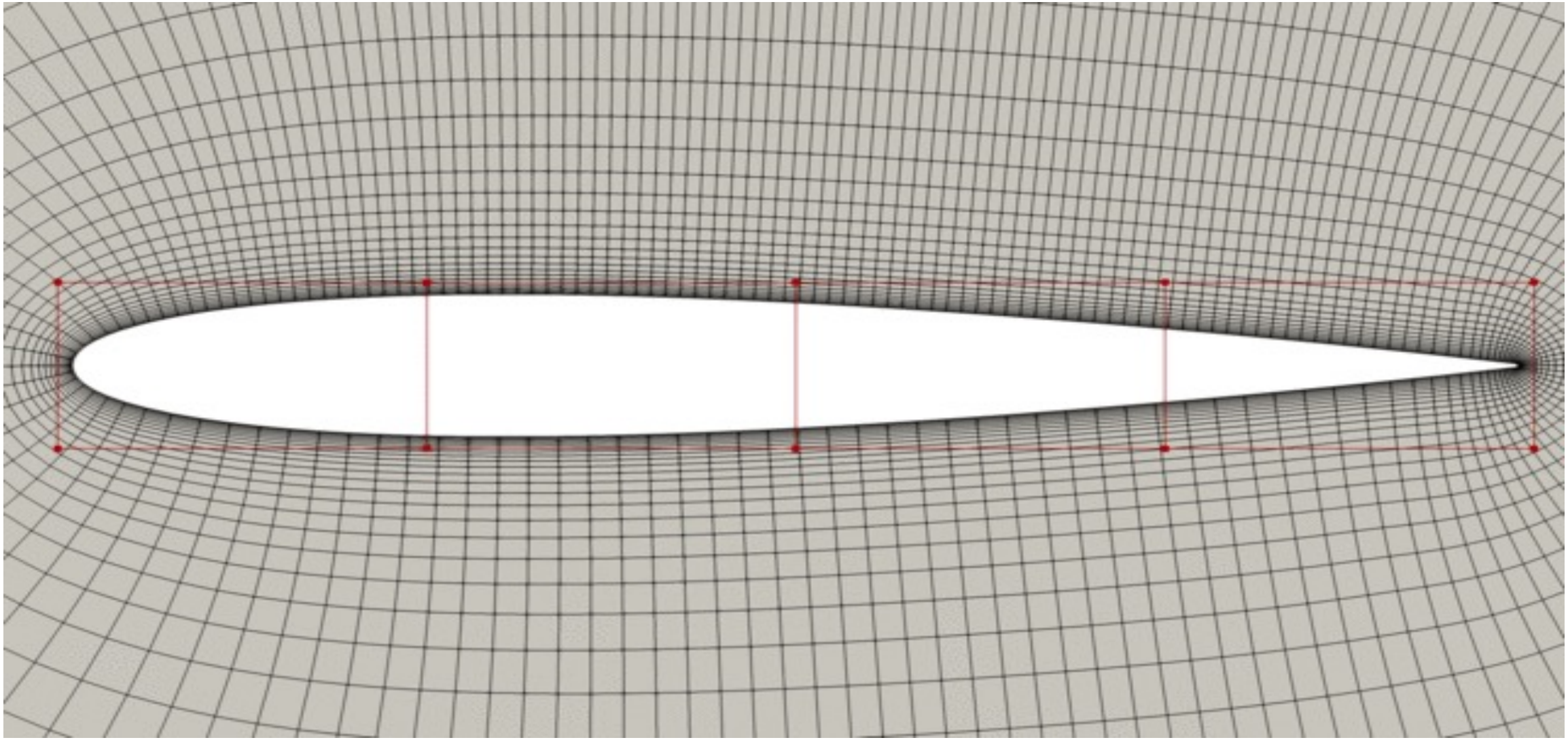
- MPhys is a library of standardized multiphysics Scenarios for OpenMDAO
- Each disciplinary module is wrapped in a builder that communicates with other components
- Model assembly process
  - Select the desired components
  - Provide the builders to the Scenario
  - Add Scenarios to optimization Groups



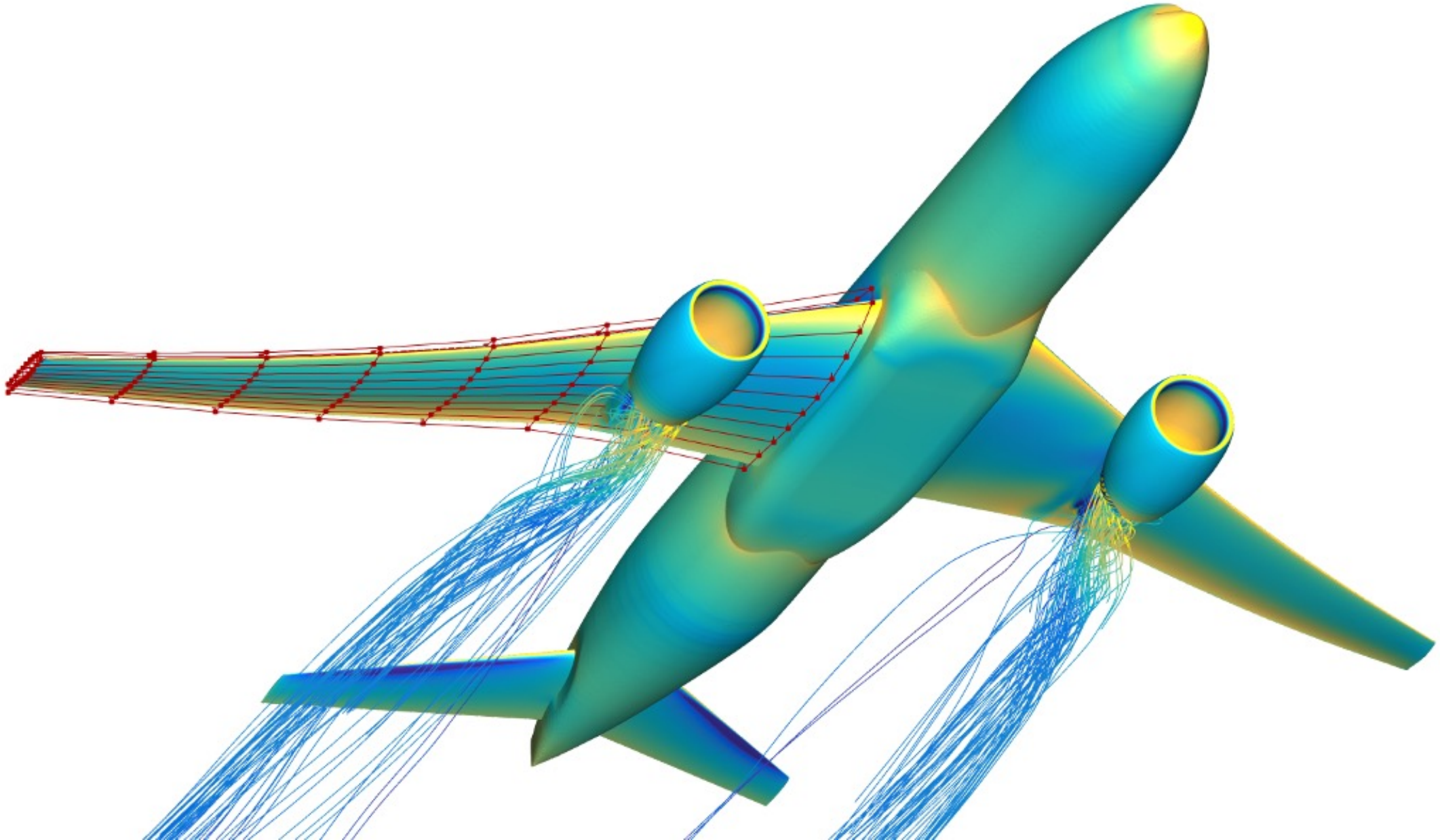
pyGeo: Geometry Parametrization with Free-Form Deformation



IDwarp: Inverse-Distance Mesh Warping

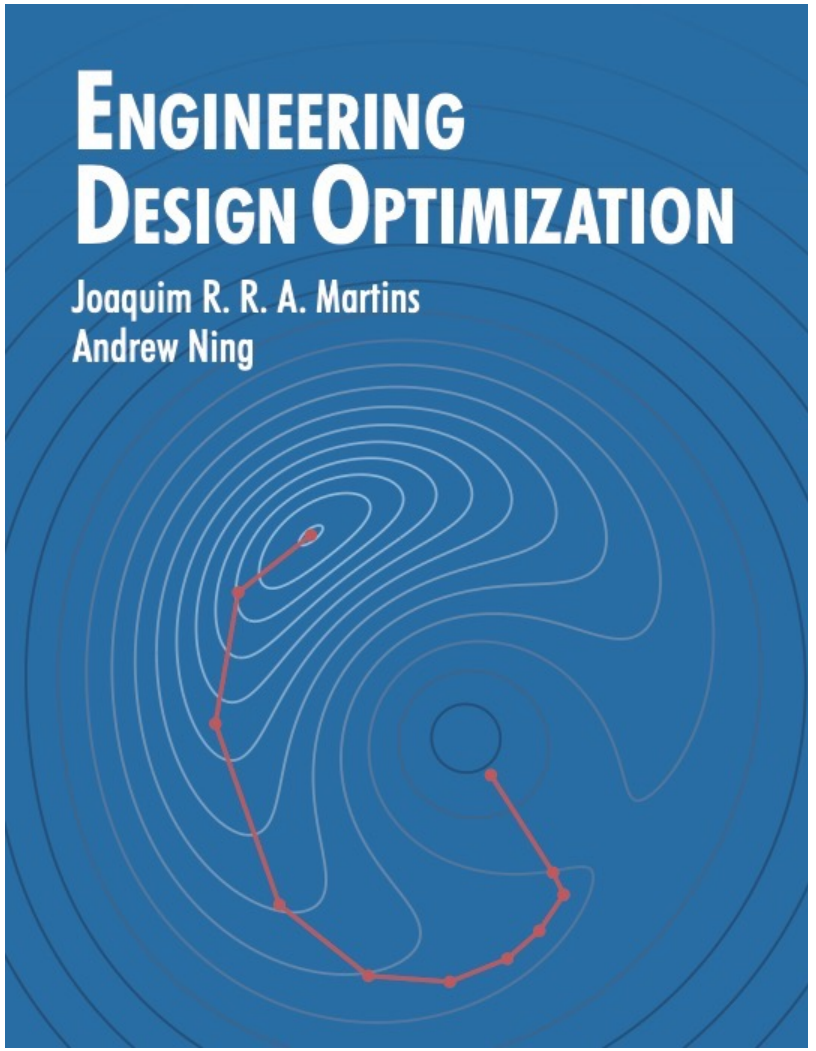


DAFoam: A Discrete-Adjoint for OpenFOAM



Review

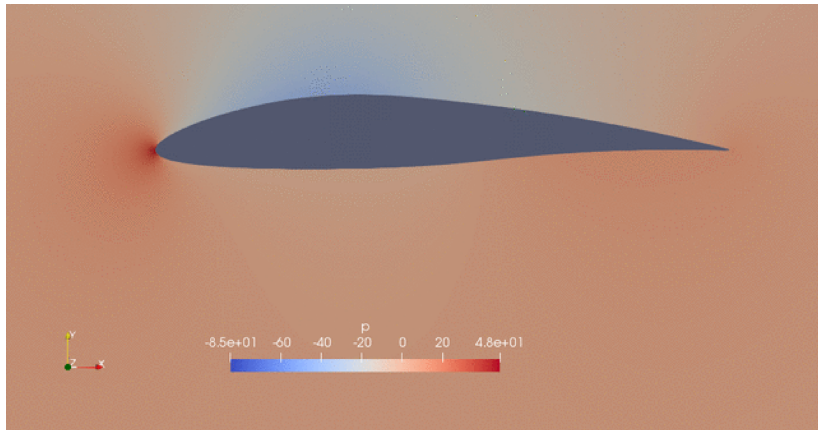
- What is multidisciplinary design optimization?
- How is aerodynamic shape optimization performed?
- Why are gradients instrumental for optimization?
- What is a discrete adjoint and why is it important?
- How are the required tools implemented and coupled?



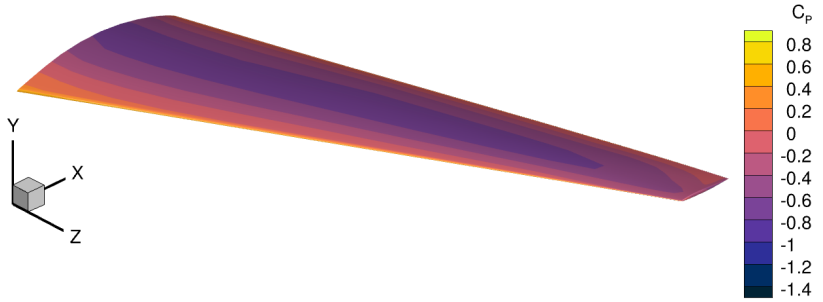
https://mdobook.github.io

Tutorials

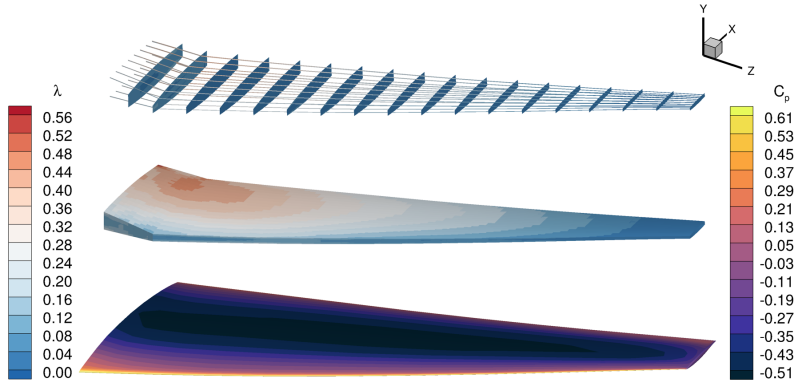
Part 1
Airfoil Optimization



Part 2
Wing Aerodynamic Optimization



Part 3
Wing Aerostructural Optimization



Git Clone the Run Files

git clone git@github.com:bernardopacini/OpenFOAMWorkshop2023-DAFoamTutorial.git

OR

Download the Run Files

https://github.com/bernardopacini/OpenFOAMWorkshop2023-DAFoamTutorial

Learning Objectives

What you *will not* get out of this training:

- How to set up OpenFOAM cases
- How to generate a mesh
- How to set up a free-form deformation grid
- In depth instructions on how to use DAFoam and its many functionalities
- Details of how the discrete adjoint is implemented
- How to develop and contribute to DAFoam

What you *will* get out of this training:

- How to structure a DAFoam runscript
- A detailed description of example cases and their runscripts
- Experience running a DAFoam optimization
- An introduction to single-discipline (aerodynamic) and multi-discipline (aerostructural) optimization with DAFoam

For installation guides, tutorials, and more detailed documentation checkout the DAFoam website:
https://dafoam.github.io

Technical Notes

- Coupling framework is written in Python – all coupled tools must have a Python interface
- Optimization problems are defined in a single Python runscript, with the possibility of importing additional helper files
- DAFoam and the MPhys framework are designed for high-fidelity optimizations that are typically run on high performance computing clusters using message passing interface (MPI) parallelism
- DAFoam and the MPhys framework operate in-memory, meaning there is no reliance on file IO

Optimization Runscript Structure

```
class Top(Multipoint):

    def setup(self):
```

MPhys optimization problem “group”

- Initialize the individual components
  - Geometry, mesh warping, flow solver
- Declare configuration files for each solver
- Set up connections between components

```
    def configure(self):
```

- Configure optimization surfaces
- Declare design variables
- Declare constraints
- Finalize model connections

OpenMDAO problem

Instantiate the Docker Container

Linux

```
docker run -it --rm -u dafoamuser --mount "type=bind,src=$(pwd),target=/home/dafoamuser/mount" -w
/home/dafoamuser/mount dafoam/opt-packages:v3.0.6 bash
```

Mac

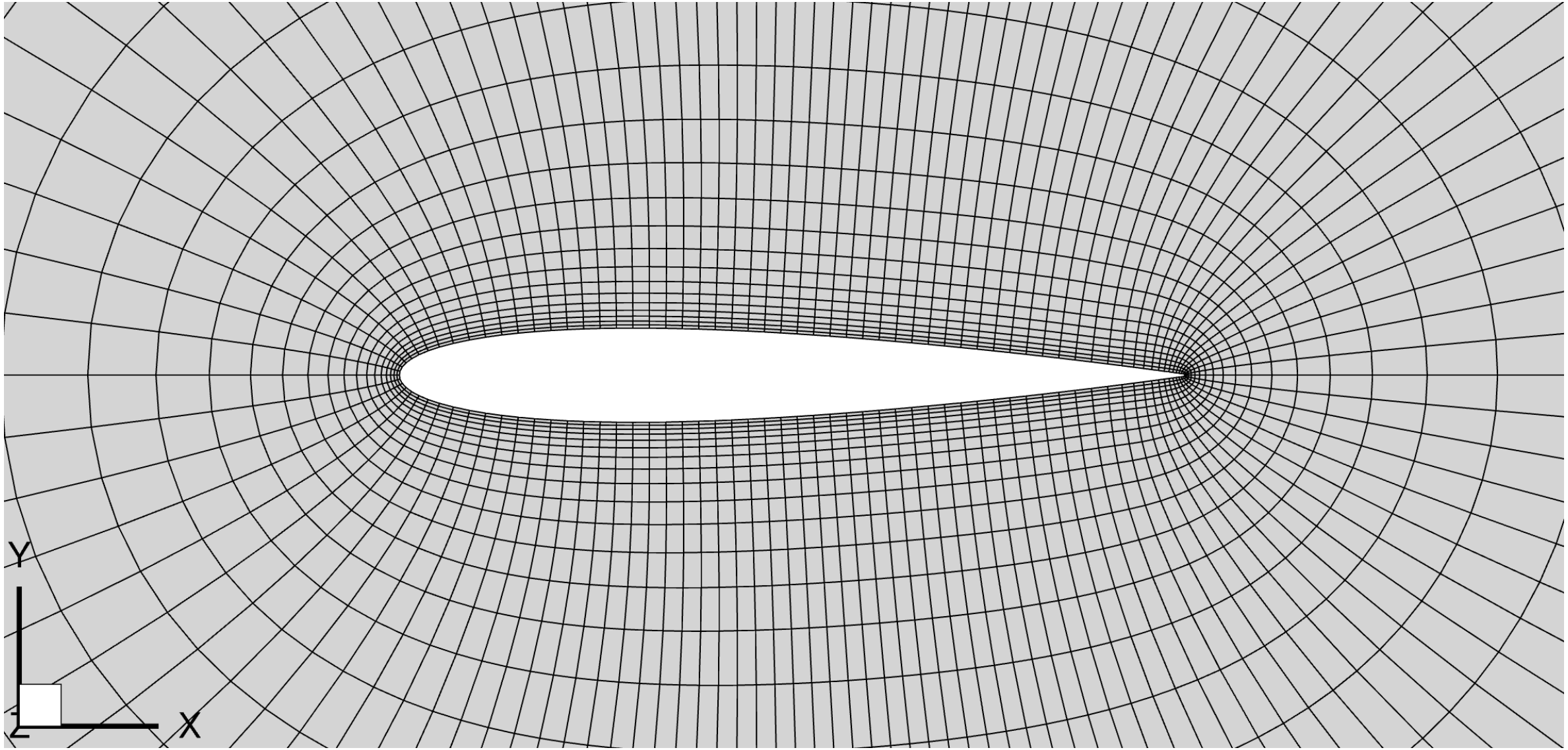
```
docker run -it --rm -u dafoamuser --mount "type=bind,src=$(pwd),target=/home/dafoamuser/mount" -w
/home/dafoamuser/mount dafoam/opt-packages:v3.0.6 bash
```

Windows

```
docker run -it --rm -u dafoamuser --mount "type=bind,src=%cd%,target=/home/dafoamuser/mount" -w
/home/dafoamuser/mount dafoam/opt-packages:v3.0.6 bash
```

**Increase the memory available to Docker – minimum requirement: 16GB*

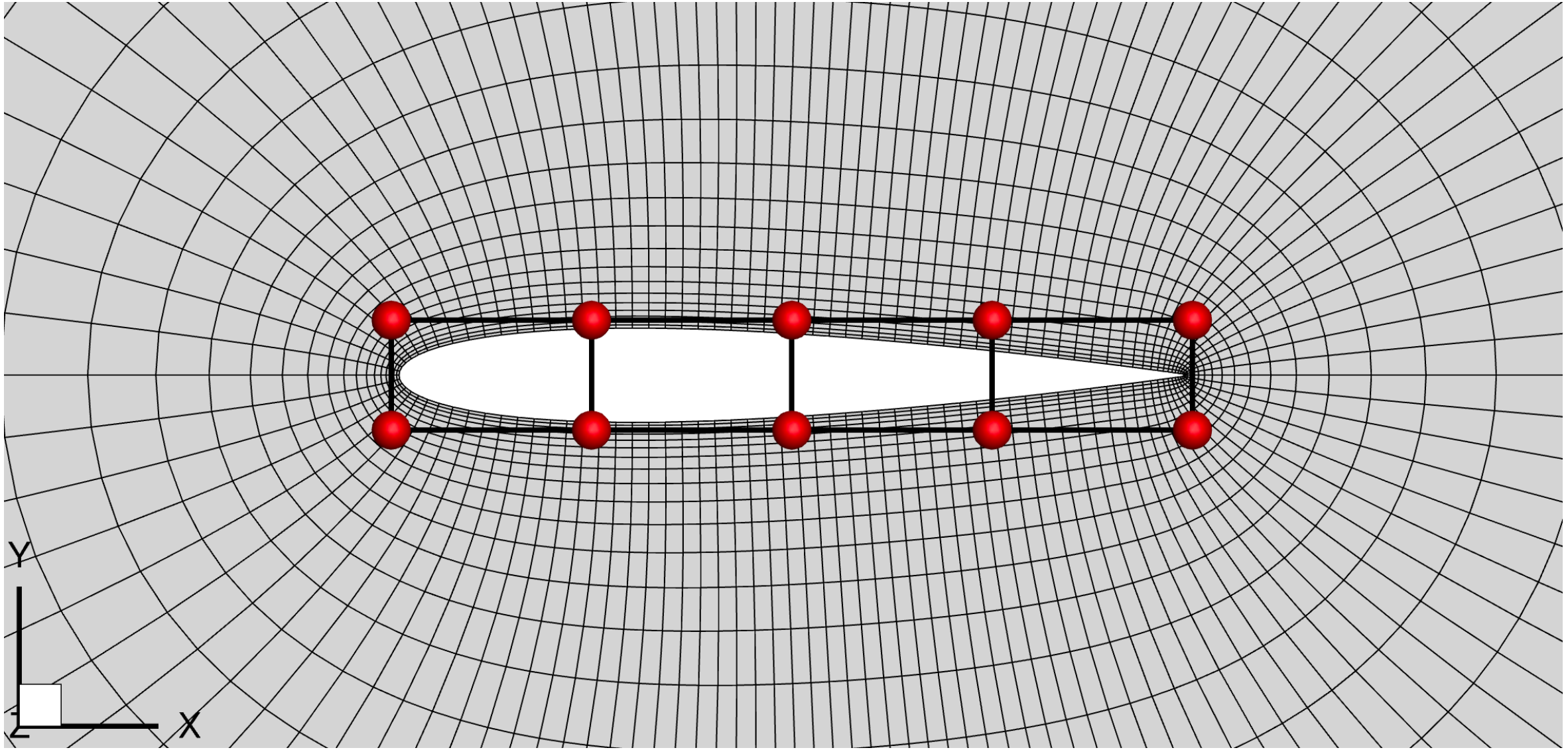
Airfoil Optimization



Airfoil Optimization Problem Statement

	Function or Variable	Unit	Description	Quantity
Minimize	C_D	-	Drag coefficient	1
			Total objectives	1
With respect to	$0 \leq \alpha < 10$	deg	Angle of attack	1
	$-1 \leq \Delta z \leq 1$	m	Vertical displacement of FFD points	20
			Total design variables	21
Subject to	$C_L = 0.5$	-	Lift coefficient	1
	$0.5 \cdot t_{bl} \leq \mathrm{t} \leq 3 \cdot t_{bl}$	m	Thickness constraint	20
	$\mathrm{V}_{bl} \leq \mathrm{V}$	m^3	Volume constraint	1
	$\Delta z_{LE,upper} = \Delta z_{LE,lower}$	m	Fixed leading edge constraint	2
	$\Delta z_{TE,upper} = \Delta z_{TE,lower}$	m	Fixed trailing edge constraint	2
	Symmetry Constraint	m	2D symmetry constraint	10
			Total constraints	36

Airfoil Optimization Problem Statement



Runscript – Package Imports and Runtime Arguments

```python
import os
import argparse
import numpy as np
import json
from mpi4py import MPI
import openmdao.api as om
from mphys.multipoint import Multipoint
from mphys.scenario_aerodynamic import ScenarioAerodynamic
from dafoam.mphys import DAFoamBuilder, OptFuncs
from pygeo.mphys import OM_DVGEOCOMP
from pygeo import geo_utils
```

Coupling framework

DAFoam

Geometry engine

```python
parser = argparse.ArgumentParser()
# which optimizer to use. Options are: IPOPT (default), SLSQP, and SNOPT
parser.add_argument("-optimizer", help="optimizer to use", type=str, default="IPOPT")
# which task to run. Options are: opt (default), runPrimal, runAdjoint, checkTotals
parser.add_argument("-task", help="type of run to do", type=str, default="opt")
args = parser.parse_args()

daOptions = {}
```

Runtime arguments

DAFoam options dictionary

Runscript – Options Dictionary OpenFOAM Setup

```
U0 = 10.0
p0 = 0.0
nuTilda0 = 4.5e-5
CL_target = 0.5
aoa0 = 5.0
A0 = 0.1
# rho is used for normalizing CD and CL
rho0 = 1.0

# Input parameters for DAFoam
daOptions = {
    "designSurfaces": ["wing"],
    "solverName": "DASimpleFoam",
    "primalMinResTol": 1.0e-8,
    "primalBC": {
        "U0": {"variable": "U", "patches": ["inout"], "value": [U0, 0.0, 0.0]},
        "p0": {"variable": "p", "patches": ["inout"], "value": [p0]},
        "nuTilda0": {"variable": "nuTilda", "patches": ["inout"], "value": [nuTilda0]},
        "useWallFunction": True,
},
```

Case input values

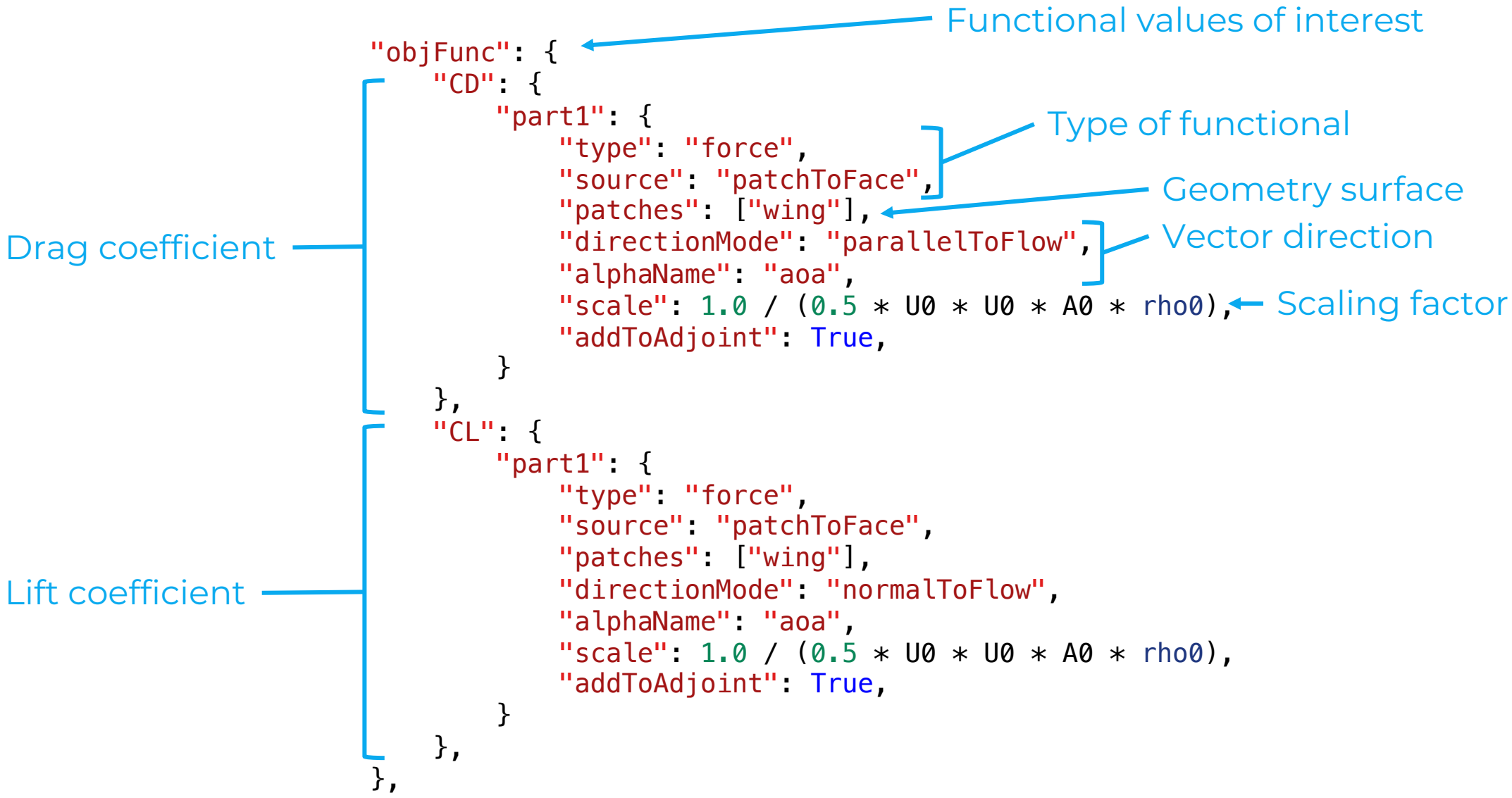
Design surface

OpenFOAM solver

Flow convergence tolerance

Boundary conditions

Runscript – Options Dictionary Functionals




```
"objFunc": {
    "CD": {
        "part1": {
            "type": "force",
            "source": "patchToFace",
            "patches": ["wing"],
            "directionMode": "parallelToFlow",
            "alphaName": "aoa",
            "scale": 1.0 / (0.5 * U0 * U0 * A0 * rho0),
            "addToAdjoint": True,
        }
    },
    "CL": {
        "part1": {
            "type": "force",
            "source": "patchToFace",
            "patches": ["wing"],
            "directionMode": "normalToFlow",
            "alphaName": "aoa",
            "scale": 1.0 / (0.5 * U0 * U0 * A0 * rho0),
            "addToAdjoint": True,
        }
    },
},
```

Runscript – Options Dictionary Derivatives

```
    "adjEqnOption": {"gmresRelTol": 1.0e-6, "pcFillLevel": 1, "jacMatReOrdering": "rcm"},
    "normalizeStates": {
        "U": U0,
        "p": U0 * U0 / 2.0,
        "nuTilda": nuTilda0 * 10.0,
        "phi": 1.0,
    },


    "designVar": {
        "aoa": {"designVarType": "AOA", "patches": ["inout"], "flowAxis": "x", "normalAxis": "y"},
        "shape": {"designVarType": "FFD"},
    },
}
```

Adjoint solver options

Adjoint state normalization factors

Design variable dictionary

End of options dictionary

Runscript – Mesh Warping Setup

```
# Mesh deformation setup
meshOptions = {
    "gridFile": os.getcwd(),
    "fileType": "OpenFOAM",
    # point and normal for the symmetry plane
    "symmetryPlanes": [[[0.0, 0.0, 0.0], [0.0, 0.0, 1.0]], [[0.0, 0.0, 0.1], [0.0, 0.0, 1.0]]],
}
```

Runscript – Setup

```
def setup(self):
    # create the builder to initialize the DASolvers
    dafoam_builder = DAFoamBuilder(daOptions, meshOptions, scenario="aerodynamic")
    dafoam_builder.initialize(self.comm)

    # add the design variable component to keep the top level design variables
    self.add_subsystem("dvs", om.IndepVarComp(), promotes=["*"])

    # add the mesh component
    self.add_subsystem("mesh", dafoam_builder.get_mesh_coordinate_subsystem())

    # add the geometry component (FFD)
    self.add_subsystem("geometry", OM_DVGEOCOMP(file="FFD/wingFFD.xyz", type="ffd"))

    # add a scenario (flow condition) for optimization, we pass the builder.
    # to the scenario to actually run the flow and adjoint
    self.mphys_add_scenario("cruise", ScenarioAerodynamic(aero_builder=dafoam_builder))

    # need to manually connect the x_aero0 between the mesh and geometry components
    # here x_aero0 means the surface coordinates of structurally undeformed mesh
    self.connect("mesh.x_aero0", "geometry.x_aero_in")
    # need to manually connect the x_aero0 between the geometry component and the cruise
    # scenario group
    self.connect("geometry.x_aero0", "cruise.x_aero")
```

Initialize DAFoam

Design variable manager

Mesh handler

Initialize geometry engine

Setup flight scenario

Setup geometry + mesh connections

Runscript – Configure Functionals and Design Surface

```
def configure(self):
    # configure and setup perform a similar function, i.e., initialize the optimization.
    # But configure will be run after setup

    # add the objective function to the cruise scenario
    self.cruise.aero_post.mphys_add_funcs()

    # get the surface coordinates from the mesh component
    points = self.mesh.mphys_get_surface_mesh()

    # add pointset to the geometry component
    self.geometry.nom_add_discipline_coords("aero", points)

    # set the triangular points to the geometry component for geometric constraints
    tri_points = self.mesh.mphys_get_triangulated_surface()
    self.geometry.nom_setConstraintSurface(tri_points)
    .
    .
    .
```

Propagate DAFoam functionals through model

Add DAFoam design surface mesh to geometry parametrization and constraint handler

Runscript – Configure Angle of Attack and Shape Design Variables

```
def configure(self):
    .
    .
    .
    # define an angle of attack function to change the U direction at the far field
    def aoa(val, DASolver):
        aoa = val[0] * np.pi / 180.0
        U = [float(U0 * np.cos(aoa)), float(U0 * np.sin(aoa)), 0]
        # we need to update the U value only
        DASolver.setOption("primalBC", {"U0": {"value": U}})
        DASolver.updateDAOption()

    # pass this aoa function to the cruise group
    self.cruise.coupling.solver.add_dv_func("aoa", aoa)
    self.cruise.aero_post.add_dv_func("aoa", aoa)

    # select the FFD points to move
    pts = self.geometry.DVGeo.getLocalIndex(0)
    indexList = pts[:, :, :].flatten()
    PS = geo_utils.PointSelect("list", indexList)
    nShapes = self.geometry.nom_addLocalDV(dvName="shape", pointSelect=PS)
    .
    .
    .
```

Define angle of attack function

Add angle of attack function to model

Add shape design variables

Runscript – Configure Constraints

```
def configure(self):
    .
    .
    .
    # setup the symmetry constraint to link the y displacement between k=0 and k=1
    nFFDs_x = pts.shape[0]
    nFFDs_y = pts.shape[1]
    indSetA = []
    indSetB = []
    for i in range(nFFDs_x):
        for j in range(nFFDs_y):
            indSetA.append(pts[i, j, 0])
            indSetB.append(pts[i, j, 1])
    self.geometry.nom_addLinearConstraintsShape("linearcon", indSetA, indSetB, factorA=1.0, factorB=-1.0)

    # setup the volume and thickness constraints
    leList = [[1e-4, 0.0, 1e-4], [1e-4, 0.0, 0.1 - 1e-4]]
    teList = [[0.998 - 1e-4, 0.0, 1e-4], [0.998 - 1e-4, 0.0, 0.1 - 1e-4]]
    self.geometry.nom_addThicknessConstraints2D("thickcon", leList, teList, nSpan=2, nChord=10)
    self.geometry.nom_addVolumeConstraint("volcon", leList, teList, nSpan=2, nChord=10)
    # add the LE/TE constraints
    self.geometry.nom_add_LETEConstraint("lecon", volID=0, faceID="iLow", topID="k")
    self.geometry.nom_add_LETEConstraint("tecon", volID=0, faceID="iHigh", topID="k")
    .
    .
    .
```

Define displacement constraint for 2D optimization

Define LE / TE lines

Add thickness constraint

Add volume constraint

Add LE / TE constraints

Runscript – Configure Design Variables, Objective, and Constraints

```
def configure(self):
    .
    .
    .
    # add the design variables to the dvs component's output
    self.dvs.add_output("shape", val=np.array([0] * nShapes))
    self.dvs.add_output("aoa", val=np.array([aoa0]))
    # manually connect the dvs output to the geometry and cruise
    self.connect("aoa", "cruise.aoa")
    self.connect("shape", "geometry.shape")

    # define the design variables to the top level
    self.add_design_var("shape", lower=-1.0, upper=1.0, scaler=1.0)
    self.add_design_var("aoa", lower=0.0, upper=10.0, scaler=1.0)

    # add objective and constraints to the top level
    self.add_objective("cruise.aero_post.CD", scaler=1.0)
    self.add_constraint("cruise.aero_post.CL", equals=CL_target, scaler=1.0)
    self.add_constraint("geometry.thickcon", lower=0.5, upper=3.0, scaler=1.0)
    self.add_constraint("geometry.volcon", lower=1.0, scaler=1.0)
    self.add_constraint("geometry.tecon", equals=0.0, scaler=1.0, linear=True)
    self.add_constraint("geometry.lecon", equals=0.0, scaler=1.0, linear=True)
    self.add_constraint("geometry.linearcon", equals=0.0, scaler=1.0, linear=True)
```

Define independent variables and connect them

Add design variables to model

Add objective function to model

Add constraints to model

Runscript – Initialize Optimization Problem

```
# OpenMDAO setup
prob = om.Problem()
prob.model = Top()
prob.setup(mode="rev")
om.n2(prob, show_browser=False, outfile="mphys.html")

# initialize the optimization function
optFuncs = OptFuncs(daOptions, prob)
```

Initialize OpenMDAO problem

Setup DAFoam helper functions

Runscript – Setup Optimizer

```
# use pyoptsparse to setup optimization
prob.driver = om.pyOptSparseDriver()
prob.driver.options["optimizer"] = args.optimizer
# options for optimizers
if args.optimizer == "SNOPT":
    .
    .
    .
elif args.optimizer == "IPOPT":
    prob.driver.opt_settings = {
        "tol": 1.0e-5,
        "constr_viol_tol": 1.0e-5,
        "max_iter": 100,
        "print_level": 5,
        "output_file": "opt_IPOPT.txt",
        "mu_strategy": "adaptive",
        "limited_memory_max_history": 10,
        "nlp_scaling_method": "none",
        "alpha_for_y": "full",
        "recalc_y": "yes",
}
elif args.optimizer == "SLSQP":
    .
    .
    .
else:
    print("optimizer arg not valid!")
    exit(1)
```

Connect an optimizer

Declare optimizer options

Runscript – Task

```
prob.driver.options["debug_print"] = ["nl_cons", "objs", "desvars"]
prob.driver.options["print_opt_prob"] = True
prob.driver.hist_file = "OptView.hst"


if args.task == "opt":
    # solve CL
    optFuncs.findFeasibleDesign(["cruise.aero_post.CL"], ["aoa"], targets=[CL_target])
    # run the optimization
    prob.run_driver()
    .
    .
    .
elif args.task == "runPrimal":
    # just run the primal once
    prob.run_model()
elif args.task == "runAdjoint":
    .
    .
    .
elif args.task == "checkTotals":
    .
    .
    .
else:
    print("task arg not found!")
    exit(1)
```

Set runtime log preferences

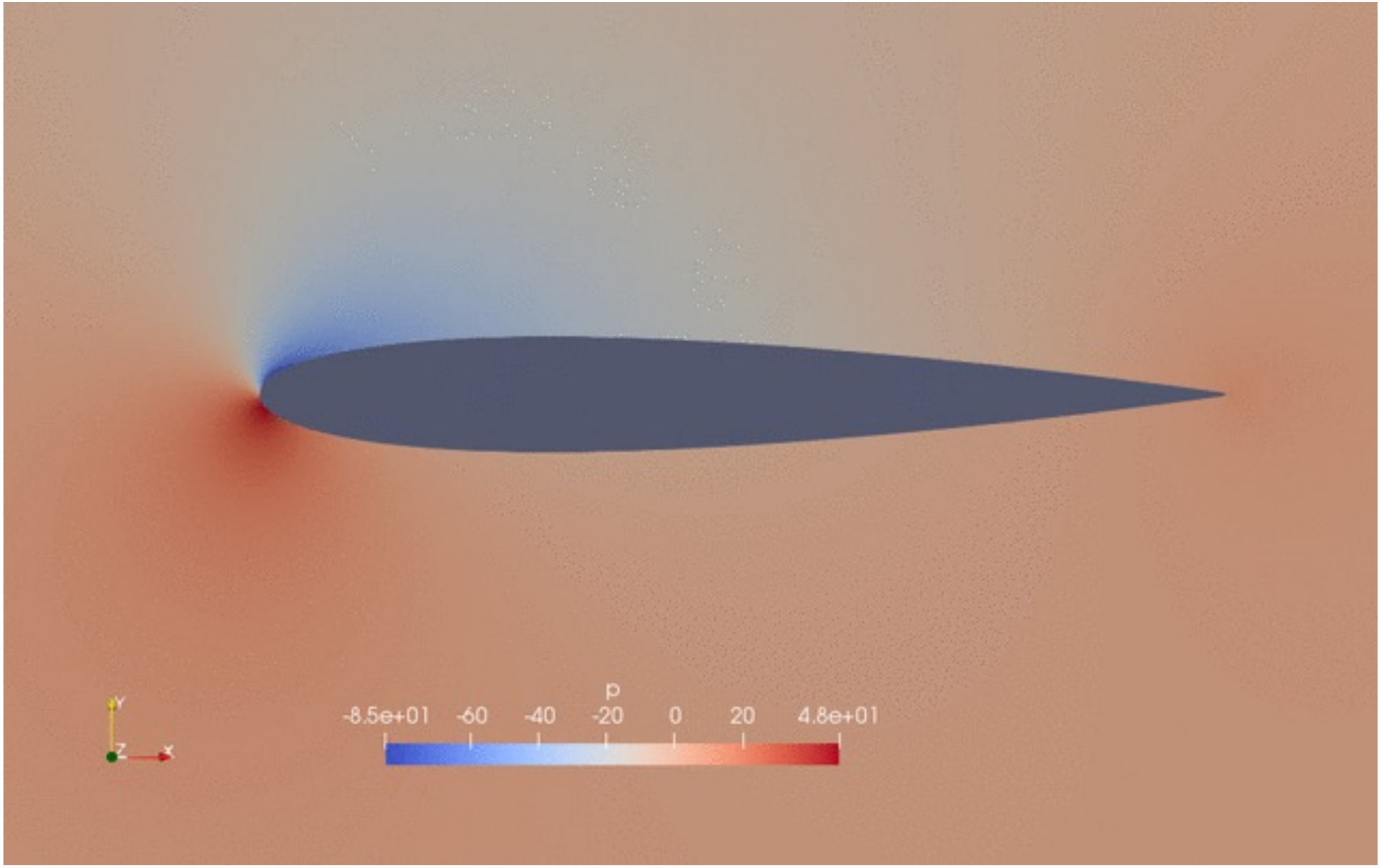
Trim baseline design

Optimize!

Run one analysis

Let’s run it!

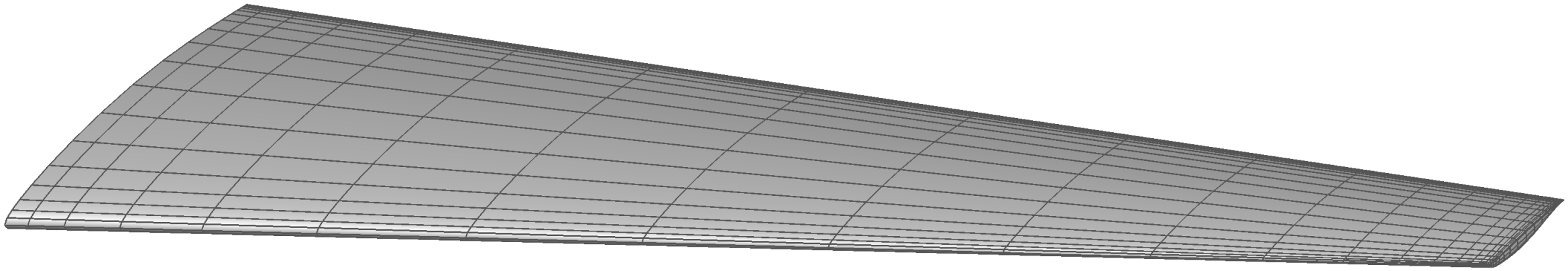
Result



Review

- Imported packages for each required component and coupling
- Declared the simulation parameters in the DAFoamOptions dictionary
- Initialized optimization components in the setup() method
- Assembled the optimization problem with the configure() method
- Created an OpenMDAO problem and connected our model
- Configured an optimizer
- Ran the optimization!

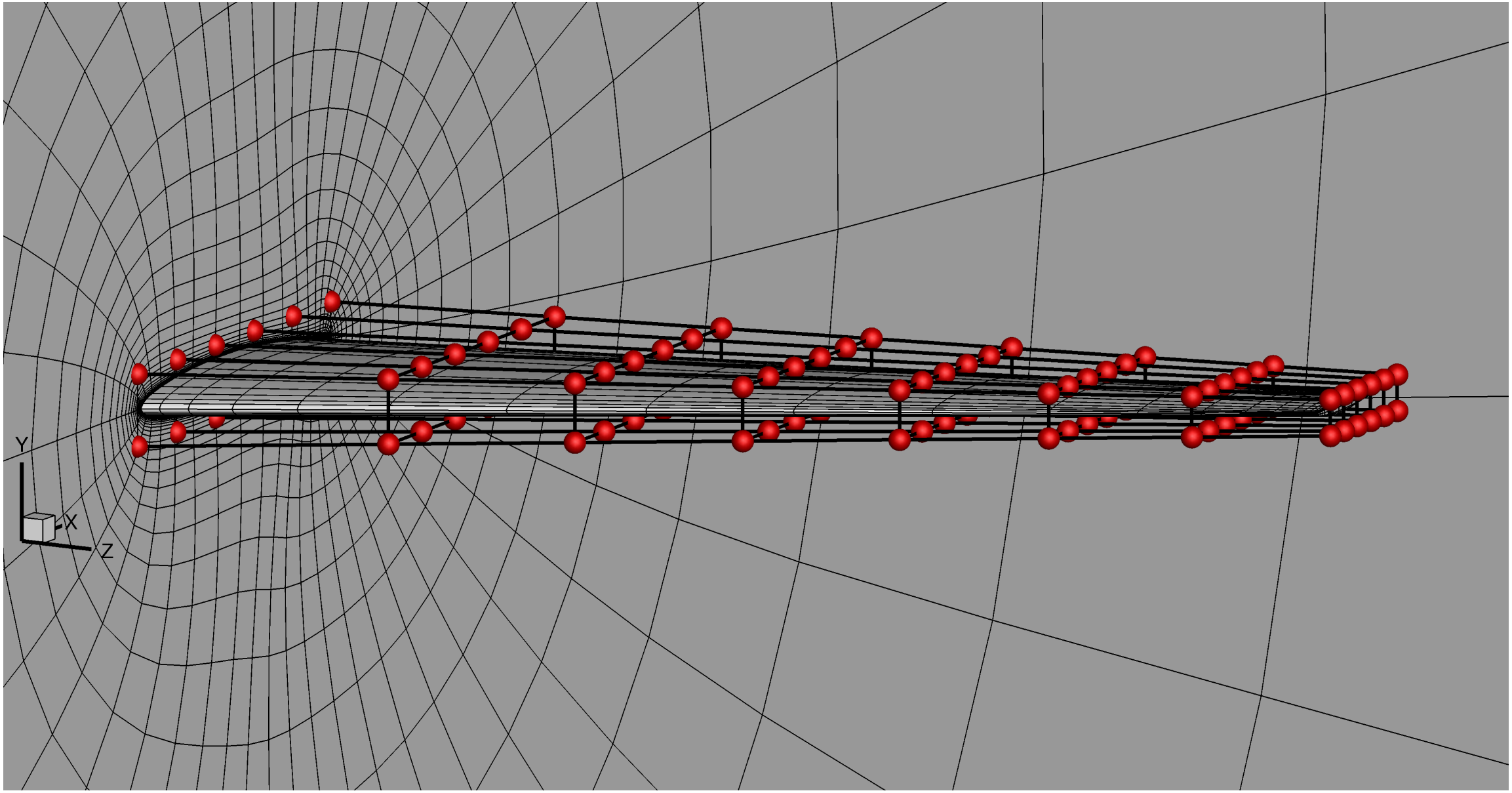
Wing Aerodynamic Optimization



Wing Aerodynamic Optimization Problem Statement

	Function or Variable	Unit	Description	Quantity
Minimize	C_D	-	Drag coefficient	1
			Total objectives	1
With respect to	$0 \leq \alpha < 10$	deg	Angle of attack	1
	$-10 \leq \gamma \leq 10$	deg	Twist of each FFD section	7
	$-1 \leq \Delta z \leq 1$	m	Vertical displacement of FFD points	64
			Total design variables	72
Subject to	$C_L = 0.5$	-	Lift coefficient	1
	$0.5 \cdot t_{bl} \leq \mathrm{t} \leq 3 \cdot t_{bl}$	m	Thickness constraint	100
	$\mathrm{V}_{bl} \leq \mathrm{V}$	m^3	Volume constraint	1
	$\Delta z_{LE,upper} = \Delta z_{LE,lower}$	m	Fixed leading edge constraint	7
	$\Delta z_{TE,upper} = \Delta z_{TE,lower}$	m	Fixed trailing edge constraint	7
			Total constraints	117

Wing Geometry Parametrization



The runscript is nearly identical to the airfoil optimization case

Runscript – Package Imports and Runtime Arguments

```python
import os
import argparse
import numpy as np
from mpi4py import MPI
import openmdao.api as om
from mphys.multipoint import Multipoint
from mphys.scenario_aerodynamic import ScenarioAerodynamic
from dafoam.mphys import DAFoamBuilder, OptFuncs
from pygeo.mphys import OM_DVGEOCOMP
from pygeo import geo_utils

parser = argparse.ArgumentParser()
# which optimizer to use. Options are: IPOPT (default), SLSQP, and SNOPT
parser.add_argument("-optimizer", help="optimizer to use", type=str, default="IPOPT")
# which task to run. Options are: opt (default), runPrimal, runAdjoint, checkTotals
parser.add_argument("-task", help="type of run to do", type=str, default="opt")
args = parser.parse_args()

daOptions = {}
```

Coupling framework

DAFoam

Geometry engine

Runtime arguments

DAFoam options dictionary

Runscript – Options Dictionary OpenFOAM Setup

```
U0 = 100.0
p0 = 101325.0
nuTilda0 = 4.5e-5
T0 = 300.0
CL_target = 0.5
aoa0 = 2.0
rho0 = p0 / T0 / 287.0
A0 = 45.5

daOptions = {
    "designSurfaces": ["wing"],
    "solverName": "DARhoSimpleFoam",
    "primalMinResTol": 1.0e-8,
    "primalBC": {
        "U0": {"variable": "U", "patches": ["inout"], "value": [U0, 0.0, 0.0]},
        "p0": {"variable": "p", "patches": ["inout"], "value": [p0]},
        "T0": {"variable": "T", "patches": ["inout"], "value": [T0]},
        "nuTilda0": {"variable": "nuTilda", "patches": ["inout"], "value": [nuTilda0]},
        "useWallFunction": True,
    },
```

Case input values

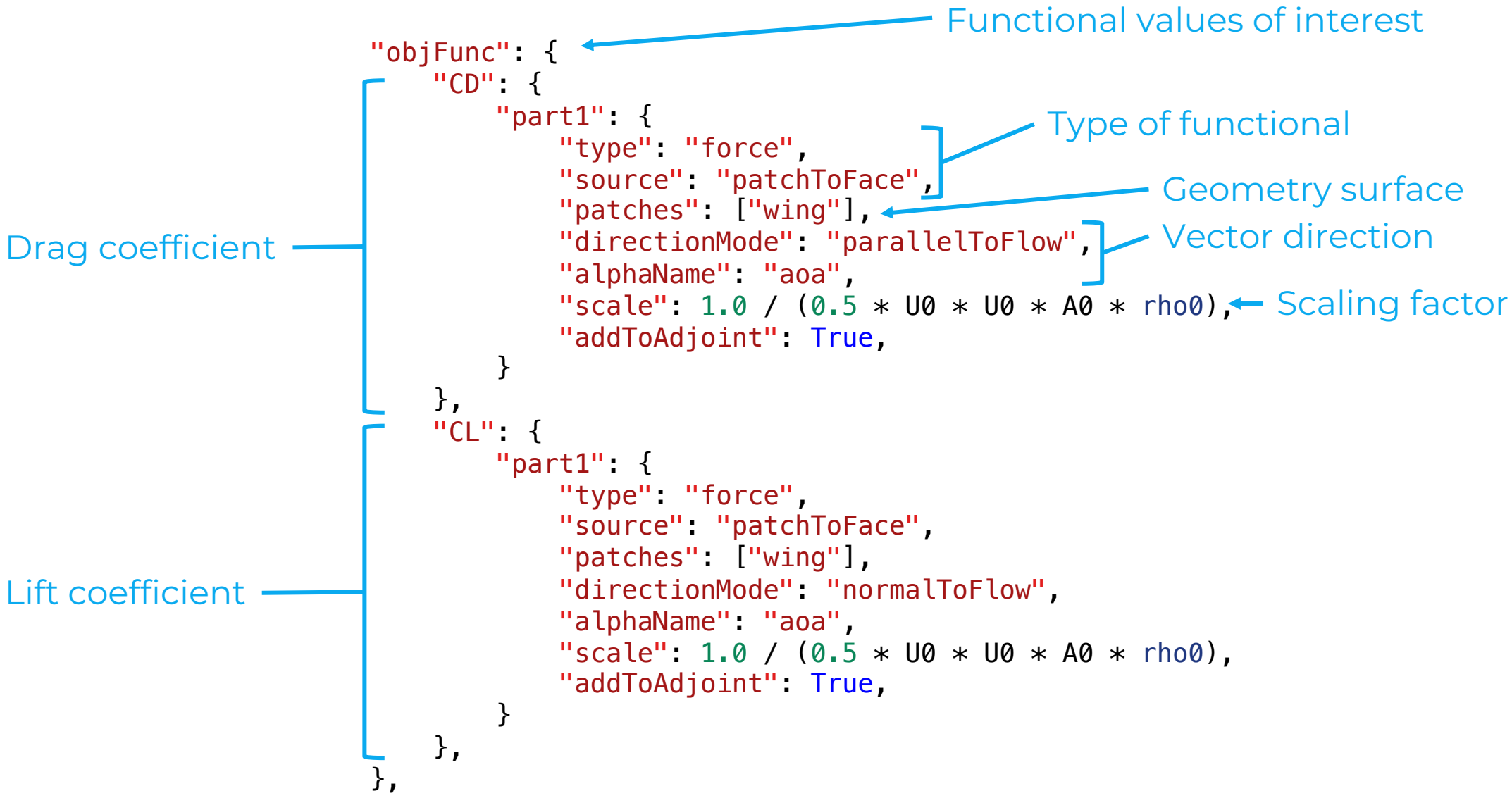
Design surface

OpenFOAM solver

Flow convergence tolerance

Boundary conditions

Runscript – Options Dictionary Functionals




```
"objFunc": {
    "CD": {
        "part1": {
            "type": "force",
            "source": "patchToFace",
            "patches": ["wing"],
            "directionMode": "parallelToFlow",
            "alphaName": "aoa",
            "scale": 1.0 / (0.5 * U0 * U0 * A0 * rho0),
            "addToAdjoint": True,
        }
    },
    "CL": {
        "part1": {
            "type": "force",
            "source": "patchToFace",
            "patches": ["wing"],
            "directionMode": "normalToFlow",
            "alphaName": "aoa",
            "scale": 1.0 / (0.5 * U0 * U0 * A0 * rho0),
            "addToAdjoint": True,
        }
    },
},
```

Runscript – Options Dictionary Derivatives

```
    "adjEqnOption": {"gmresRelTol": 1.0e-6, "pcFillLevel": 1, "jacMatReOrdering": "rcm"},
    "normalizeStates": {
    "U": U0,
    "p": U0 * U0 / 2.0,
    "T": T0,
    "nuTilda": 1e-3,
    "phi": 1.0,
},
"checkMeshThreshold": {
    "maxAspectRatio": 1000.0,
    "maxNonOrth": 70.0,
    "maxSkewness": 5.0,
},
"designVar": {
    "aoa": {"designVarType": "AOA", "patches": ["inout"], "flowAxis": "x", "normalAxis": "y"},
    "twist": {"designVarType": "FFD"},
    "shape": {"designVarType": "FFD"},
},
}
```

Adjoint solver options

Adjoint state normalization factors

Mesh quality check thresholds

Design variable dictionary

End of options dictionary

Runscript – Mesh Warping Setup

```
# Mesh deformation setup
meshOptions = {
    "gridFile": os.getcwd(),
    "fileType": "OpenFOAM",
    # point and normal for the symmetry plane
    "symmetryPlanes": [[[0.0, 0.0, 0.0], [0.0, 0.0, 1.0]]],
}
```

Runscript – Setup

```
class Top(Multipoint):
    def setup(self):

        # create the builder to initialize the DASolvers
        dafoam_builder = DAFoamBuilder(daOptions, meshOptions, scenario="aerodynamic")
        dafoam_builder.initialize(self.comm)

        # add the design variable component to keep the top level design variables
        self.add_subsystem("dvs", om.IndepVarComp(), promotes=["*"])

        # add the mesh component
        self.add_subsystem("mesh", dafoam_builder.get_mesh_coordinate_subsystem())

        # add the geometry component (FFD)
        self.add_subsystem("geometry", OM_DVGEOCOMP(file="FFD/wingFFD.xyz", type="ffd"))

        # add a scenario (flow condition) for optimization, we pass the builder
        # to the scenario to actually run the flow and adjoint
        self.mphys_add_scenario("cruise", ScenarioAerodynamic(aero_builder=dafoam_builder))

        # need to manually connect the x_aero0 between the mesh and geometry components
        # here x_aero0 means the surface coordinates of structurally undeformed mesh
        self.connect("mesh.x_aero0", "geometry.x_aero_in")
        # need to manually connect the x_aero0 between the geometry component and the cruise
        # scenario group
        self.connect("geometry.x_aero0", "cruise.x_aero")
```

Initialize DAFoam

Design variable manager

Mesh handler

Initialize geometry engine

Setup flight scenario

Setup geometry + mesh connections

Runscript – Configure Functionals and Design Surface

```
def configure(self):
    # configure and setup perform a similar function, i.e., initialize the optimization.
    # But configure will be run after setup

    # add the objective function to the cruise scenario
    self.cruise.aero_post.mphys_add_funcs()

    # get the surface coordinates from the mesh component
    points = self.mesh.mphys_get_surface_mesh()

    # add pointset to the geometry component
    self.geometry.nom_add_discipline_coords("aero", points)

    # set the triangular points to the geometry component for geometric constraints
    tri_points = self.mesh.mphys_get_triangulated_surface()
    self.geometry.nom_setConstraintSurface(tri_points)
    .
    .
    .
```

Propagate DAFoam functionals through model

Add DAFoam design surface mesh to geometry parametrization and constraint handler

Runscript – Configure Twist Design Variables

```
def configure(self):
    .
    .
    .
    # Create reference axis for the twist variable
    nRefAxPts = self.geometry.nom_addRefAxis(name="wingAxis", xFraction=0.25, alignIndex="k")

    # Set up global design variables. We dont change the root twist
    def twist(val, geo):
        for i in range(1, nRefAxPts):
        geo.rot_z["wingAxis"].coef[i] = -val[i - 1]

    # add twist variable
    self.geometry.nom_addGlobalDV(dvName="twist", value=np.array([0] * (nRefAxPts - 1)), func=twist)
    .
    .
    .
```

Define wing twist axis

Define wing twist function

Add twist design variable to geometry parametrization

Runscript – Configure Angle of Attack and Shape Design Variables

```
def configure(self):
    .
    .
    .
    # define an angle of attack function to change the U direction at the far field
    def aoa(val, DASolver):
        aoa = val[0] * np.pi / 180.0
        U = [float(U0 * np.cos(aoa)), float(U0 * np.sin(aoa)), 0]
        # we need to update the U value only
        DASolver.setOption("primalBC", {"U0": {"value": U}})
        DASolver.updateDAOption()

    # pass this aoa function to the cruise group
    self.cruise.coupling.solver.add_dv_func("aoa", aoa)
    self.cruise.aero_post.add_dv_func("aoa", aoa)

    # select the FFD points to move
    pts = self.geometry.DVGeo.getLocalIndex(0)
    indexList = pts[:, :, :].flatten()
    PS = geo_utils.PointSelect("list", indexList)
    nShapes = self.geometry.nom_addLocalDV(dvName="shape", pointSelect=PS)
    .
    .
    .
```

Define angle of attack function

Add angle of attack function to model

Add shape design variables

Runscript – Configure Constraints

```
def configure(self):
    .
    .
    .
    # setup the volume and thickness constraints
    leList = [[0.1, 0, 0.01], [7.5, 0, 13.9]]
    teList = [[4.9, 0, 0.01], [8.9, 0, 13.9]]
    self.geometry.nom_addThicknessConstraints2D("thickcon", leList, teList, nSpan=10, nChord=10)
    self.geometry.nom_addVolumeConstraint("volcon", leList, teList, nSpan=10, nChord=10)
    # add the LE/TE constraints
    self.geometry.nom_add_LETEConstraint("lecon", volID=0, faceID="iLow")
    self.geometry.nom_add_LETEConstraint("tecon", volID=0, faceID="iHigh")
    .
    .
    .
```

Define LE / TE lines

Add thickness constraint

Add volume constraint

Add LE / TE constraints

Runscript – Configure Design Variables, Objective, and Constraints

```
def configure(self):
    .
    .
    .
    # add the design variables to the dvs component's output
    self.dvs.add_output("twist", val=np.array([0] * (nRefAxPts - 1)))
    self.dvs.add_output("shape", val=np.array([0] * nShapes))
    self.dvs.add_output("aoa", val=np.array([aoa0]))
    # manually connect the dvs output to the geometry and cruise
    self.connect("twist", "geometry.twist")
    self.connect("shape", "geometry.shape")
    self.connect("aoa", "cruise.aoa")

    # define the design variables
    self.add_design_var("twist", lower=-10.0, upper=10.0, scaler=1.0)
    self.add_design_var("shape", lower=-1.0, upper=1.0, scaler=1.0)
    self.add_design_var("aoa", lower=0.0, upper=10.0, scaler=1.0)

    # add objective and constraints to the top level
    self.add_objective("cruise.aero_post.CD", scaler=1.0)
    self.add_constraint("cruise.aero_post.CL", equals=CL_target, scaler=1.0)
    self.add_constraint("geometry.thickcon", lower=0.5, upper=3.0, scaler=1.0)
    self.add_constraint("geometry.volcon", lower=1.0, scaler=1.0)
    self.add_constraint("geometry.tecon", equals=0.0, scaler=1.0, linear=True)
    self.add_constraint("geometry.lecon", equals=0.0, scaler=1.0, linear=True)
```

Define independent variables and connect them

Add design variables to model

Add objective function to model

Add constraints to model

Runscript – Initialize Optimization Problem

```
# OpenMDAO setup
prob = om.Problem()
prob.model = Top()
prob.setup(mode="rev")
om.n2(prob, show_browser=False, outfile="mphys.html")

# initialize the optimization function
optFuncs = OptFuncs(daOptions, prob)
```

Initialize OpenMDAO problem

Setup DAFoam helper functions

Runscript – Setup Optimizer

```
# use pyoptsparse to setup optimization
prob.driver = om.pyOptSparseDriver()
prob.driver.options["optimizer"] = args.optimizer
# options for optimizers
if args.optimizer == "SNOPT":
    .
    .
    .
elif args.optimizer == "IPOPT":
    prob.driver.opt_settings = {
        "tol": 1.0e-5,
        "constr_viol_tol": 1.0e-5,
        "max_iter": 100,
        "print_level": 5,
        "output_file": "opt_IPOPT.txt",
        "mu_strategy": "adaptive",
        "limited_memory_max_history": 10,
        "nlp_scaling_method": "none",
        "alpha_for_y": "full",
        "recalc_y": "yes",
}
elif args.optimizer == "SLSQP":
    .
    .
    .
else:
    print("optimizer arg not valid!")
    exit(1)
```

Connect an optimizer

Declare optimizer options

Runscript – Task

```
prob.driver.options["debug_print"] = ["nl_cons", "objs", "desvars"]
prob.driver.options["print_opt_prob"] = True
prob.driver.hist_file = "OptView.hst"


if args.task == "opt":
    # solve CL
    optFuncs.findFeasibleDesign(["cruise.aero_post.CL"], ["aoa"], targets=[CL_target])
    # run the optimization
    prob.run_driver()
elif args.task == "runPrimal":
    # just run the primal once
    prob.run_model()
elif args.task == "runAdjoint":
    .
    .
    .
elif args.task == "checkTotals":
    .
    .
    .
else:
    print("task arg not found!")
    exit(1)
```

Set runtime log preferences

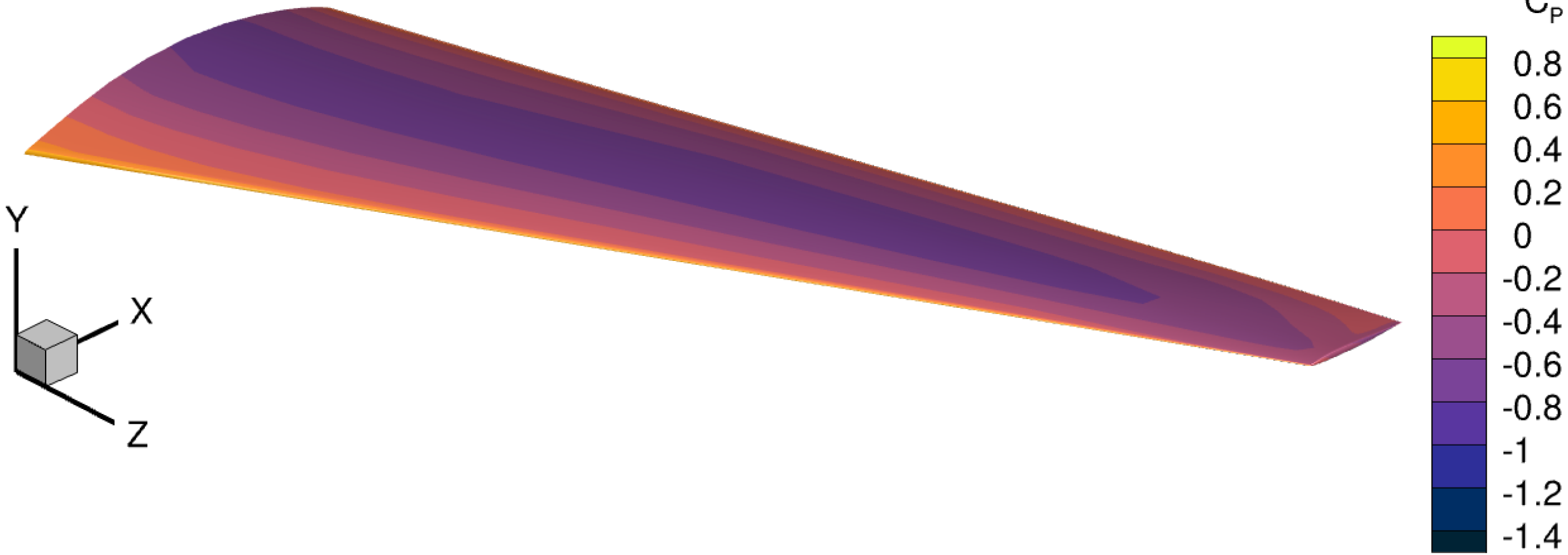
Trim baseline design

Optimize!

Run one analysis

Let’s run it!

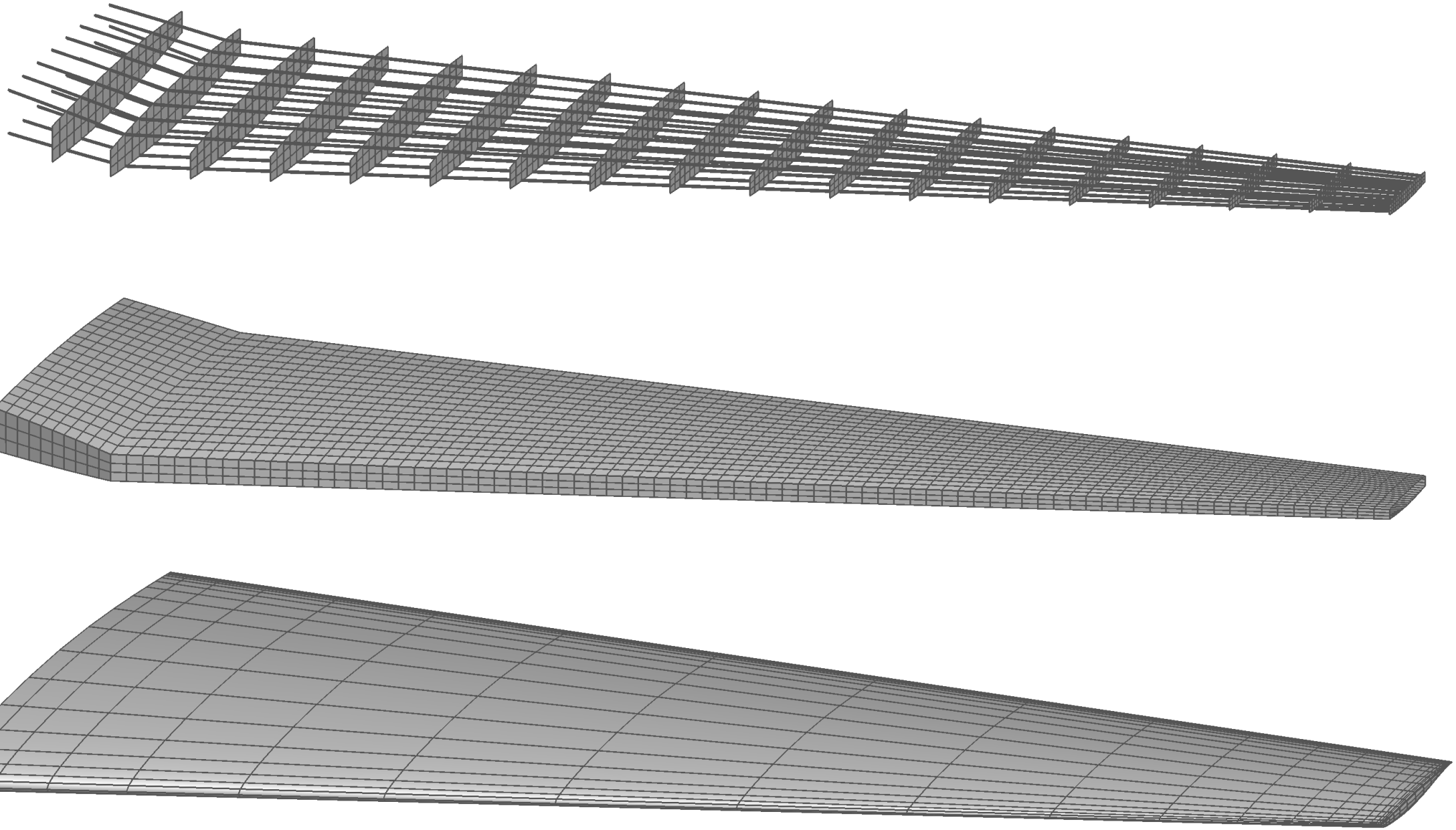
Result



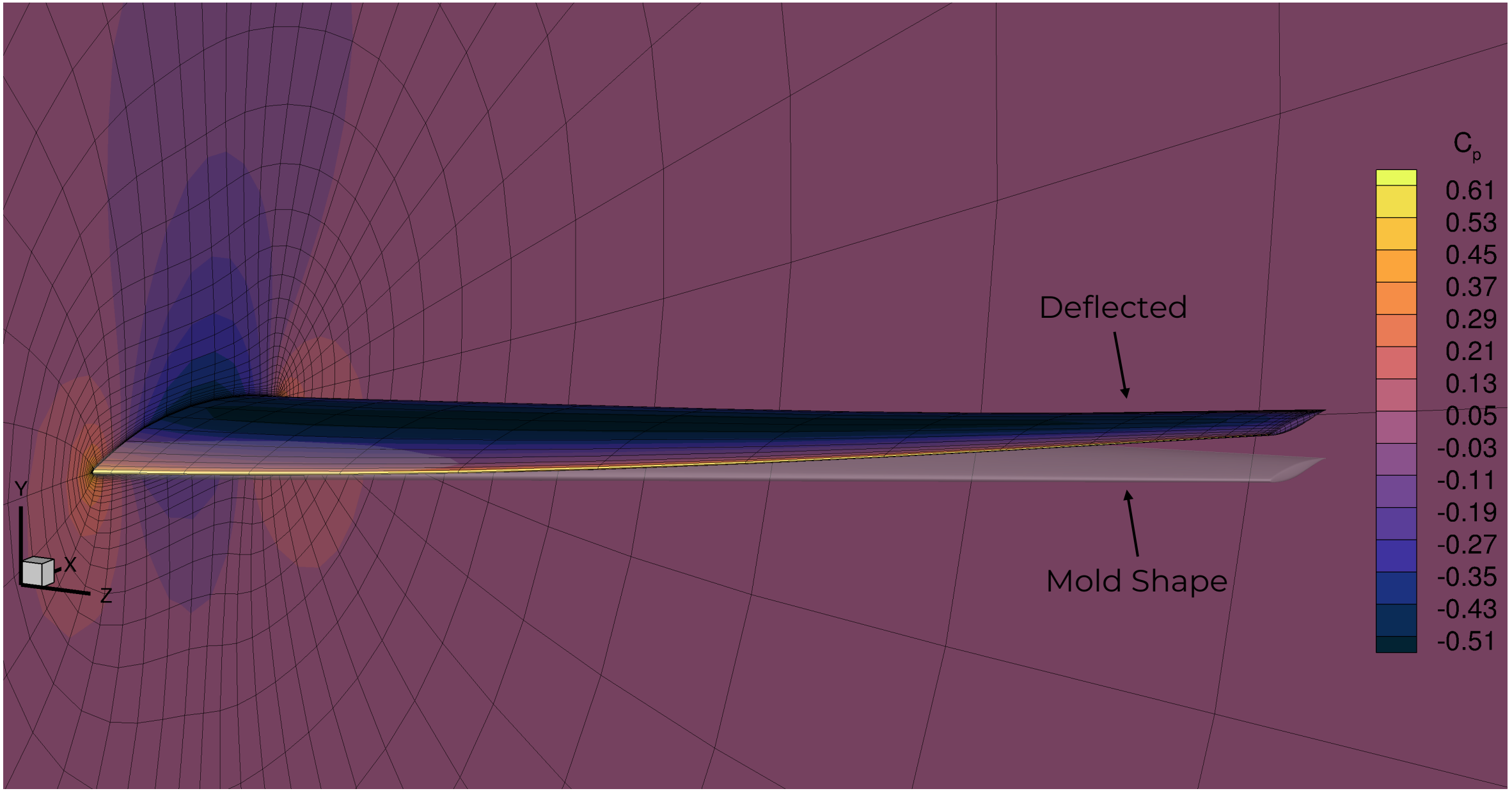
Review

- Imported packages for each required component and coupling
- Declared the simulation parameters in the DAFoamOptions dictionary
- Initialized optimization components in the setup() method
- Assembled the optimization problem with the configure() method
- Created an OpenMDAO problem and connected our model
- Configured an optimizer
- Ran the optimization!

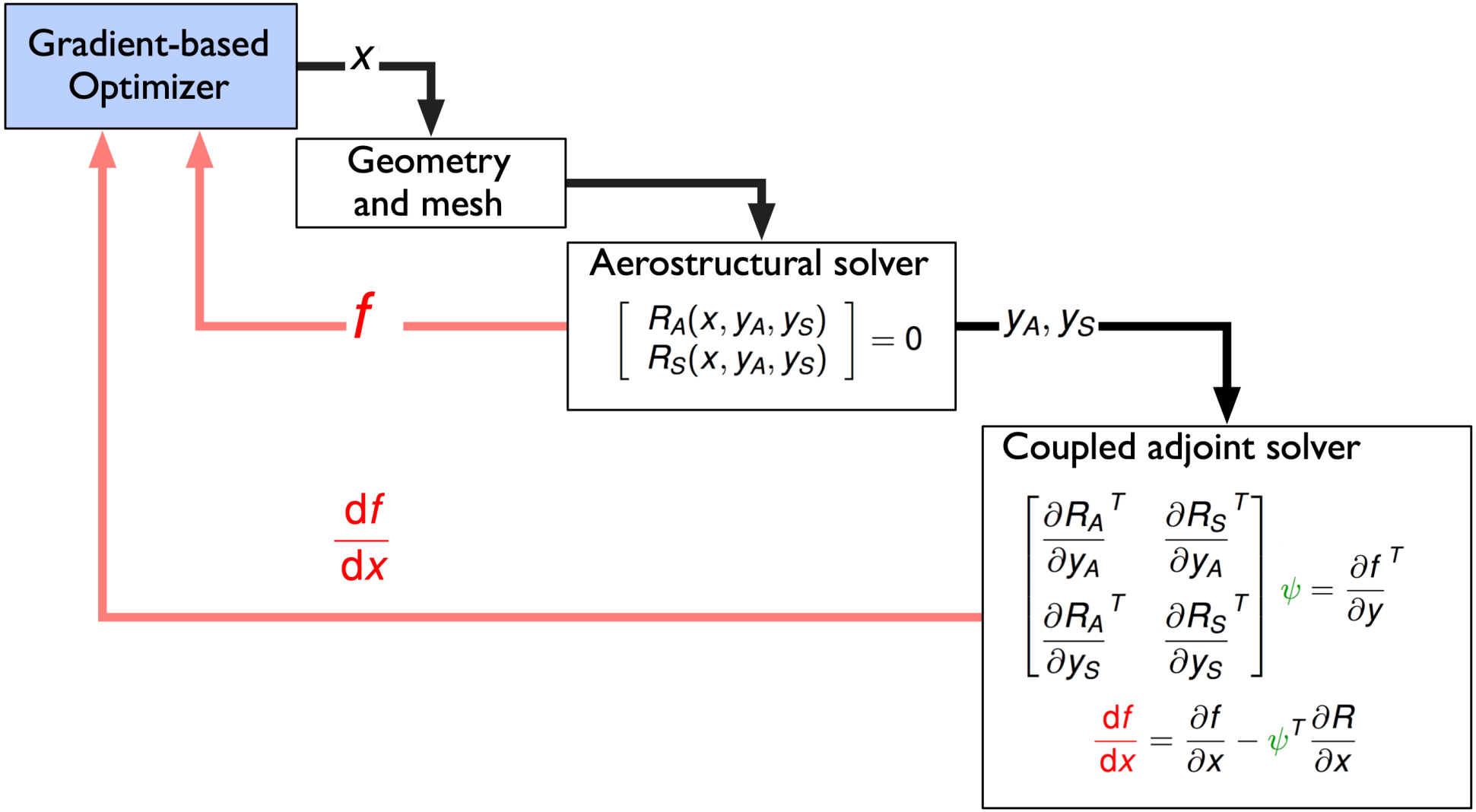
Wing Aerostructural Optimization



Aerostructural Simulations are Used to Model Fluid Structure Interaction

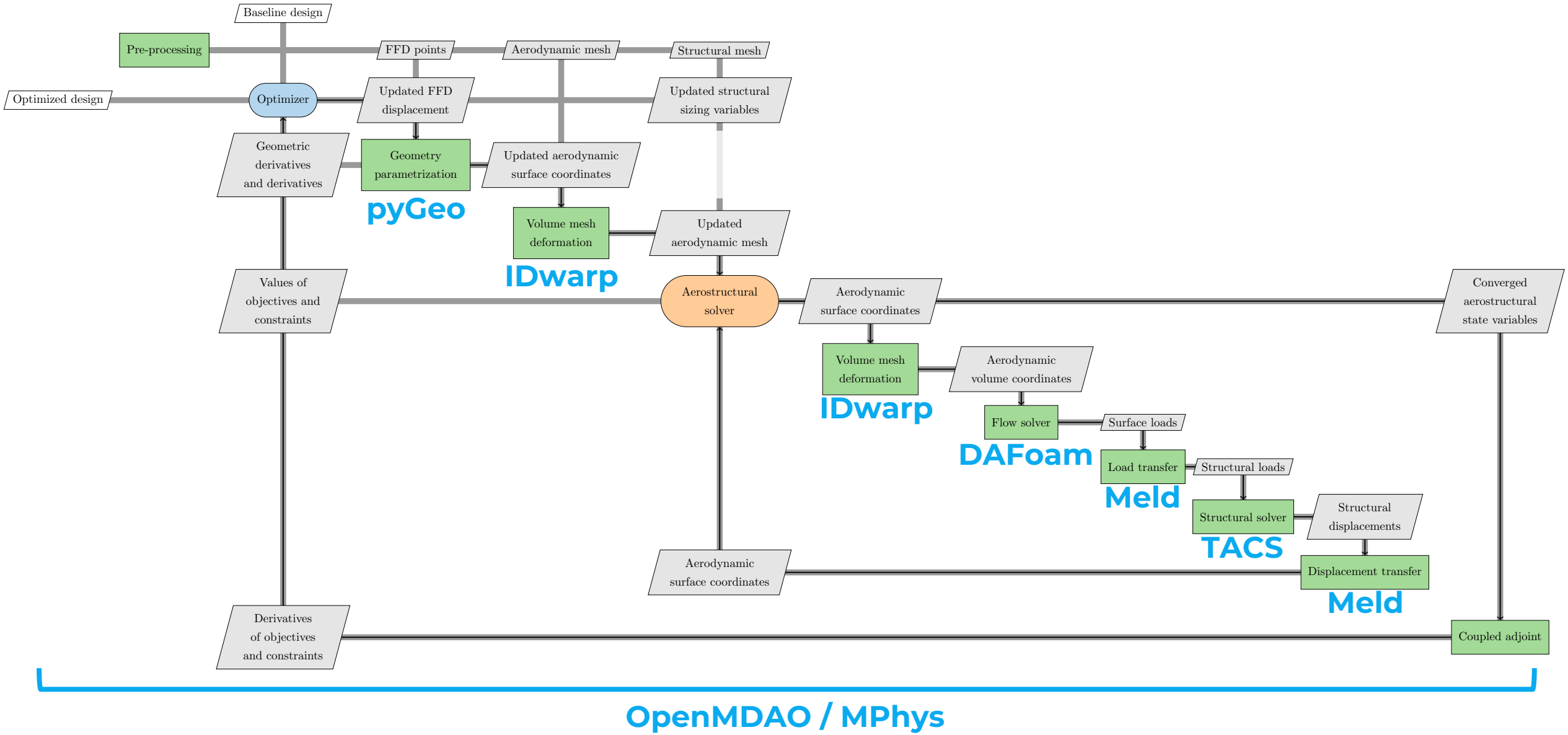


Optimization Subsystems – Aerostructural Coupling

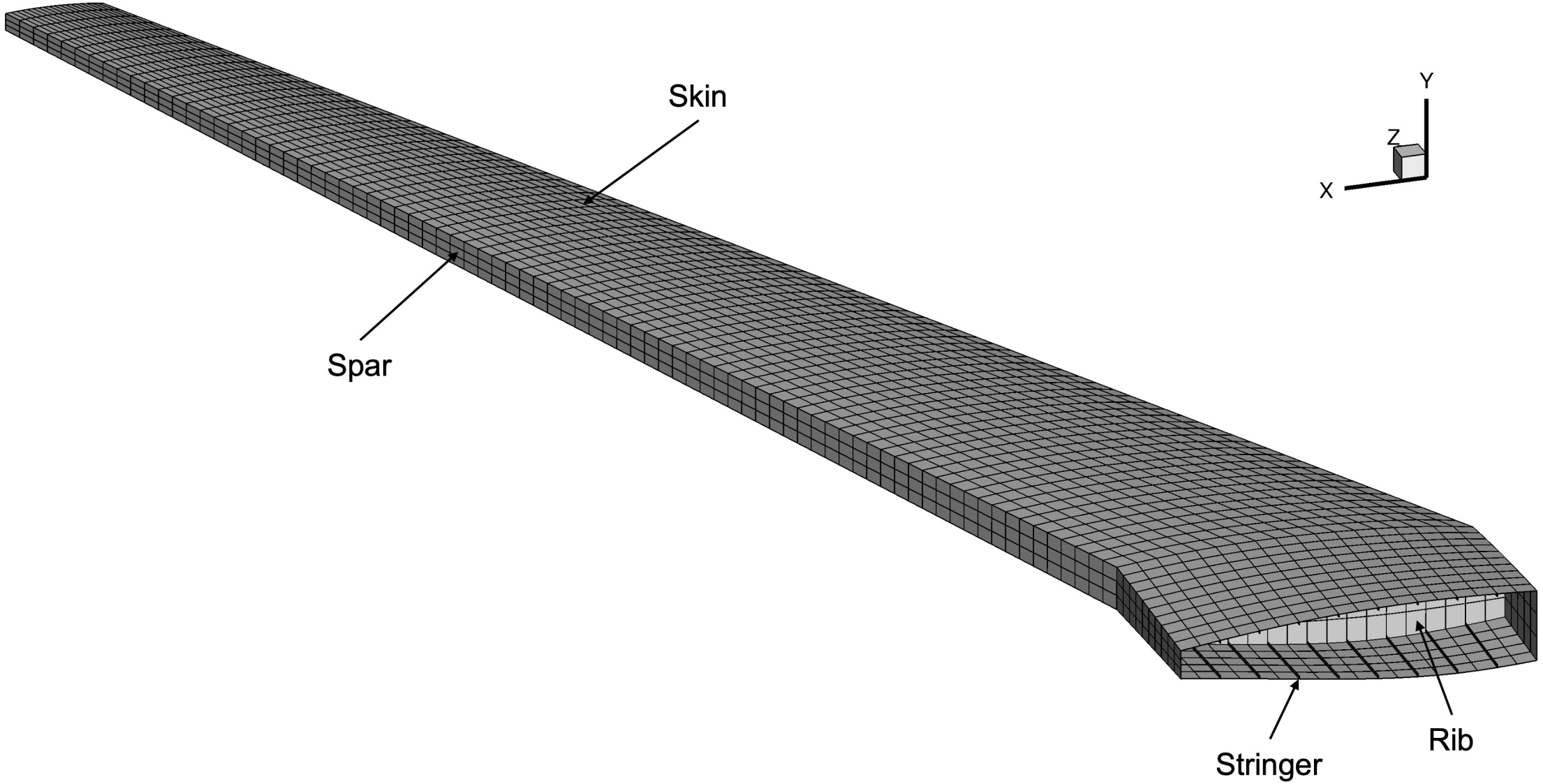


Slide adapted from: *Martins et al, Multidisciplinary Design Optimization of Engineering Systems*

Aerodynamic Shape Optimization



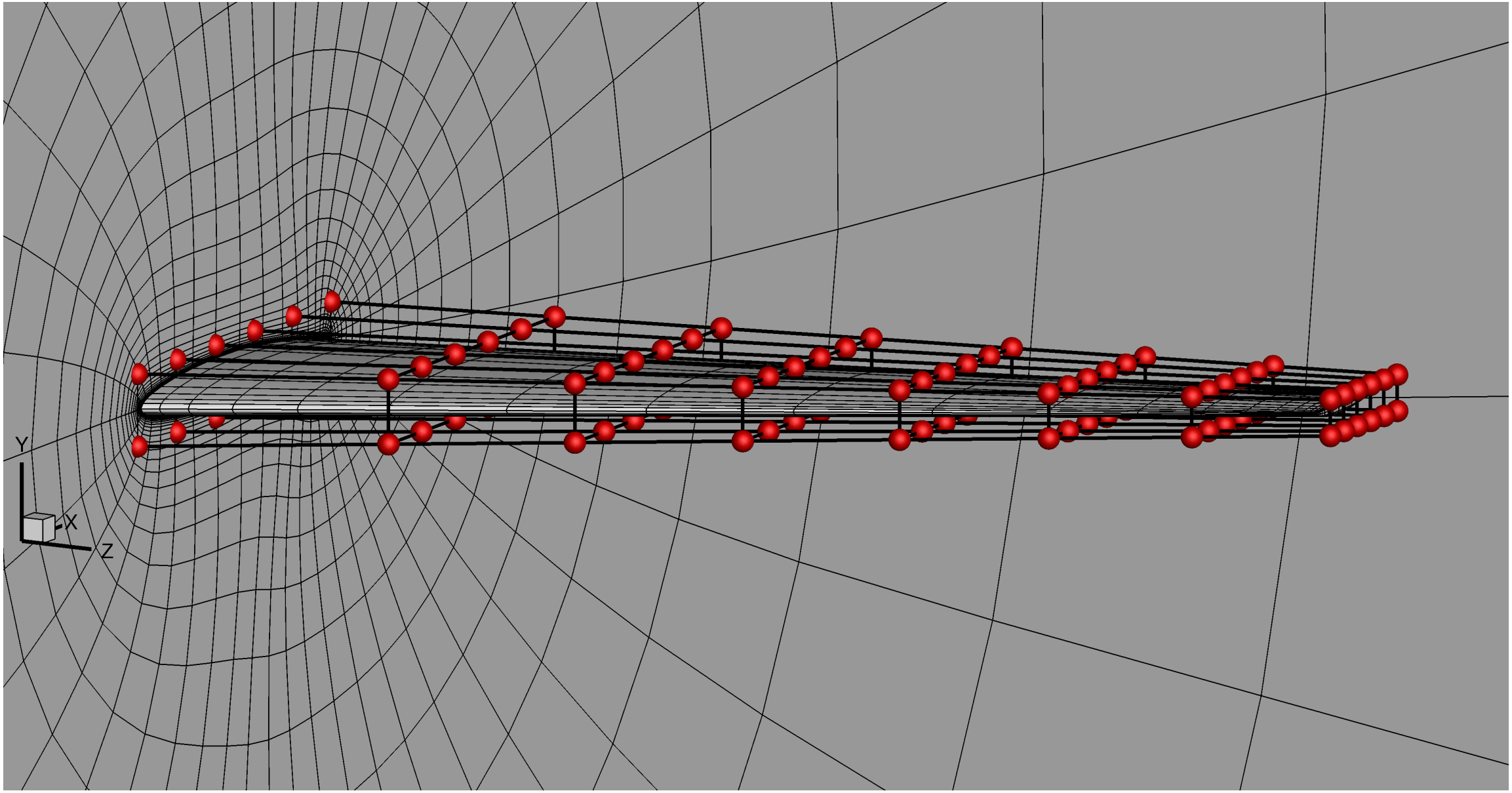
Aerodynamic Shape Optimization



Wing Aerostructural Optimization Problem Statement

	Function or Variable	Unit	Description	Quantity
Minimize	C_D	-	Drag coefficient	1
			Total objectives	1
With respect to	$0 \leq \alpha < 10$	deg	Angle of attack	1
	$-10 \leq \gamma \leq 10$	deg	Twist of each FFD section	7
	$-1 \leq \Delta z \leq 1$	m	Vertical displacement of FFD points	64
			Total design variables	72
Subject to	$C_L = 0.5$	-	Lift coefficient	1
	$KS(VM_{failure}) \leq 1.0$	-	Material failure constraint	1
	$0.5 \cdot t_{bl} \leq \mathrm{t} \leq 3 \cdot t_{bl}$	m	Thickness constraint	100
	$\mathrm{V}_{bl} \leq \mathrm{V}$	m^3	Volume constraint	1
	$\Delta z_{LE,upper} = \Delta z_{LE,lower}$	m	Fixed leading edge constraint	7
	$\Delta z_{TE,upper} = \Delta z_{TE,lower}$	m	Fixed trailing edge constraint	7
			Total constraints	117

Wing Geometry Parametrization



Runscript – Package Imports and Runtime Arguments

```python
import os
import argparse
import numpy as np
from mpi4py import MPI
import openmdao.api as om
from mphys.multipoint import Multipoint
from mphys.scenario_aerostructural import ScenarioAeroStructural
from dafoam.mphys import DAFoamBuilder, OptFuncs
from tacs.mphys import TacsBuilder
from funtofem.mphys import MeldBuilder
from pygeo.mphys import OM_DVGEOCOMP
from pygeo import geo_utils

import tacsSetup

parser = argparse.ArgumentParser()
# which optimizer to use. Options are: IPOPT (default), SLSQP, and SNOPT
parser.add_argument("-optimizer", help="optimizer to use", type=str, default="IPOPT")
# which task to run. Options are: opt (default), runPrimal, runAdjoint, checkTotals
parser.add_argument("-task", help="type of run to do", type=str, default="opt")
args = parser.parse_args()
```

Coupling framework

DAFoam

Structural solver

Load and displacement transfer

Geometry engine

Structural solver configuration file

Runtime arguments

Runscript – Options Dictionary OpenFOAM Setup

```
U0 = 100.0
p0 = 101325.0
nuTilda0 = 4.5e-5
T0 = 300.0
CL_target = 0.5
aoa0 = 2.0
rho0 = p0 / T0 / 287.0
A0 = 45.5

daOptions = {
    "designSurfaces": ["wing"],
    "solverName": "DARhoSimpleFoam",
    "primalMinResTol": 1.0e-8,
    "couplingInfo": {
        "aerostructural": {"active": True, "pRef": p0, "propMovement": False, ...
                           "couplingSurfaceGroups": {"wingGroup": ["wing"]}}
    }, # set the ref pressure for computing force for FSI
    "primalBC": {
        "U0": {"variable": "U", "patches": ["inout"], "value": [U0, 0.0, 0.0]},
        "p0": {"variable": "p", "patches": ["inout"], "value": [p0]},
        "T0": {"variable": "T", "patches": ["inout"], "value": [T0]},
        "nuTilda0": {"variable": "nuTilda", "patches": ["inout"], "value": [nuTilda0]},
        "useWallFunction": True,
    },
```

Case input values

Design surface

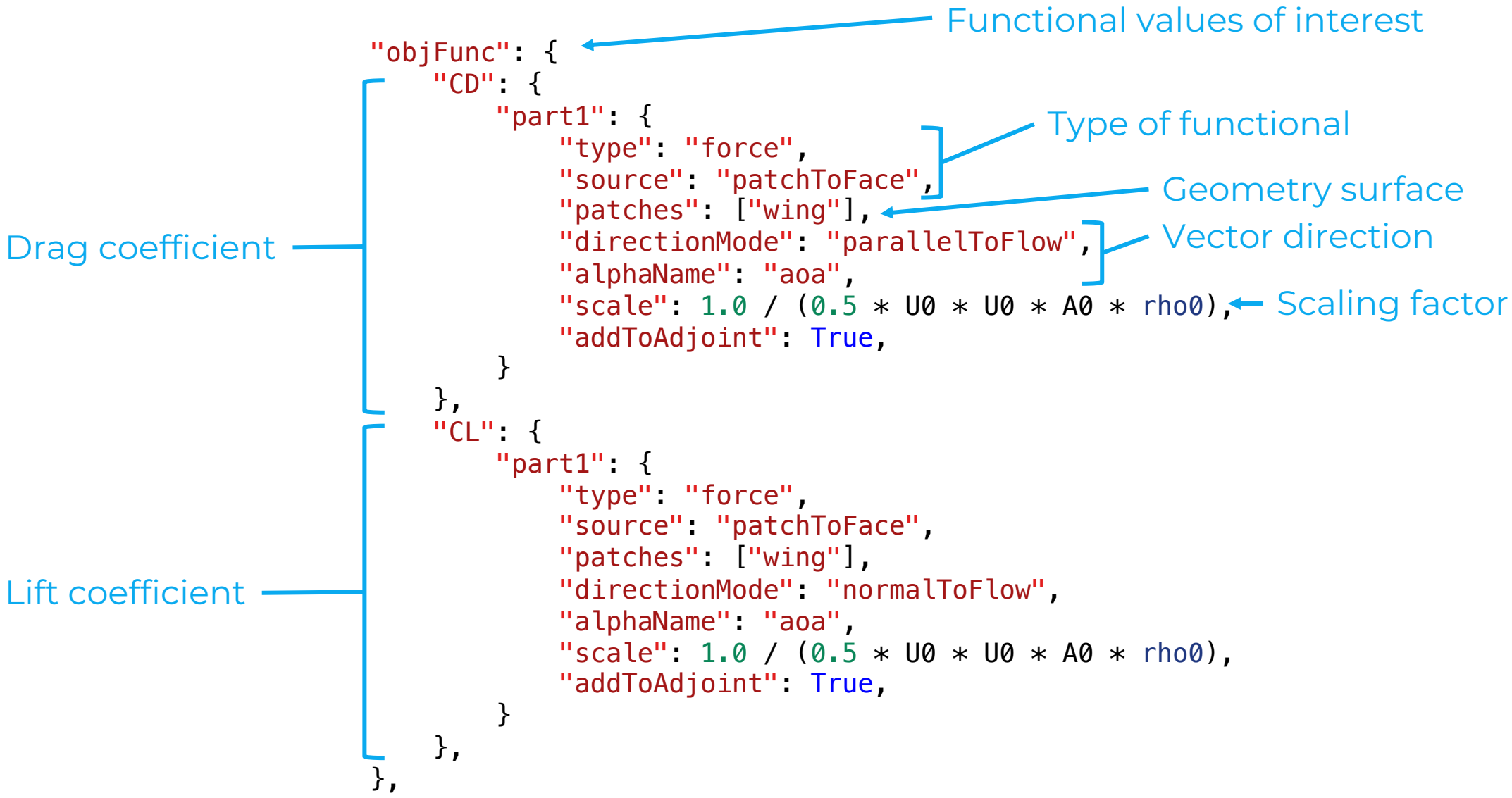
OpenFOAM solver

Flow convergence tolerance

Aerostructural coupling data

Boundary conditions

Runscript – Options Dictionary Functionals




```
"objFunc": {
    "CD": {
        "part1": {
            "type": "force",
            "source": "patchToFace",
            "patches": ["wing"],
            "directionMode": "parallelToFlow",
            "alphaName": "aoa",
            "scale": 1.0 / (0.5 * U0 * U0 * A0 * rho0),
            "addToAdjoint": True,
        }
    },
    "CL": {
        "part1": {
            "type": "force",
            "source": "patchToFace",
            "patches": ["wing"],
            "directionMode": "normalToFlow",
            "alphaName": "aoa",
            "scale": 1.0 / (0.5 * U0 * U0 * A0 * rho0),
            "addToAdjoint": True,
        }
    },
},
```

Runscript – Options Dictionary Derivatives

```
    "adjEqnOption": {
        "gmresRelTol": 1.0e-2, # set relative tolerance for block Gauss-Seidel adjoint
        "pcFillLevel": 1,
        "jacMatReOrdering": "rcm",
        "useNonZeroInitGuess": True,
    },
    "normalizeStates": {
        "U": U0,
        "p": p0,
        "T": T0,
        "nuTilda": 1e-3,
        "phi": 1.0,
    },
    "checkMeshThreshold": {
        "maxAspectRatio": 1000.0,
        "maxNonOrth": 70.0,
        "maxSkewness": 5.0,
    },
    "designVar": {
        "aoa": {"designVarType": "AOA", "patches": ["inout"], "flowAxis": "x", "normalAxis": "y"},
        "twist": {"designVarType": "FFD"},
        "shape": {"designVarType": "FFD"},
    },
}
```

Adjoint solver options

Adjoint state normalization factors

Mesh quality check thresholds

Design variable dictionary

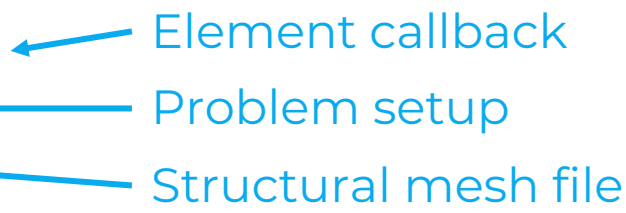
End of options dictionary

Runscript – Mesh Warping Setup

```
# Mesh deformation setup
meshOptions = {
    "gridFile": os.getcwd(),
    "fileType": "OpenFOAM",
    # point and normal for the symmetry plane
    "symmetryPlanes": [[[0.0, 0.0, 0.0], [0.0, 0.0, 1.0]]],
}
```

Runscript – Mesh Warping Setup

```
# TACS Setup
tacsOptions = {
    "element_callback": tacsSetup.element_callback,
    "problem_setup": tacsSetup.problem_setup,
    "mesh_file": "./wingbox.bdf",
}
```



- *Element callback:* sets up each element type read in through the BDF file, assigning it to a constitutive class
- *Problem setup:* defines the finite element model, design variables, and functionals of interest
- *Mesh file:* a NASTRAN BDF file consisting of a finite element mesh

Runscript – Setup Components

```
class Top(Multipoint):
    def setup(self):
        # create the builder to initialize the DASolvers
        aero_builder = DAFoamBuilder(daOptions, meshOptions, scenario="aerostructural")
        aero_builder.initialize(self.comm)

        # add the aerodynamic mesh component
        self.add_subsystem("mesh_aero", aero_builder.get_mesh_coordinate_subsystem())

        # create the builder to initialize TACS
        struct_builder = TacsBuilder(tacsOptions)
        struct_builder.initialize(self.comm)

        # add the structure mesh component
        self.add_subsystem("mesh_struct", struct_builder.get_mesh_coordinate_subsystem())

        # load and displacement transfer builder (meld), isym sets the symmetry plan axis (k)
        xfer_builder = MeldBuilder(aero_builder, struct_builder, isym=2, check_partials=True)
        xfer_builder.initialize(self.comm)

        # add the design variable component to keep the top level design variables
        dvs = self.add_subsystem("dvs", om.IndepVarComp(), promotes=["*"])

        # add the geometry component (FFD)
        self.add_subsystem("geometry", OM_DVGEOCOMP(file="FFD/wingFFD.xyz", type="ffd"))
        .
        .
        .
```

Initialize DAFoam

Aerodynamic mesh handler

Initialize TACS

Structural mesh handler

Initialize load / displacement transfer

Design variable manager

Initialize geometry engine

Runscript – Setup Coupling

```
class Top(Multipoint):
    def setup(self):
        .
        .
        .
        # primal and adjoint solution options, i.e., nonlinear block Gauss-Seidel for aerostructural analysis
        # and linear block Gauss-Seidel for the coupled adjoint
        nonlinear_solver = om.NonlinearBlockGS(maxiter=25, iprint=2, use_aitken=True, rtol=1e-8, atol=1e-8)
        linear_solver = om.LinearBlockGS(maxiter=25, iprint=2, use_aitken=True, rtol=1e-6, atol=1e-6)
        # add the coupling aerostructural scenario
        self.mphys_add_scenario(
            "cruise",
            ScenarioAeroStructural(
                aero_builder=aero_builder, struct_builder=struct_builder, ldxfer_builder=xfer_builder),
            nonlinear_solver,
            linear_solver,
        )
        .
        .
        .
```

Setup flight scenario

Declare analysis and derivative coupling solvers

Runscript – Setup Connections

```
class Top(Multipoint):
    def setup(self):
        .
        .
        .
        # need to manually connect the vars in the geo component to cruise
        for discipline in ["aero"]:
            self.connect("geometry.x_%s0" % discipline, "cruise.x_%s0_masked" % discipline)
        for discipline in ["struct"]:
            self.connect("geometry.x_%s0" % discipline, "cruise.x_%s0" % discipline)

        # add the structural thickness DVs
        ndv_struct = struct_builder.get_ndv()
        dvs.add_output("dv_struct", np.array(ndv_struct * [0.01]))
        self.connect("dv_struct", "cruise.dv_struct")

        # more manual connection
        self.connect("mesh_aero.x_aero0", "geometry.x_aero_in")
        self.connect("mesh_struct.x_struct0", "geometry.x_struct_in")
```

Setup geometry connections

Setup structural design variables

Setup mesh connections

Runscript – Configure Functionals and Design Surface

```
def configure(self):
    # call this to configure the coupling solver
    super().configure()

    # add the objective function to the cruise scenario
    self.cruise.aero_post.mphys_add_funcs()

    # get the surface coordinates from the mesh component
    points = self.mesh_aero.mphys_get_surface_mesh()

    # add pointset for both aero and struct
    self.geometry.nom_add_discipline_coords("aero", points)
    self.geometry.nom_add_discipline_coords("struct")

    # set the triangular points to the geometry component for geometric constraints
    tri_points = self.mesh_aero.mphys_get_triangulated_surface()
    self.geometry.nom_setConstraintSurface(tri_points)
    .
    .
    .
```

Execute default coupling group configuration

Propagate DAFoam functionals through model

Add DAFoam design surface mesh to geometry parametrization and constraint handler

Runscript – Configure Geometric Design Variables

```
def configure(self):
    .
    .
    .
    # Create reference axis for the twist variable
    nRefAxPts = self.geometry.nom_addRefAxis(name="wingAxis", xFraction=0.25, alignIndex="k")

    #  Set up global design variables. We dont change the root twist
    def twist(val, geo):
        for i in range(1, nRefAxPts):
        geo.rot_z["wingAxis"].coef[i] = -val[i - 1]

    # add twist variable
    self.geometry.nom_addGlobalDV(dvName="twist", value=np.array([0] * (nRefAxPts - 1)), func=twist)

    # add shape variable
    pts = self.geometry.DVGeo.getLocalIndex(0)
    indexList = pts[:, :, :].flatten()
    PS = geo_utils.PointSelect("list", indexList)
    nShapes = self.geometry.nom_addLocalDV(dvName="shape", pointSelect=PS)
    .
    .
    .
```

Define wing twist axis

Define wing twist function

Add twist design variable to geometry parametrization

Add shape design variables

Runscript – Configure Angle of Attack Design Variable

```
def configure(self):
    .
    .
    .
    # define an angle of attack function to change the U direction at the far field
    def aoa(val, DASolver):
        aoa = val[0] * np.pi / 180.0
        U = [float(U0 * np.cos(aoa)), float(U0 * np.sin(aoa)), 0]
        # we need to update the U value only
        DASolver.setOption("primalBC", {"U0": {"value": U}})
        DASolver.updateDAOption()

    # pass this aoa function to the cruise group
    self.cruise.coupling.aero.solver.add_dv_func("aoa", aoa)
    self.cruise.aero_post.add_dv_func("aoa", aoa)
    .
    .
    .
```

Define angle of attack function

Add angle of attack function to model

Runscript – Configure Constraints

```
def configure(self):
    .
    .
    .
    # setup the volume and thickness constraints
    leList = [[0.1, 0, 0.01], [7.5, 0, 13.9]]
    teList = [[4.9, 0, 0.01], [8.9, 0, 13.9]]
    self.geometry.nom_addThicknessConstraints2D("thickcon", leList, teList, nSpan=10, nChord=10)
    self.geometry.nom_addVolumeConstraint("volcon", leList, teList, nSpan=10, nChord=10)
    # add the LE/TE constraints
    self.geometry.nom_add_LETEConstraint("lecon", volID=0, faceID="iLow")
    self.geometry.nom_add_LETEConstraint("tecon", volID=0, faceID="iHigh")
    .
    .
    .
```

Define LE / TE lines

Add thickness constraint

Add volume constraint

Add LE / TE constraints

Runscript – Configure Design Variables, Objective, and Constraints

```
def configure(self):
    .
    .
    .
    # add the design variables to the dvs component's output
    self.dvs.add_output("twist", val=np.array([0] * (nRefAxPts - 1)))
    self.dvs.add_output("shape", val=np.array([0] * nShapes))
    self.dvs.add_output("aoa", val=np.array([aoa0]))
    # manually connect the dvs output to the geometry and cruise
    self.connect("twist", "geometry.twist")
    self.connect("shape", "geometry.shape")
    self.connect("aoa", "cruise.aoa")

    # define the design variables
    self.add_design_var("twist", lower=-10.0, upper=10.0, scaler=1.0)
    self.add_design_var("shape", lower=-1.0, upper=1.0, scaler=1.0)
    self.add_design_var("aoa", lower=0.0, upper=10.0, scaler=1.0)

    # add constraints and the objective
    self.add_objective("cruise.aero_post.CD", scaler=1.0)
    self.add_constraint("cruise.aero_post.CL", equals=0.3, scaler=1.0)
    # stress constraint
    self.add_constraint("cruise.ks_vmfailure", lower=0.0, upper=1.0, scaler=1.0)
    self.add_constraint("geometry.thickcon", lower=0.5, upper=3.0, scaler=1.0)
    self.add_constraint("geometry.volcon", lower=1.0, scaler=1.0)
    self.add_constraint("geometry.tecon", equals=0.0, scaler=1.0, linear=True)
    self.add_constraint("geometry.lecon", equals=0.0, scaler=1.0, linear=True)
```

Define independent variables and connect them

Add design variables to model

Add objective function to model

Add constraints to model

Runscript – Initialize Optimization Problem

```
# OpenMDAO setup
prob = om.Problem()
prob.model = Top()
prob.setup(mode="rev")
om.n2(prob, show_browser=False, outfile="mphys_aero_struct.html")

# initialize the optimization function
optFuncs = OptFuncs(daOptions, prob)
```

Initialize OpenMDAO problem

Setup DAFoam helper functions

Runscript – Setup Optimizer

```
# use pyoptsparse to setup optimization
prob.driver = om.pyOptSparseDriver()
prob.driver.options["optimizer"] = args.optimizer
# options for optimizers
if args.optimizer == "SNOPT":
    .
    .
    .
elif args.optimizer == "IPOPT":
    prob.driver.opt_settings = {
        "tol": 1.0e-5,
        "constr_viol_tol": 1.0e-5,
        "max_iter": 100,
        "print_level": 5,
        "output_file": "opt_IPOPT.txt",
        "mu_strategy": "adaptive",
        "limited_memory_max_history": 10,
        "nlp_scaling_method": "none",
        "alpha_for_y": "full",
        "recalc_y": "yes",
}
elif args.optimizer == "SLSQP":
    .
    .
    .
else:
print("optimizer arg not valid!")
exit(1)
```

Connect an optimizer

Declare optimizer options

Runscript – Task

```
prob.driver.options["debug_print"] = ["nl_cons", "objs", "desvars"]
prob.driver.options["print_opt_prob"] = True
prob.driver.hist_file = "OptView.hst"


if args.task == "opt":
    # solve CL
    optFuncs.findFeasibleDesign(["cruise.aero_post.CL"], ["aoa"], targets=[CL_target])
    # run the optimization
    prob.run_driver()
elif args.task == "runPrimal":
    # just run the primal once
    prob.run_model()
elif args.task == "runAdjoint":
    .
    .
    .
elif args.task == "checkTotals":
    .
    .
    .
else:
    print("task arg not found!")
    exit(1)
```

Set runtime log preferences

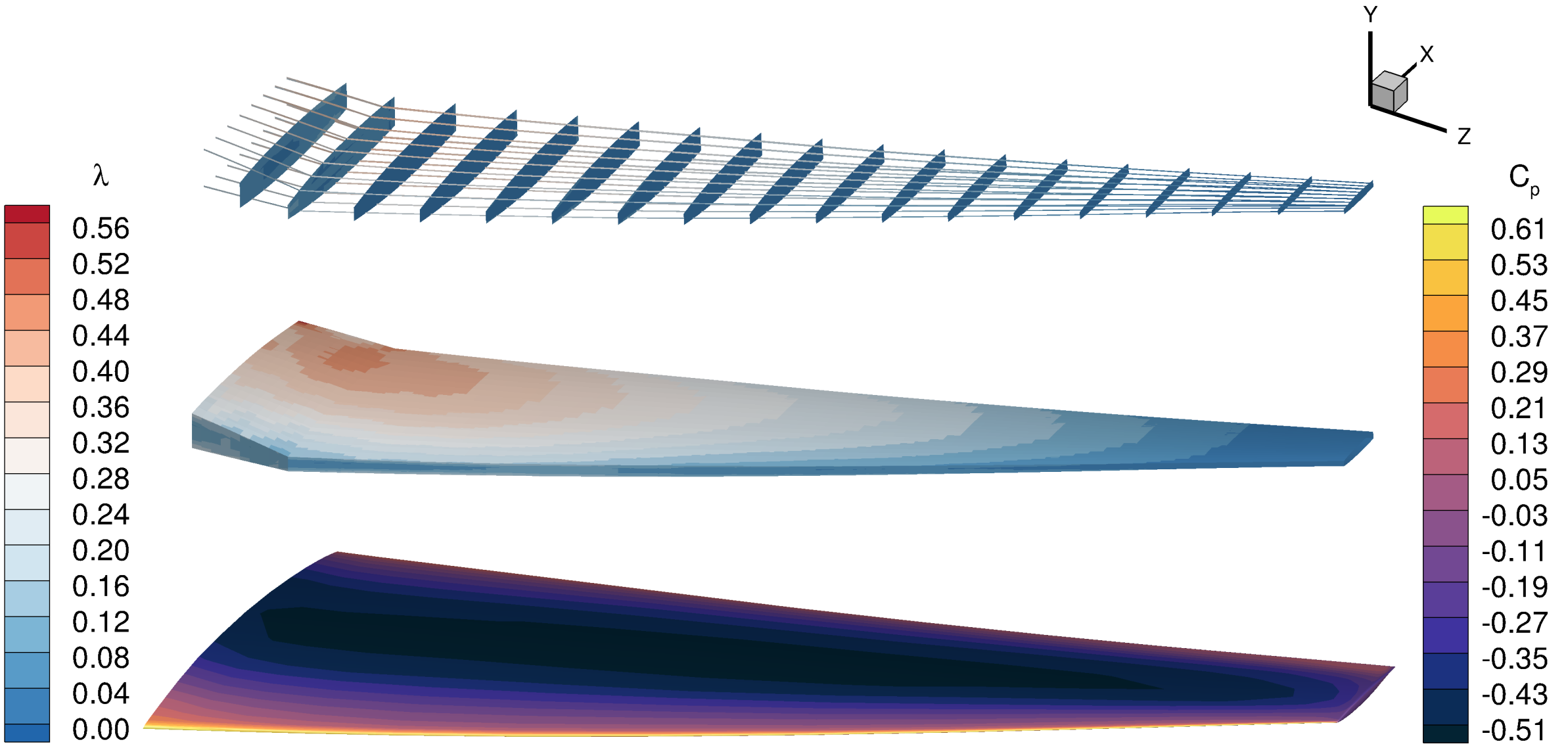
Trim baseline design

Optimize!

Run one analysis

Let’s run it!

Result

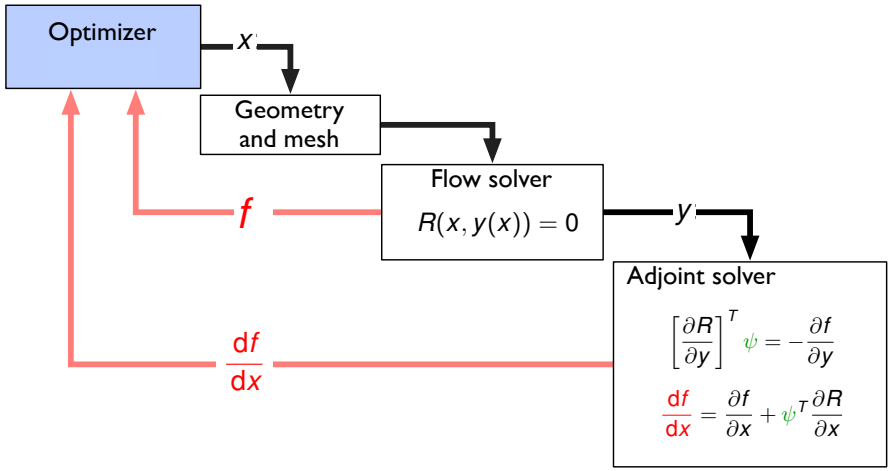

Y
X
Z
λ
0.56
0.52
0.48
0.44
0.40
0.36
0.32
0.28
0.24
0.20
0.16
0.12
0.08
0.04
0.00
Cp
0.61
0.53
0.45
0.37
0.29
0.21
0.13
0.05
-0.03
-0.11
-0.19
-0.27
-0.35
-0.43
-0.51

Review

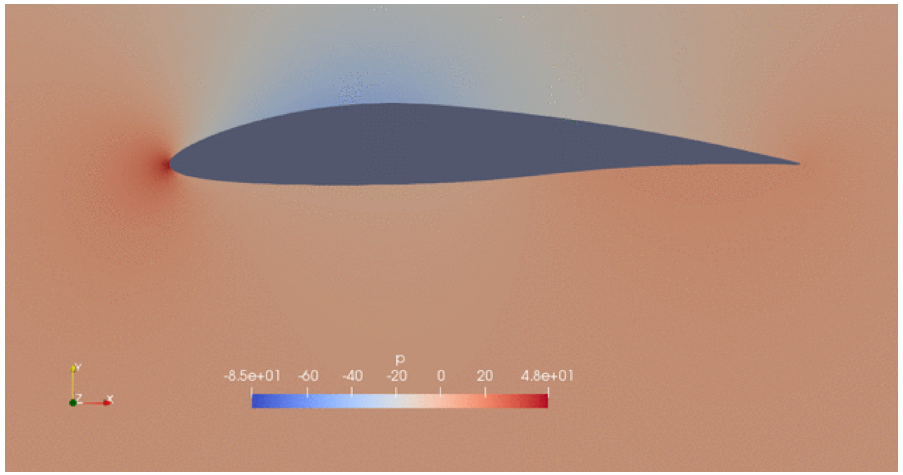
- Imported packages for each required component and coupling
- Declared the aerodynamic simulation parameters in the DAFoamOptions dictionary
- Declared the structural simulation parameters in the DAFoamOptions dictionary
- Initialized optimization components in the setup() method
- Assembled the optimization problem with the configure() method
- Created an OpenMDAO problem and connected our model
- Configured an optimizer
- Ran the optimization!

Recap

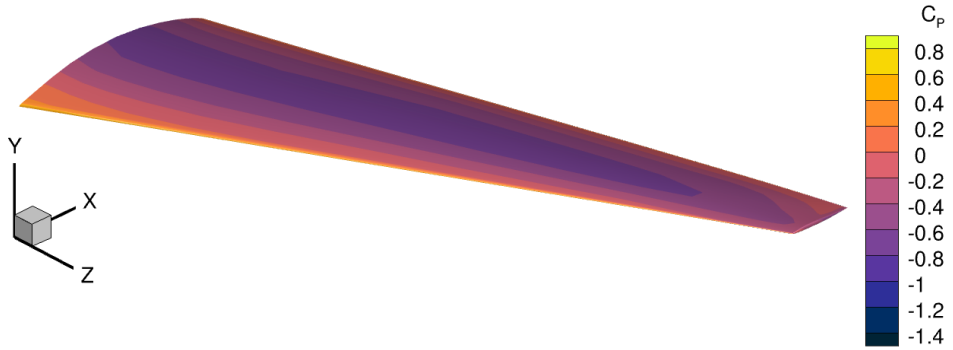
Theory



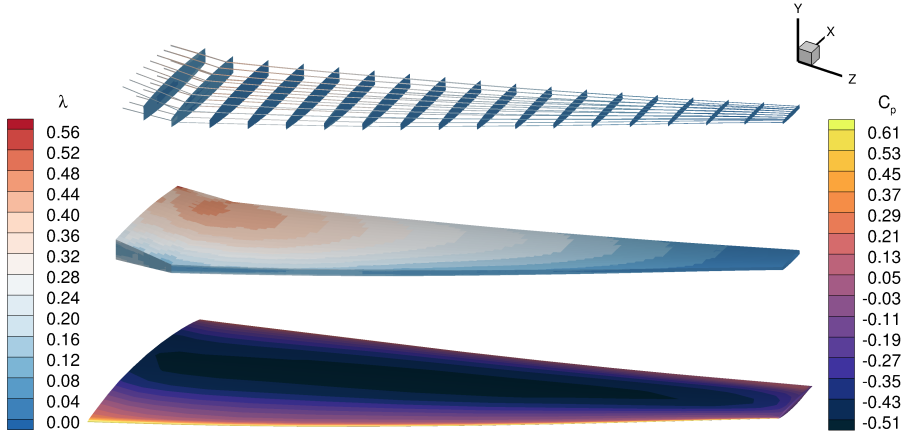
Airfoil Optimization



Wing Aerodynamic Optimization



Wing Aerostructural Optimization



Additional Resources

DAFoam: https://dafoam.github.io/index.html

MACH-Aero: https://github.com/mdolab/MACH-Aero

MPhys: https://github.com/OpenMDAO/mphys

OpenMDAO: https://openmdao.org

For additional DAFoam workshops, checkout: https://github.com/DAFoam/workshops