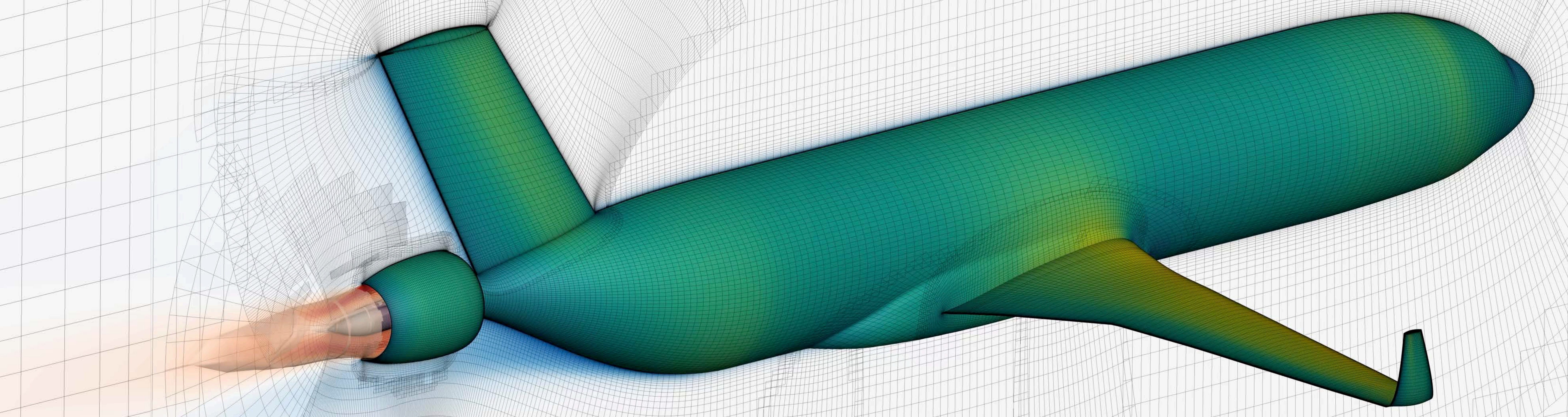
MDO with Coupled Adjoints: From the Unified Derivatives Equation to OpenMDAO



OpenMDAO Workshop—Now with more MPhys!

October 25, 2022

NASA Glenn Research Center • Ohio

Aerospace Engineering
University of Michigan

http://mdolab.engin.umich.edu

Joaquim R. R. A. Martins

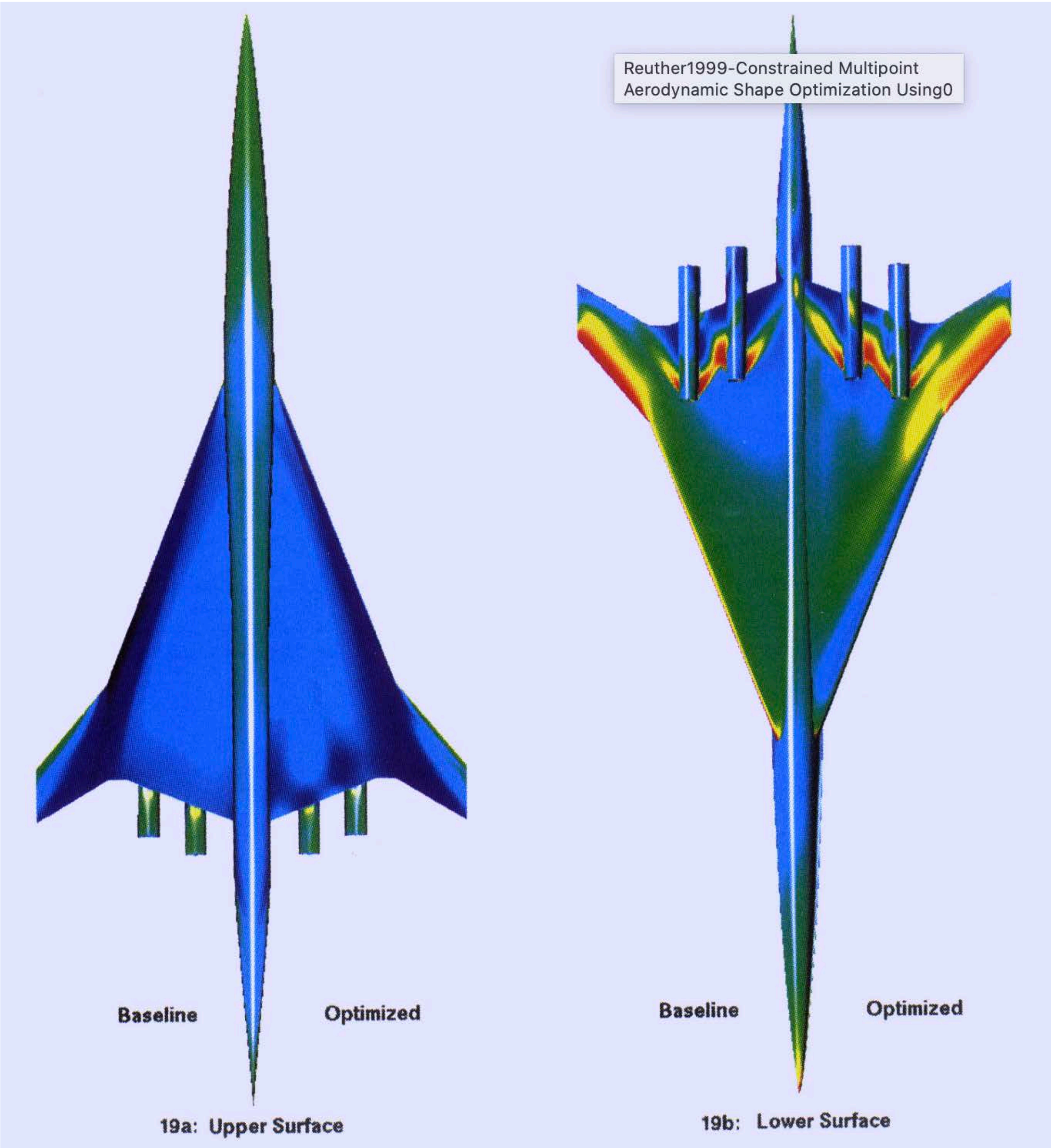
with contributions from Justin Gray, John Hwang, John Jasa, Andrew Ning, Bernardo Pacini, and Anil Yildirim


LET'S DO SOME ADJOINTS!


WHEN YOU
GET ADJOINT
SUPER
POWERS

For my PhD thesis, I decided to combine ideas from two research streams

Adjoint-based aerodynamic shape optimization



Reuther, Jameson, Alonso, Rimlinger, and Saunders. **Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers,** (parts 1 and 2)**.** *Journal of Aircraft*,1999.

MDO of aircraft configurations

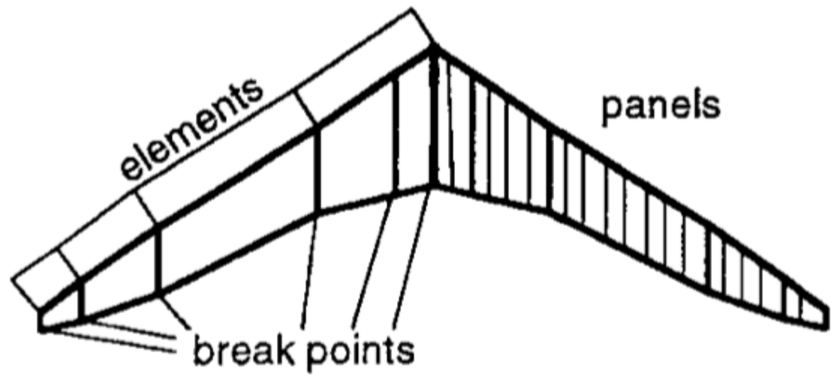


Fig. 1 Wing element and panel geometry.

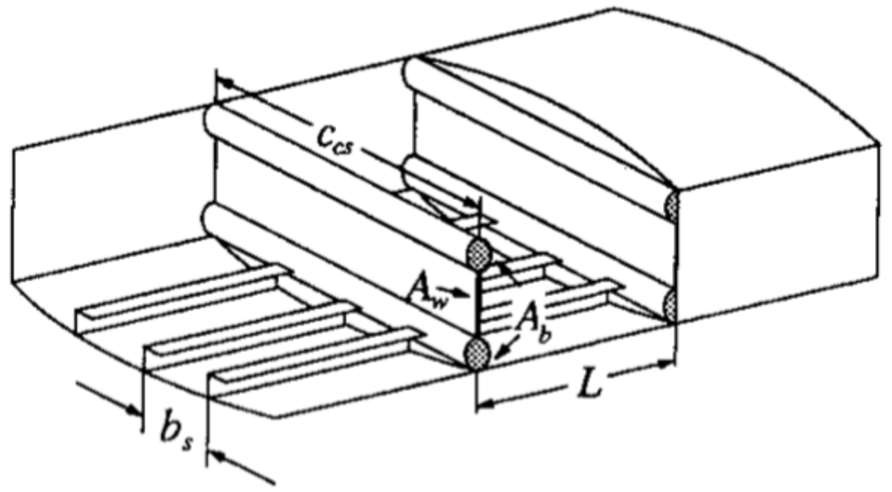
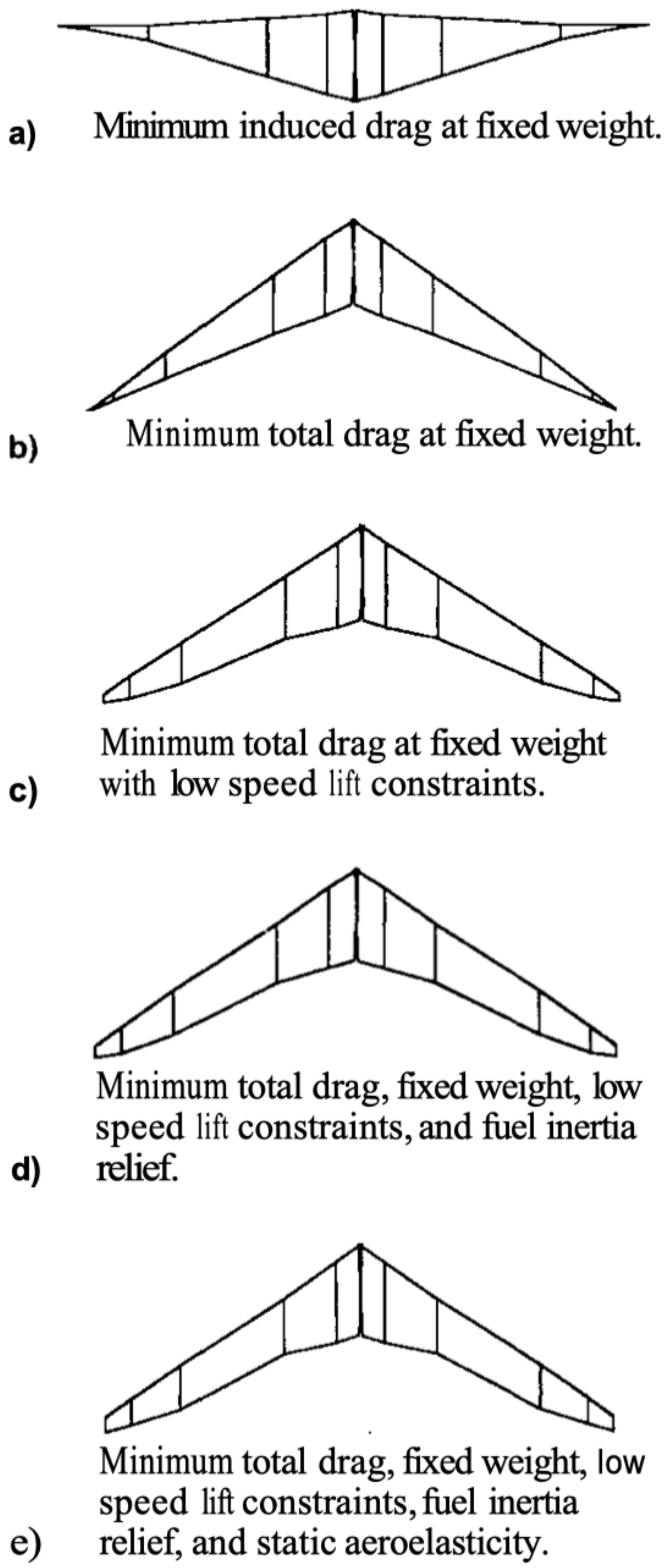
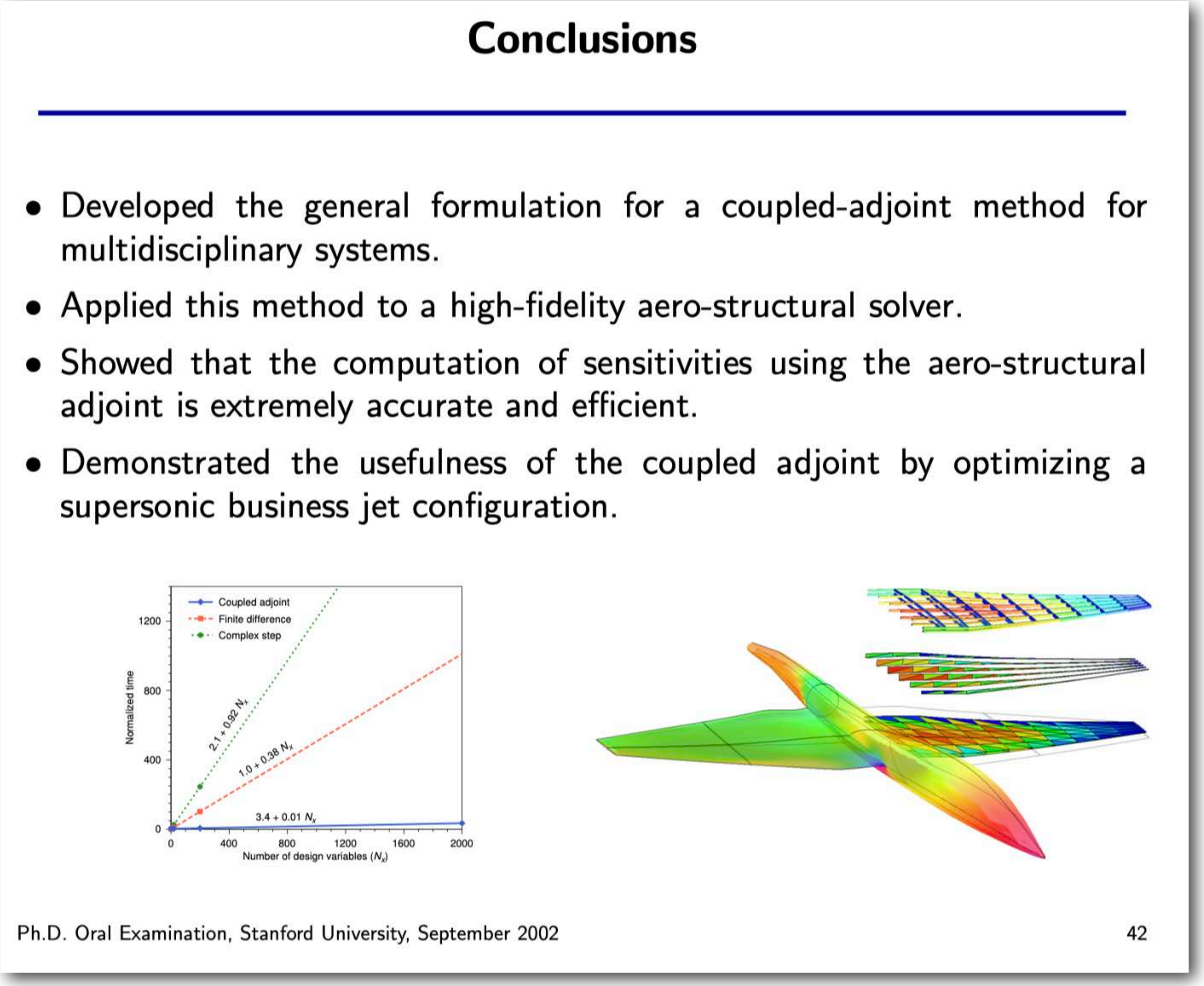


Fig. 2 Structural geometry.



Wakayama and Kroo. **Subsonic wing planform design using multidisciplinary optimization**. Journal of Aircraft,1995.

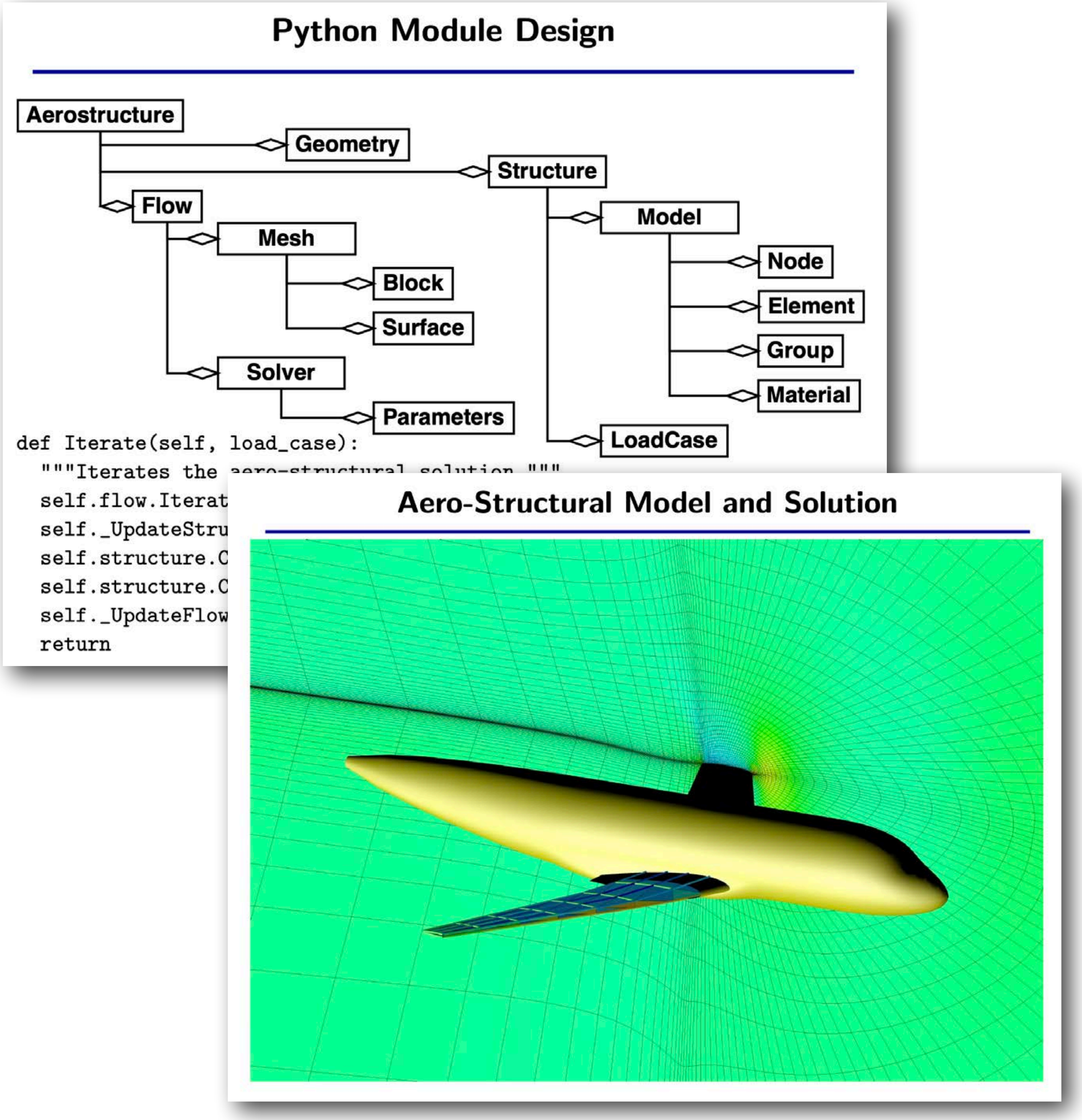
The first CFD-based aerostructural design optimization



Martins, Alonso, and Reuther. **High-fidelity aerostructural design optimization of a supersonic business jet**.
Journal of Aircraft, 2004

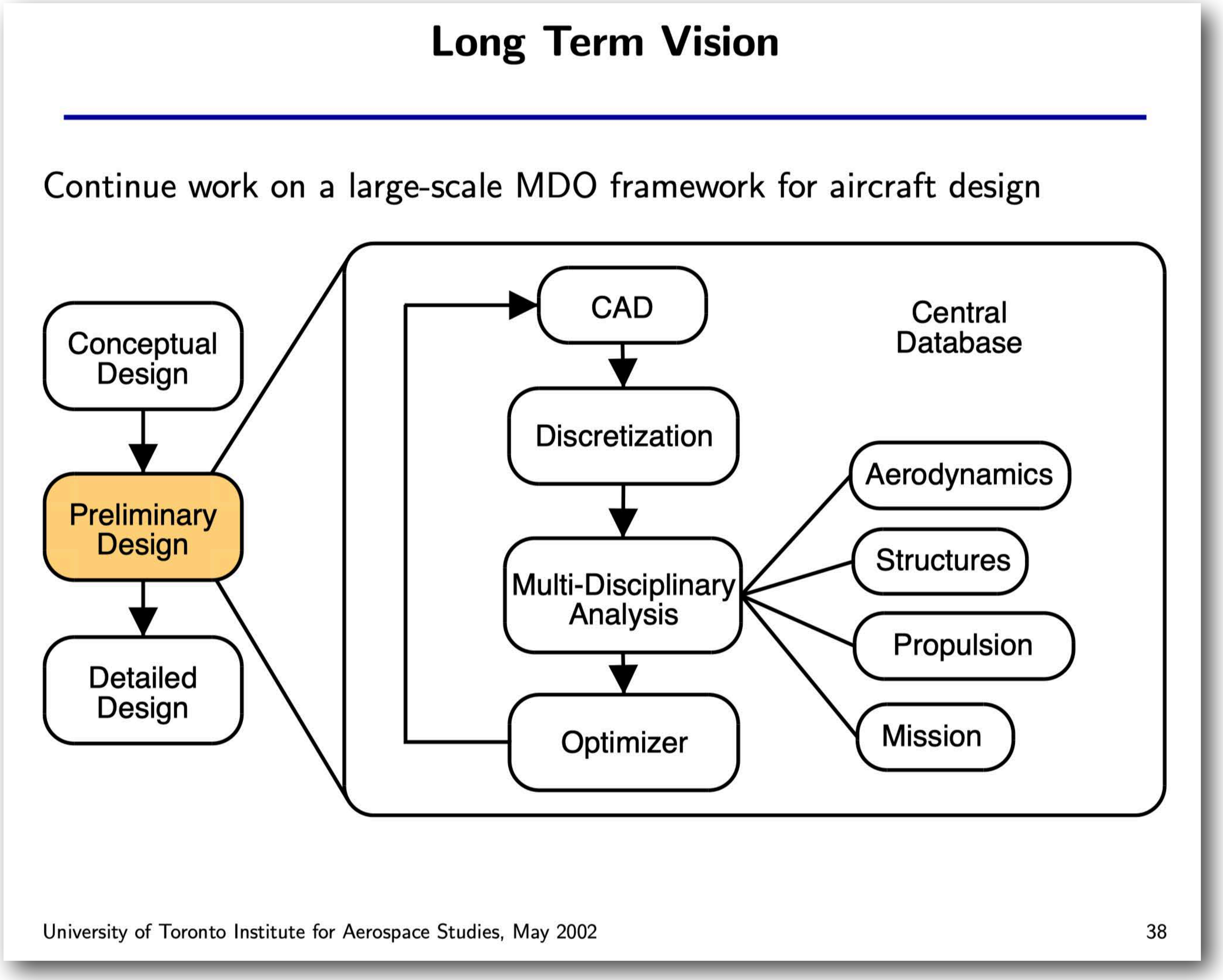
I selected Python to implement a second version of the aerostructural optimization framework

Discovered f2py



Peterson, Martins, and Alonso. **Fortran to Python interface generator with an application to aerospace engineering**. *Proceedings of the 9th International Python Conference*, 2001.

Planned Python-based framework



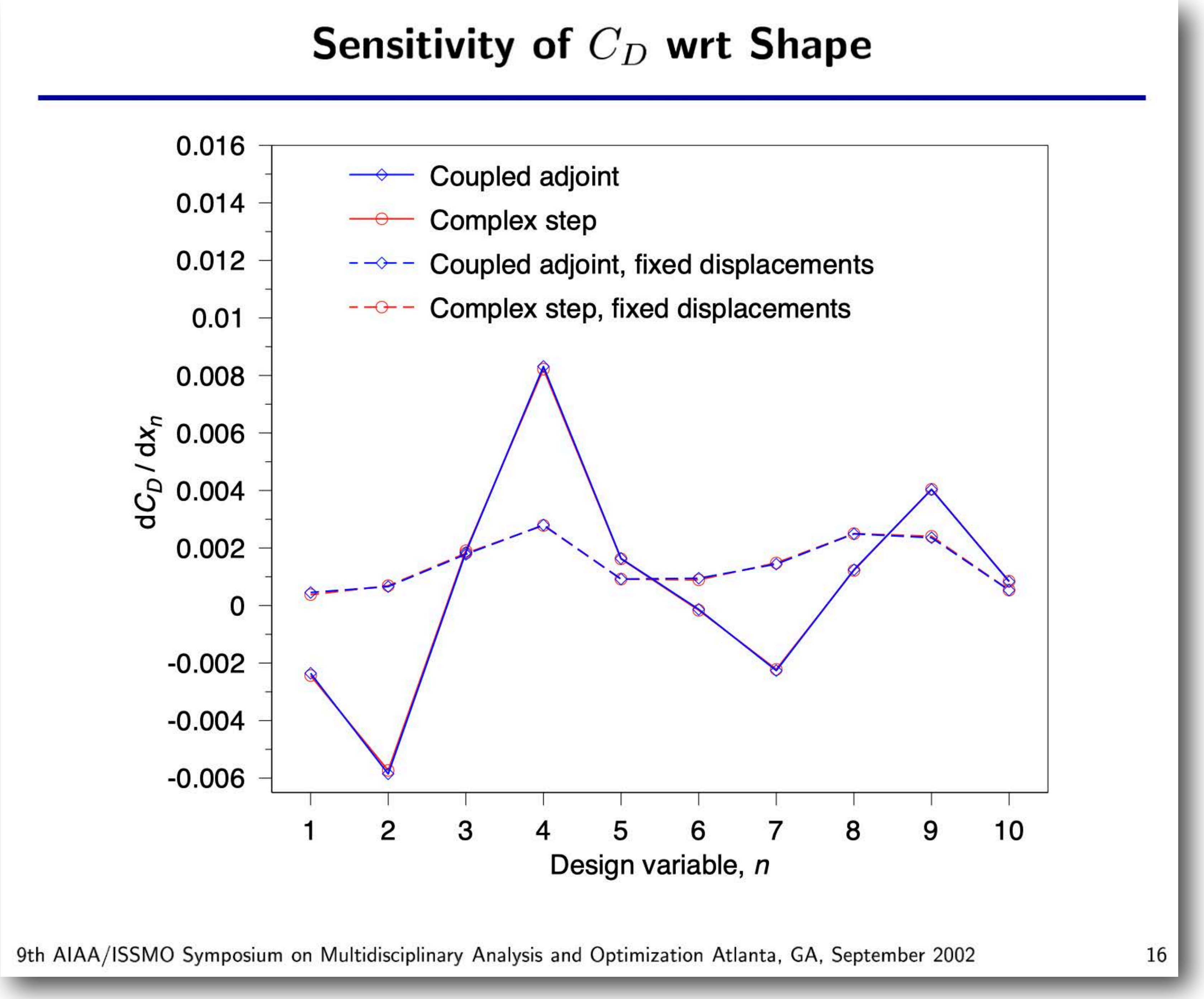
PhD defense and UTIAS interview, 2002

I also worked on the complex-step method, for which Python was really useful

Complex-step derivative approximation

```
#! /usr/local/bin/python

header_string = """
     ____________________________________
    |                                    |
    |          Complexify 1.3            |
    |       J.R.R.A.Martins 1999         |
    |     update July 00 P. Sturdza      |
    |                                    |
    |    f'(x) ~  Im [ f(x+ih) ] / h     |
    |____________________________________|

    Notes:
      1) Make sure you compile with -r8 flag
      2) Does not handle f90 free format or f77 tab-format files yet
      3) Make sure the main routine begins with 'PROGRAM'
      4) Use 132 column option in compiler
      4) Command line options:
         -lucky_logic  -- don't need fixing of .eq. and .ne.
         -MIPS_logic   -- bug in MIPS pro V7.3 reqiures .ge. fixed too
         -fudge_format -- dumb fix for format statements
"""

def main():
    global fix_relationals, fudge_format_statement
    bad = 0
    print header_string
    if not sys.argv[1:]: # No arguments
        err('usage: \n\t' + sys.argv[0]
            + ' [-lucky_logic|-MIPS_logic|-fudge_format] file-pattern \t\n' +
\
            '\tpython ' + sys.argv[0]
            + ' [-lucky_logic|-MIPS_logic|-fudge_format] file-pattern \n\n' )
      sys.exit(2)
    for arg in sys.argv[1:]:
        if arg == "-lucky_logic":
            # don't attempt to fix .eq. and .ne. (works on PGF90)
            fix_relationals = 0
            continue
        if arg == "-MIPS_logic":
            # cheap fix for MIPS Pro
            fix_relationals = 2
```

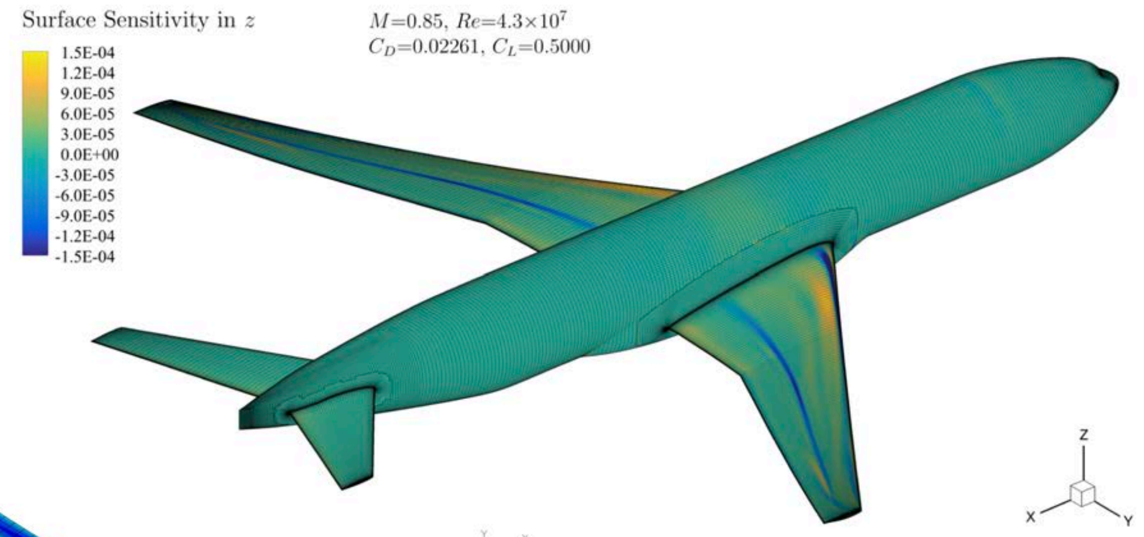
Verification of the coupled adjoint



Martins, Sturdza, and Alonso. **The complex-step derivative approximation**. *ACM Transactions on Mathematical Software*, 2003

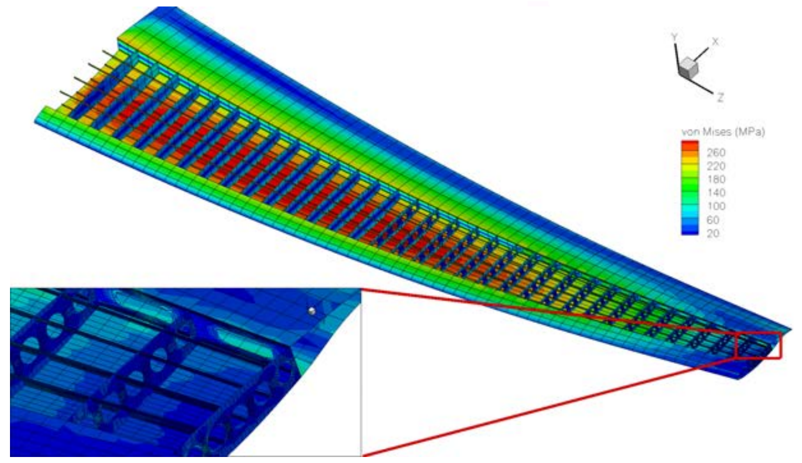
The MDO Lab developed MACH, a framework for aerostructural design optimization

- Aerodynamics



Kenway, Mader, He, and Martins. **Effective adjoint approaches for computational fluid dynamics**. *Progress in Aerospace Sciences*, 2019

- Structures



Kennedy and Martins. **A parallel finite-element framework for large-scale gradient-based design optimization of high-performance structures**. *Finite Elements in Analysis and Design*, 2014

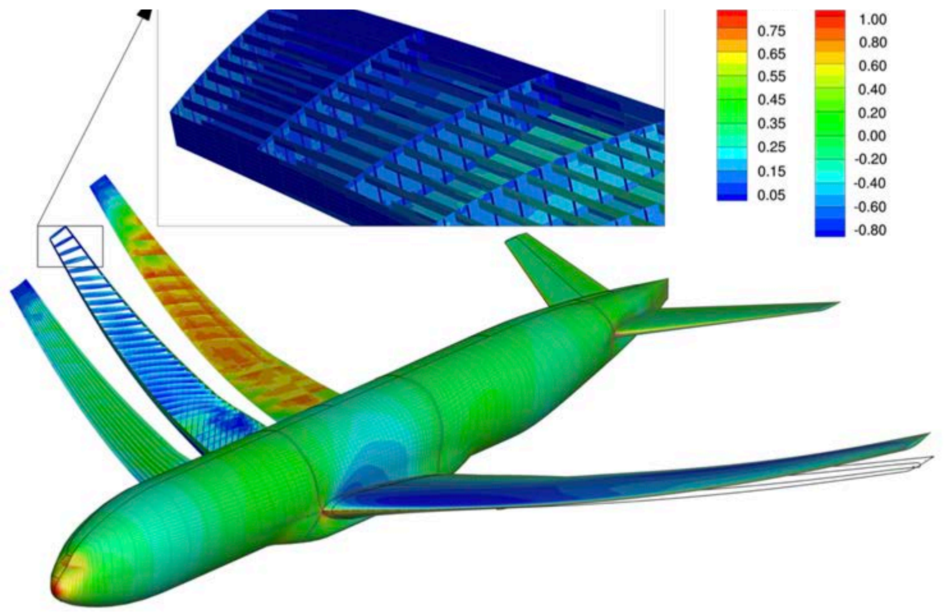
- Coupled adjoint

$$\left(\frac{\partial \mathcal{A}}{\partial w}\right)^T \psi^{(k)} = \left(\frac{\partial I}{\partial w}\right)^T - \left(\frac{\partial \mathcal{S}}{\partial w}\right)^T \phi^{(k-1)}$$

$$\left(\frac{\partial \mathcal{S}}{\partial u}\right)_K \phi^{(k)} = \left(\frac{\partial I}{\partial u}\right)^T - \left(\frac{\partial \mathcal{A}}{\partial u}\right)^T \psi^{(k)} - \left(\frac{\partial \mathcal{S}}{\partial u}\right)^T_F \phi^{(k-1)}$$

Kenway, Kennedy, and Martins. **Scalable parallel approach for high-fidelity steady-state aeroelastic analysis and derivative computations**. *AIAA Journal*, 2014

- First application



Kenway and Martins. **Multipoint high-fidelity aerostructural optimization of a transport aircraft configuration**. *Journal of Aircraft*, 2014

MACH took years of development and many students

Python user script
Setup up the problem: objective function, constraints, design variables, optimizer and solver options

Optimizer interface
pyOptSparse
Common interface to various optimization software

- SNOPT
- Other optimizers

Aerostructural solver
AeroStruct
Coupled solution methods and coupled derivative evaluation

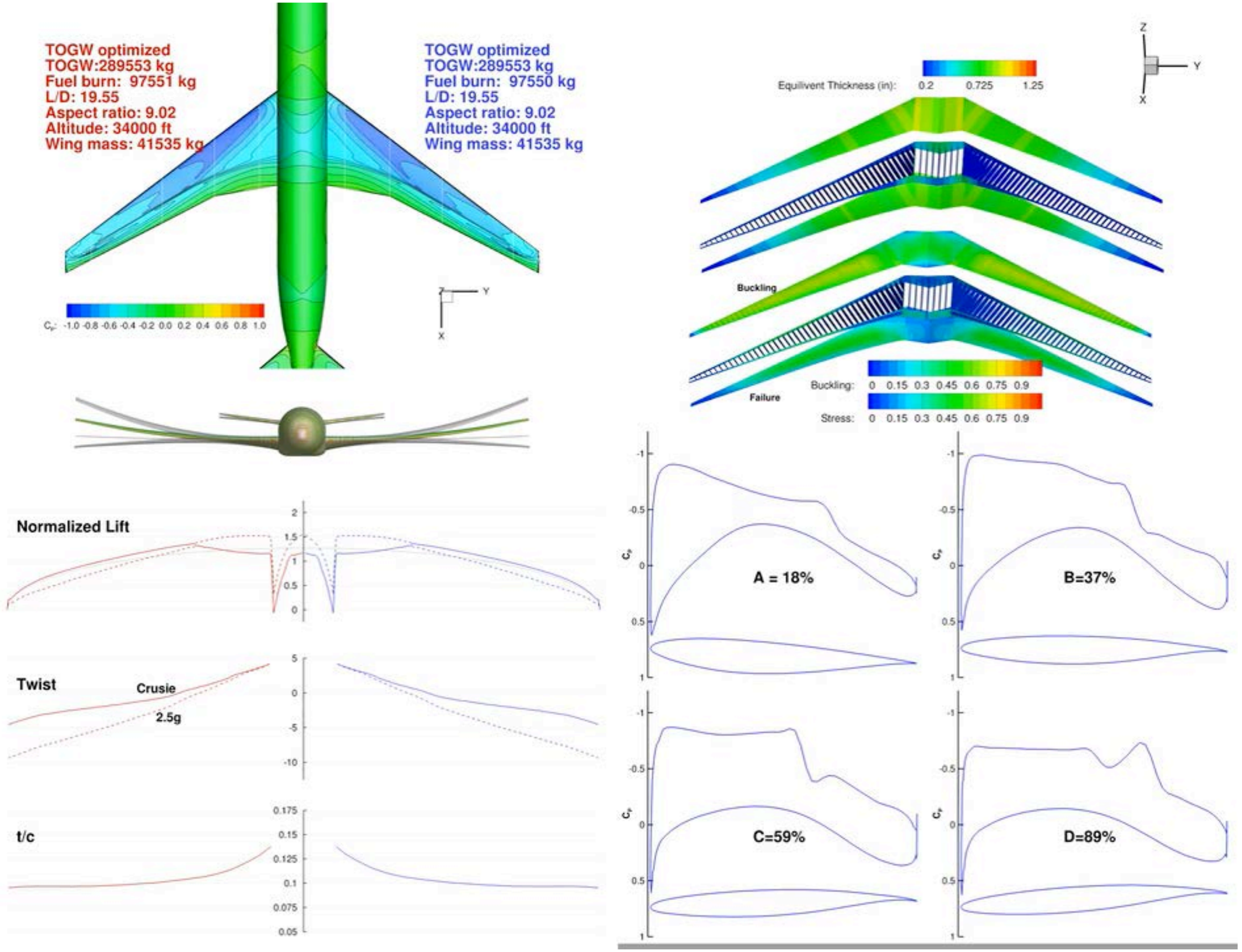
- Flow solver
  ADflow
  Governing and adjoint equations
- Structural solver
  TACS
  Governing and adjoint equations

Geometry modeler
DVGeometry/GeoMACH
Defines and manipulates geometry, evaluates derivatives

Kenway, Kennedy, and Martins. **Scalable parallel approach for high-fidelity steady-state aeroelastic analysis and derivative computations**. *AIAA Journal*, 2014

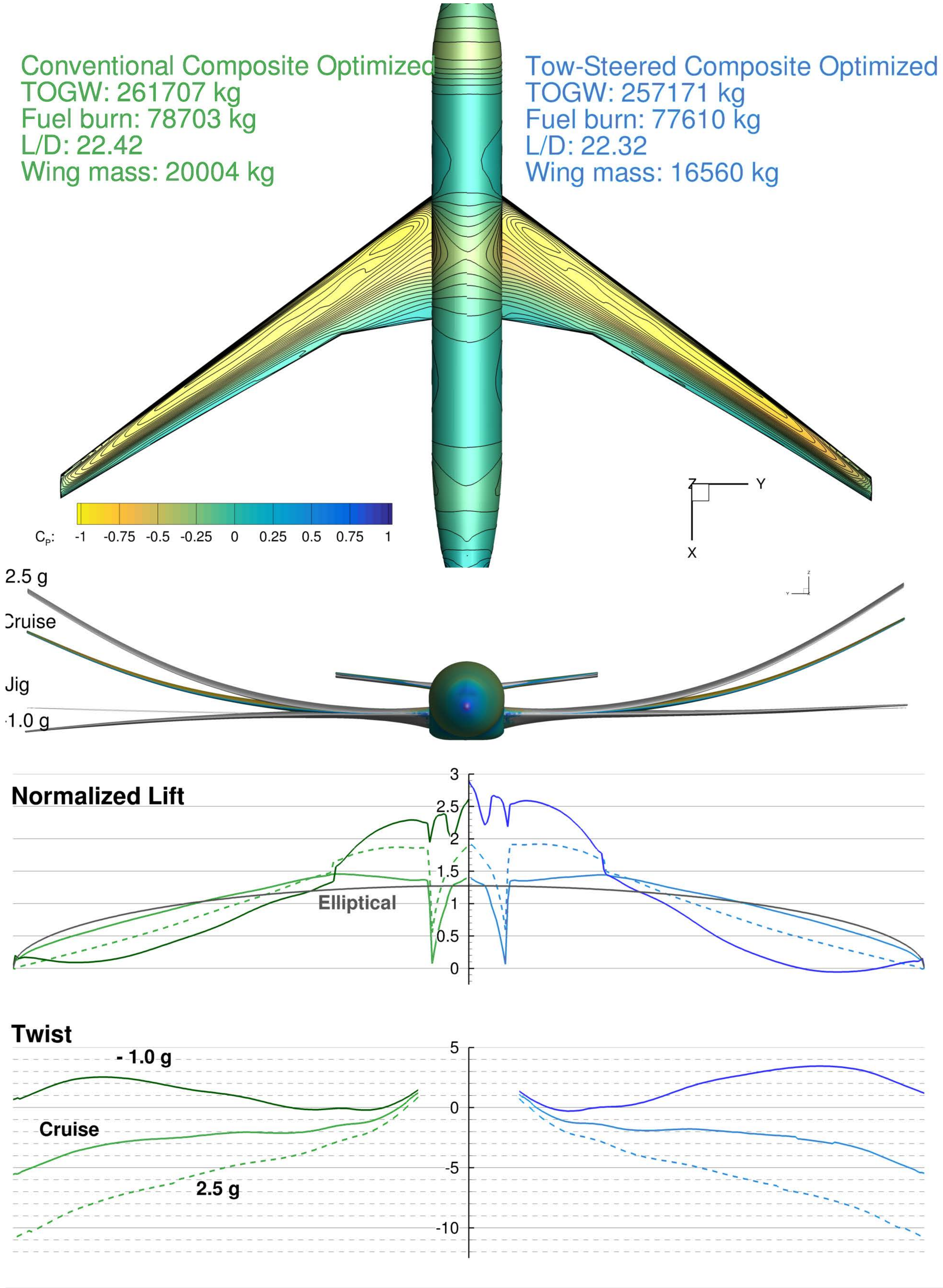
Kennedy and Martins. **A parallel finite-element framework for large-scale gradient-based design optimization of high-performance structures**. *Finite Elements in Analysis and Design*, 2014

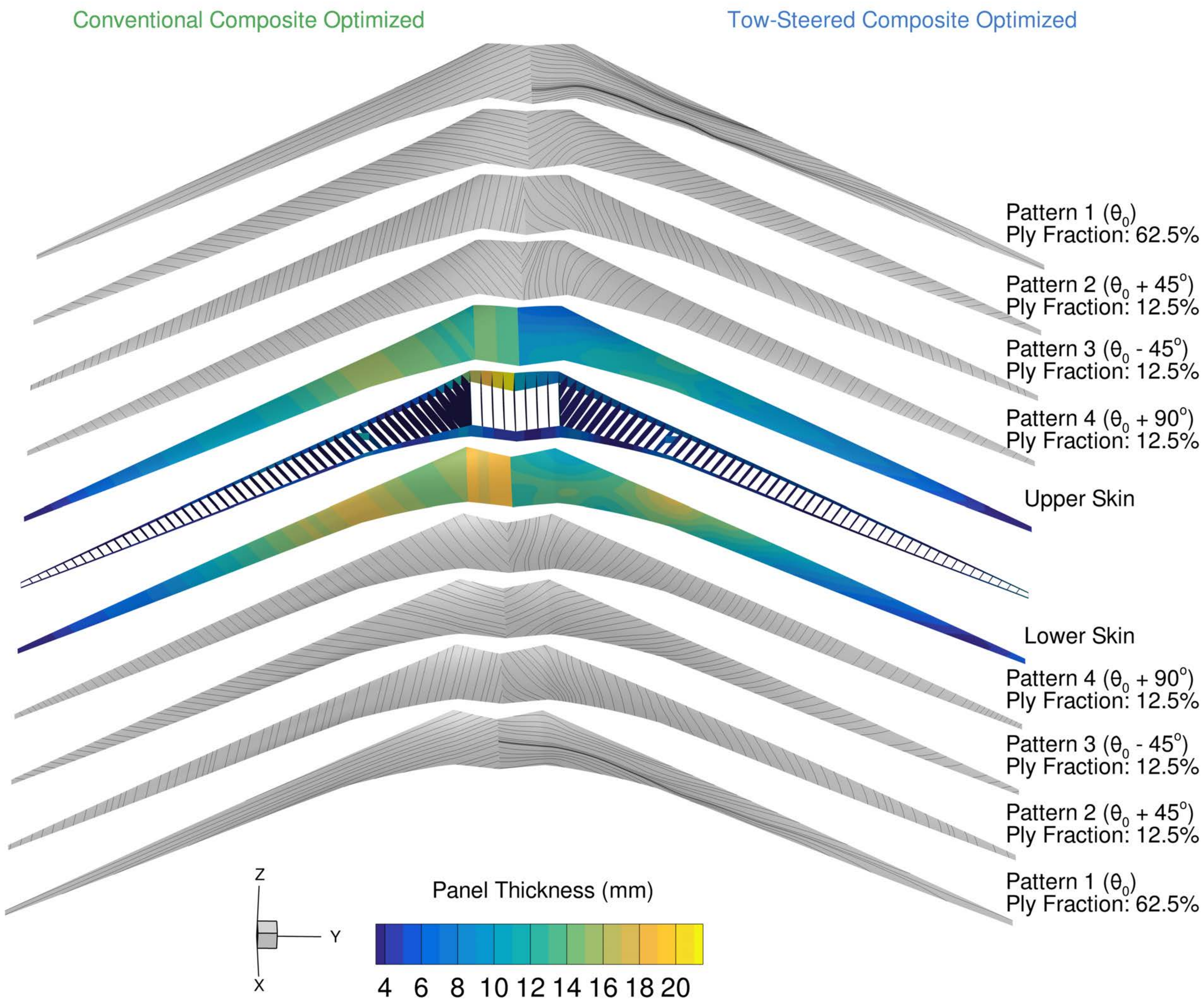
The coupled adjoint enabled us to perform high-fidelity aerostructural optimization (again)



Kenway and Martins. **Multipoint high-fidelity aerostructural optimization of a transport aircraft configuration**. *Journal of Aircraft*, 2014

Tow-steered composite high AR wing

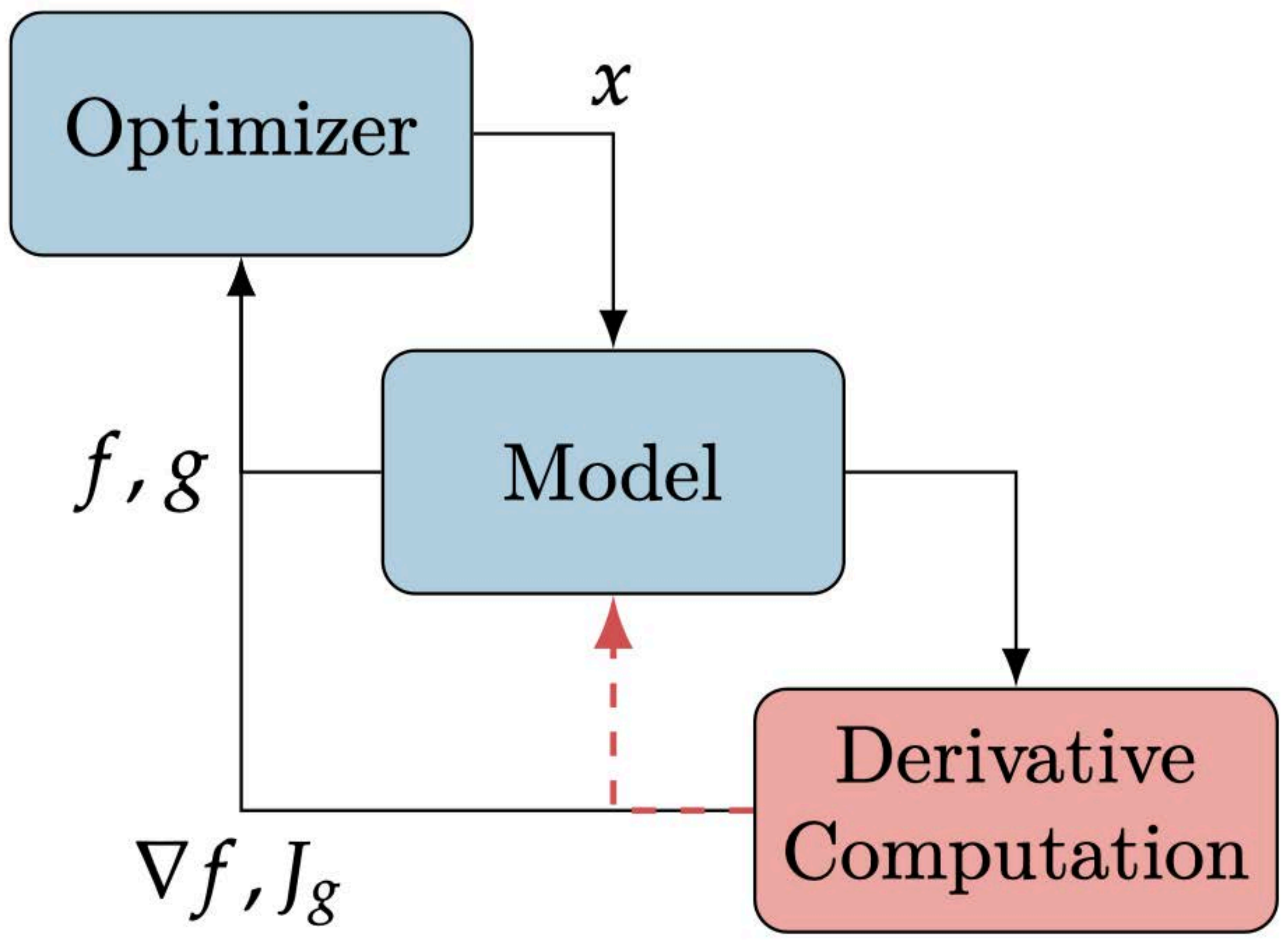


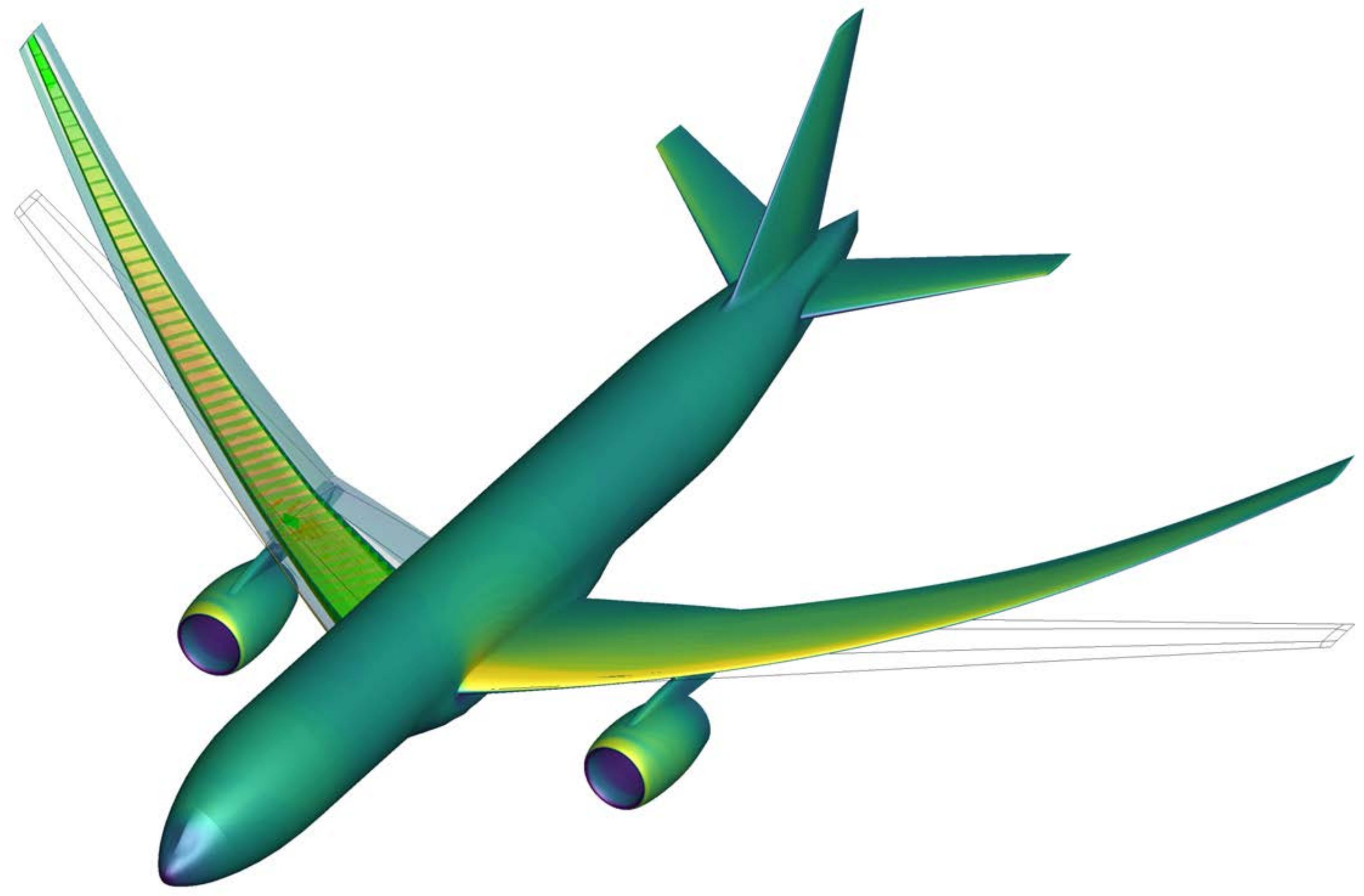


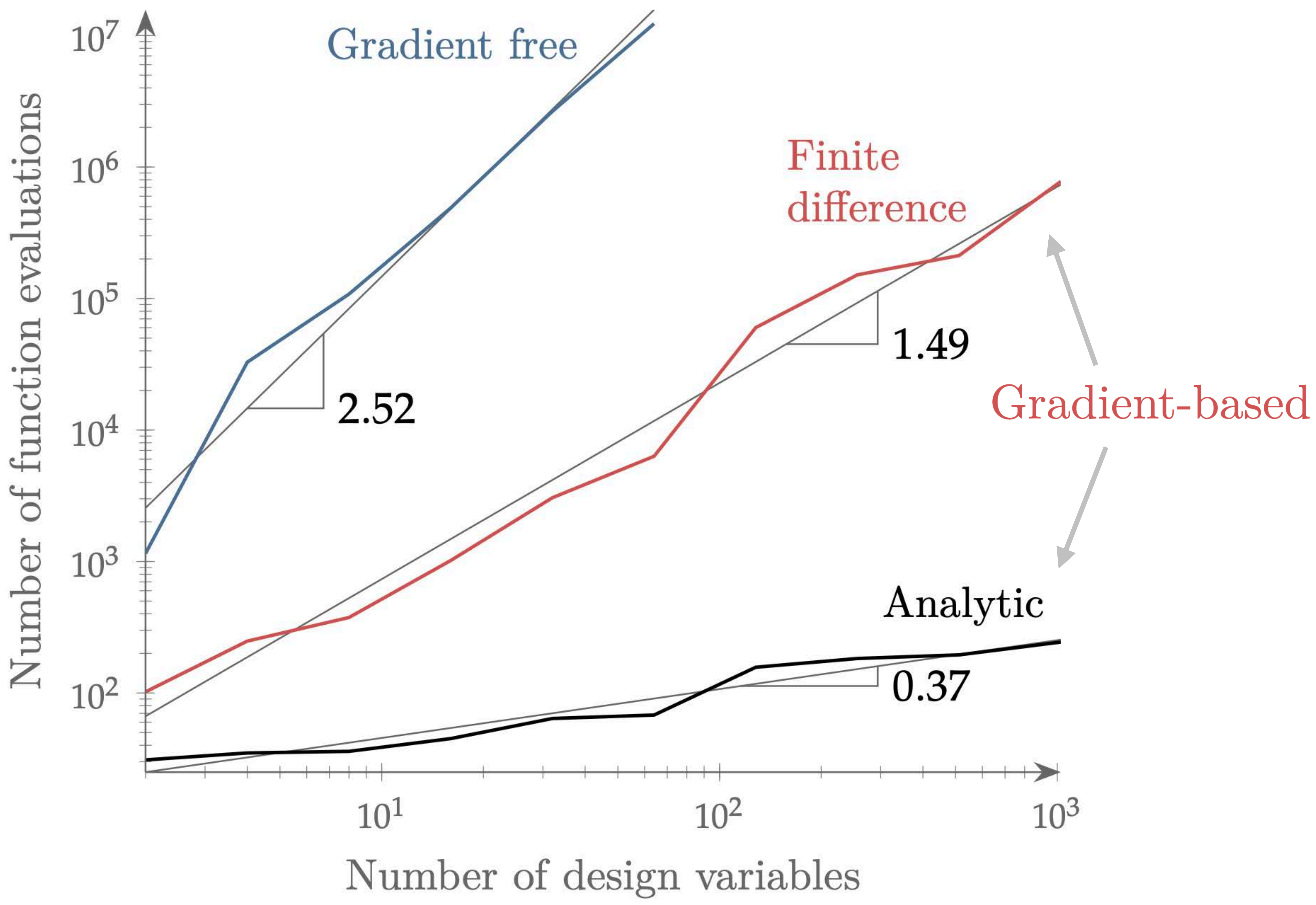
Brooks, Martins, and Kennedy. **High-fidelity aerostructural optimization of tow-steered composite wings**. *Journal of Fluids and Structures*, 2019

The wingbox was tested by NASA and is now on display here

Gradient-based optimization is the only hope for large numbers of design variables

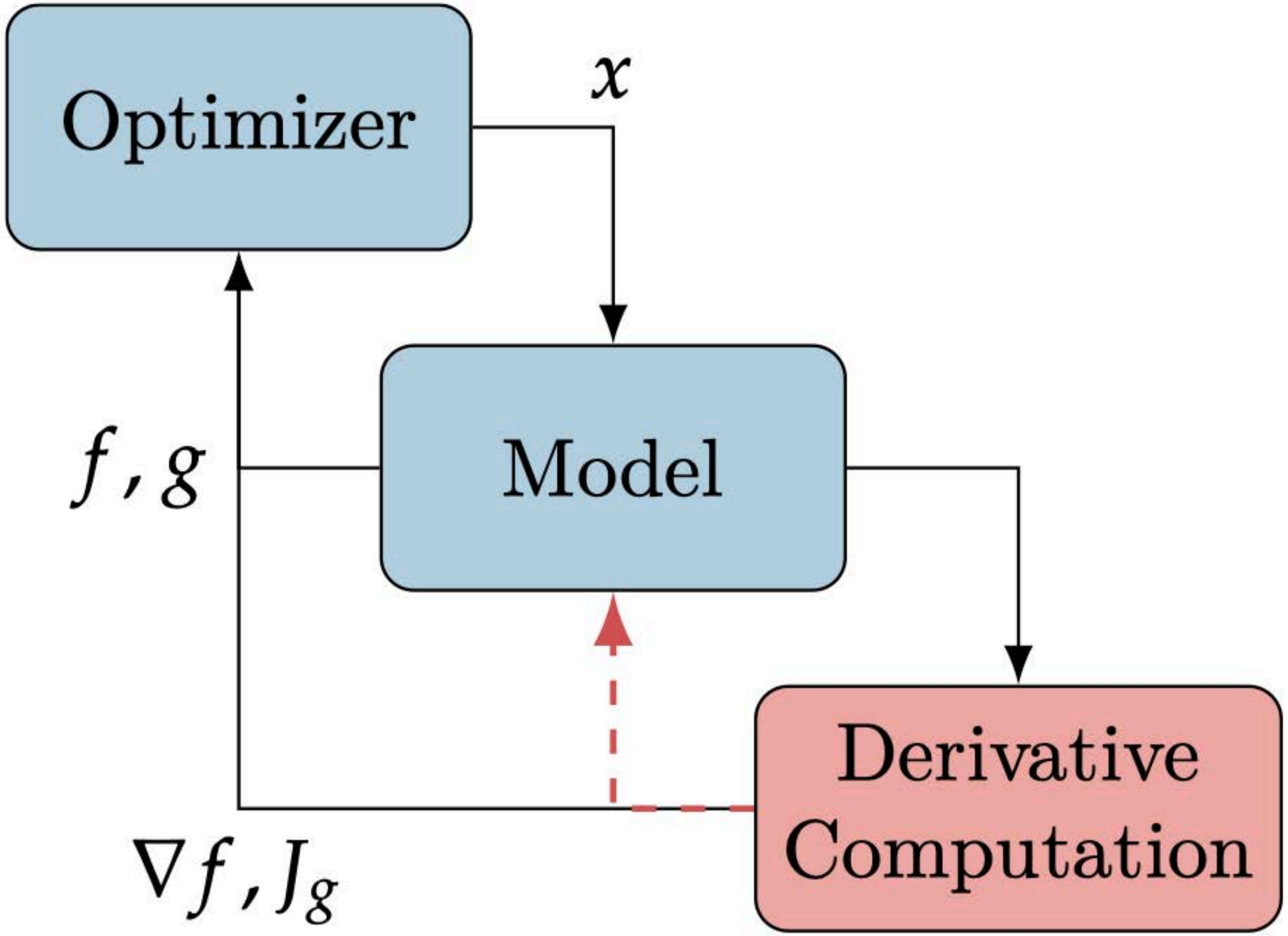






Martins and Ning. **Engineering Design Optimization.** Cambridge University Press, 2021.

Efficient and robust derivative computation is crucial for successful gradient-based optimization



Methods for computing derivatives

Black box

Inputs and outputs

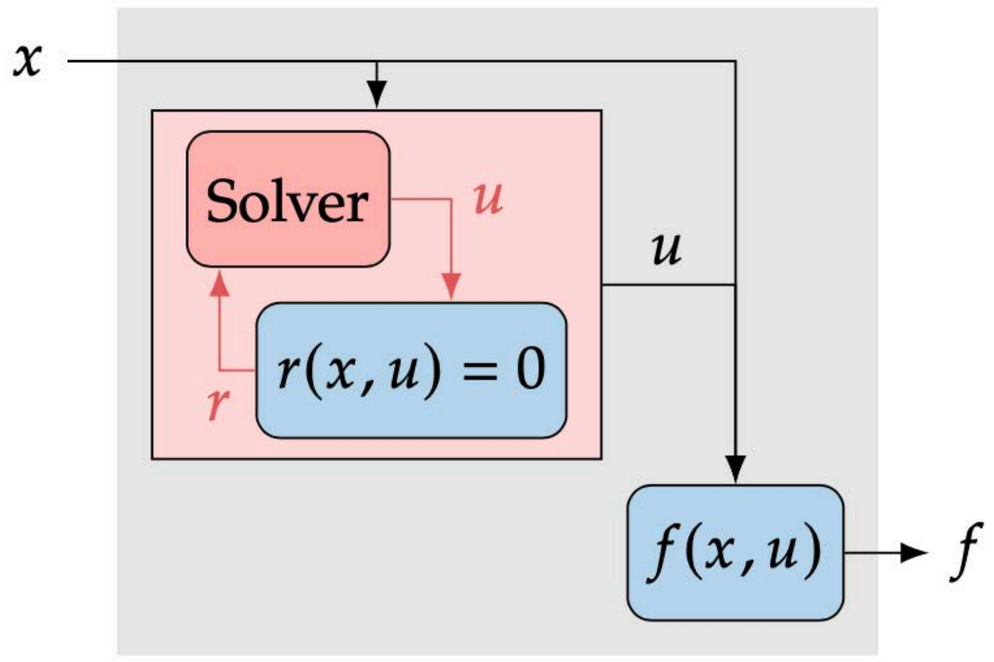
Finite differences

$$\frac{\partial f}{\partial x_j} = \frac{f(x) - f(x - h\hat{e}_j)}{h} + \mathcal{O}(h)$$

Complex-step method

$$\frac{\partial f}{\partial x_j} = \frac{\mathrm{Im}\left[f(x + ih\hat{e}_j)\right]}{h} + \mathcal{O}(h^2)$$

Analytic



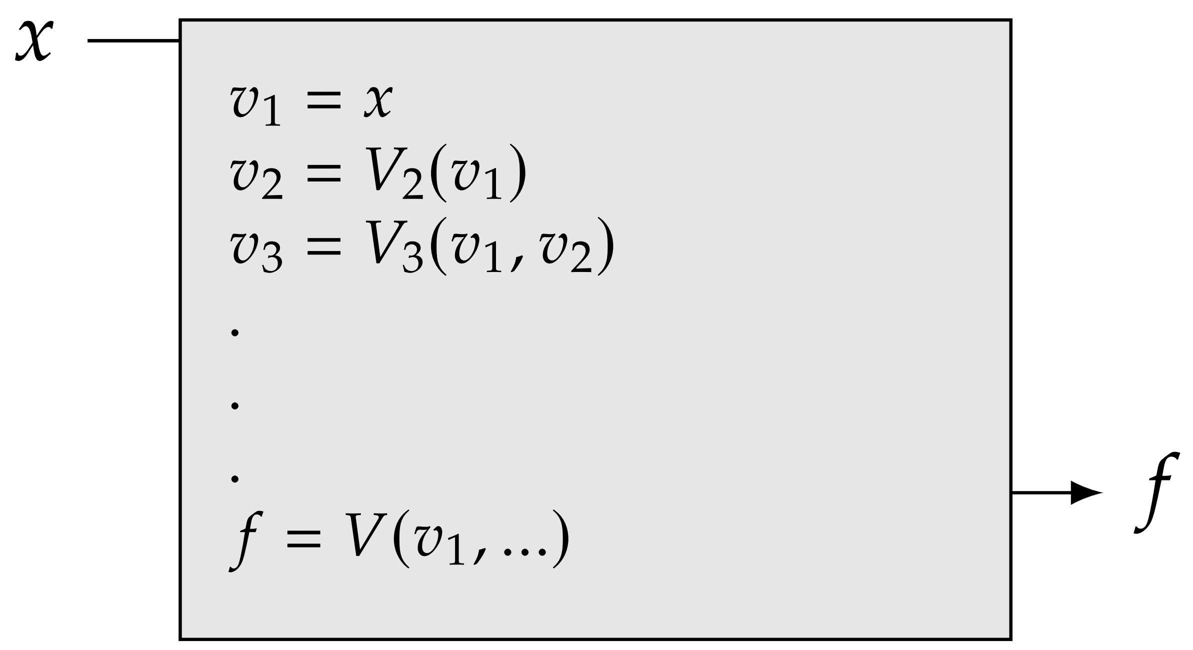
Governing equation residuals and states

Direct method

$$\underbrace{\frac{\mathrm{d}f}{\mathrm{d}x}}_{(n_f \times n_x)} = \underbrace{\frac{\partial f}{\partial x}}_{(n_f \times n_x)} - \underbrace{\underbrace{\frac{\partial f}{\partial u}}_{(n_f \times n_u)} \overbrace{\underbrace{\frac{\partial r}{\partial u}^{-1}}_{(n_u \times n_u)} \underbrace{\frac{\partial r}{\partial x}}_{(n_u \times n_x)}}^{\phi \quad (n_u \times n_x)}}_{\psi^\intercal \quad (n_f \times n_u)}$$

Adjoint method

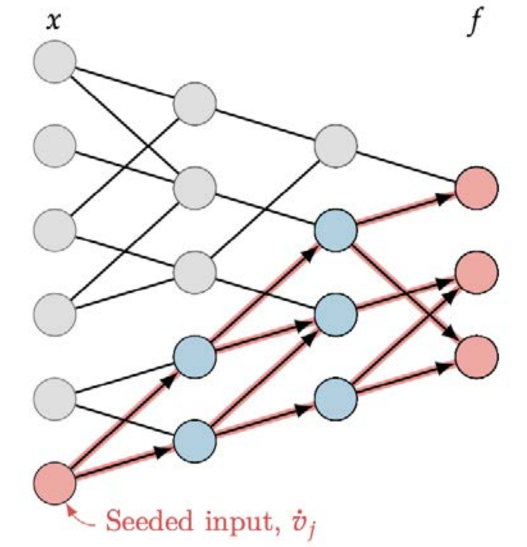
Algorithmic



Lines of code

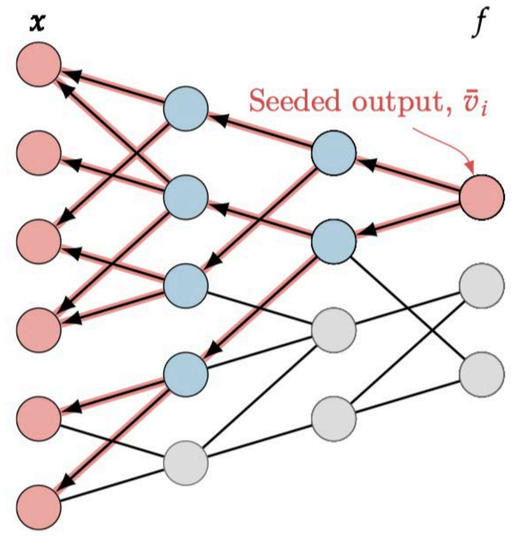
Forward mode

$$\frac{\mathrm{d}v_i}{\mathrm{d}v_j} = \sum_{k=j}^{i-1} \frac{\partial v_i}{\partial v_k} \frac{\mathrm{d}v_k}{\mathrm{d}v_j}$$



Reverse mode

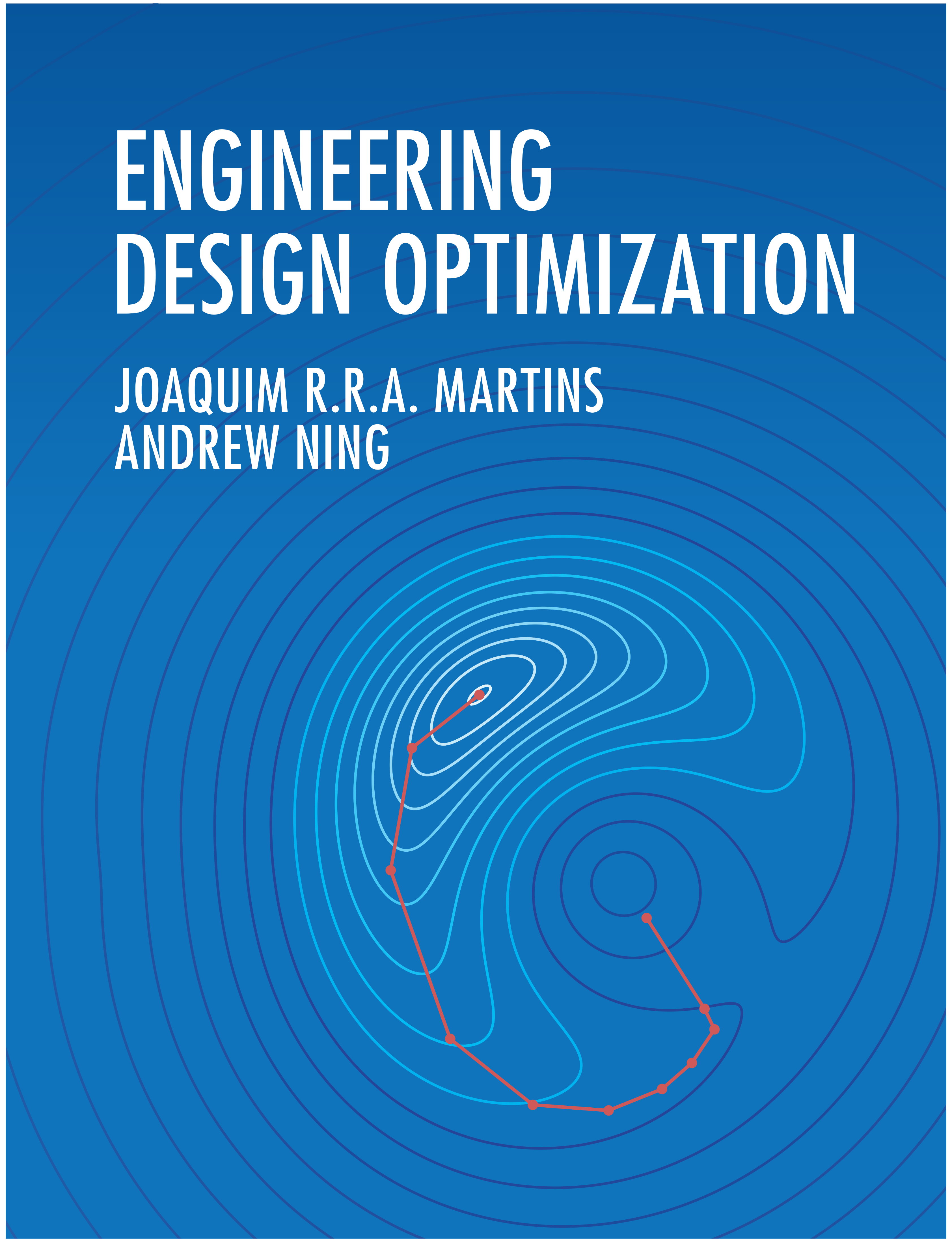
$$\frac{\mathrm{d}v_i}{\mathrm{d}v_j} = \sum_{k=j+1}^{i} \frac{\partial v_k}{\partial v_j} \frac{\mathrm{d}v_i}{\mathrm{d}v_k}$$

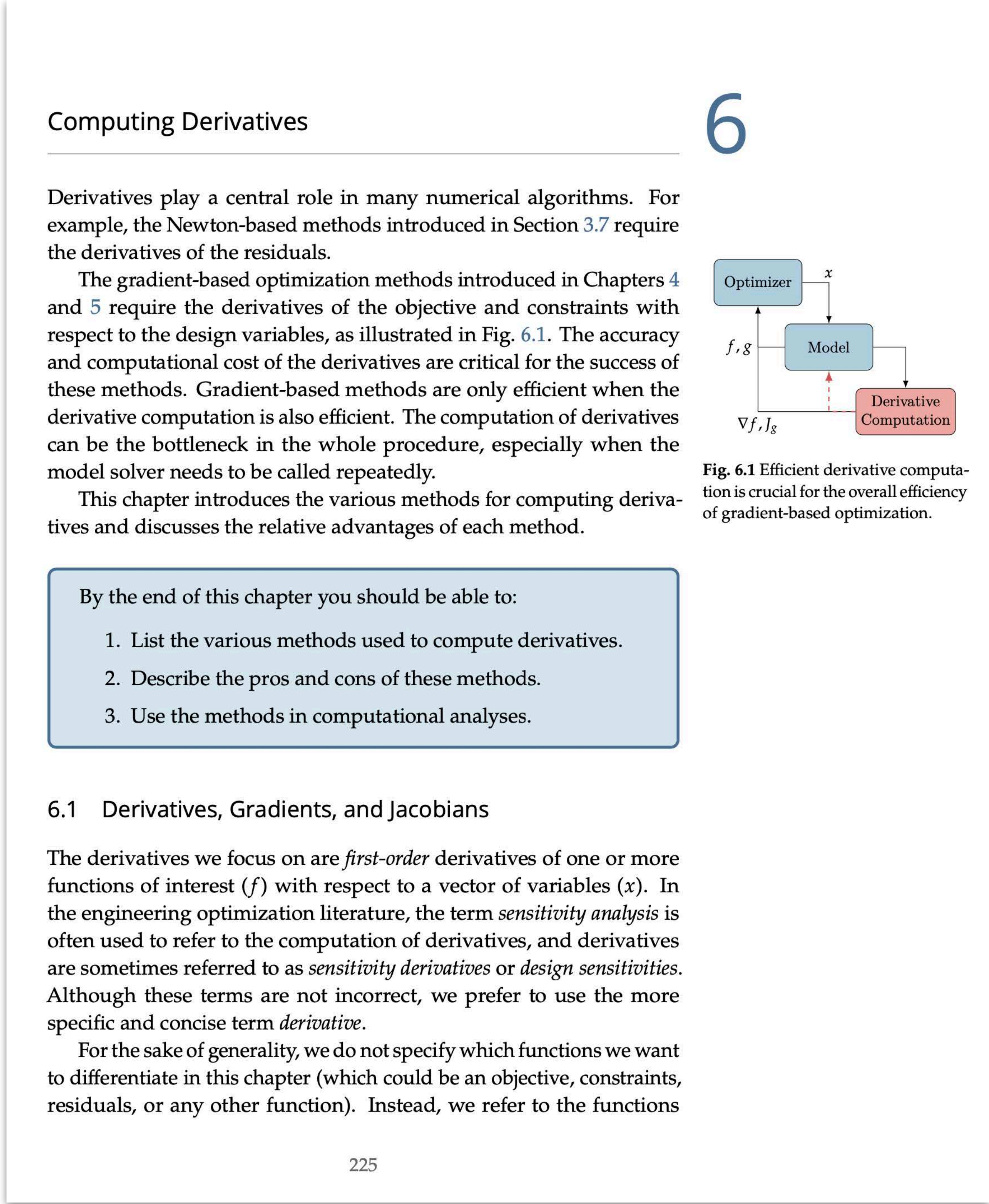


Martins, Sturdza, and Alonso. **The complex-step derivative approximation**. *ACM Transactions on Mathematical Software*, 2003.

Martins and Hwang. **Review and unification of methods for computing derivatives of multidisciplinary computational models**. *AIAA Journal*, 2013.

For more details, see Chapter 6 of my new book
(the PDF if free at https://mdobook.github.io)





Computing Derivatives

6

Derivatives play a central role in many numerical algorithms. For example, the Newton-based methods introduced in Section 3.7 require the derivatives of the residuals.

The gradient-based optimization methods introduced in Chapters 4 and 5 require the derivatives of the objective and constraints with respect to the design variables, as illustrated in Fig. 6.1. The accuracy and computational cost of the derivatives are critical for the success of these methods. Gradient-based methods are only efficient when the derivative computation is also efficient. The computation of derivatives can be the bottleneck in the whole procedure, especially when the model solver needs to be called repeatedly.

This chapter introduces the various methods for computing derivatives and discusses the relative advantages of each method.

Optimizer, Model, Derivative Computation, x, f, g, $\nabla f, J_g$

Fig. 6.1 Efficient derivative computation is crucial for the overall efficiency of gradient-based optimization.

By the end of this chapter you should be able to:

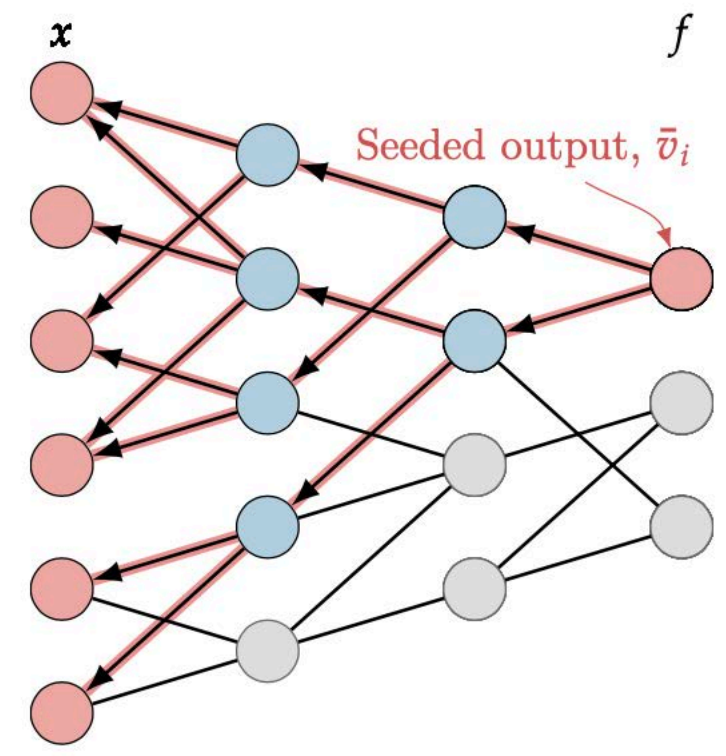
1. List the various methods used to compute derivatives.
2. Describe the pros and cons of these methods.
3. Use the methods in computational analyses.

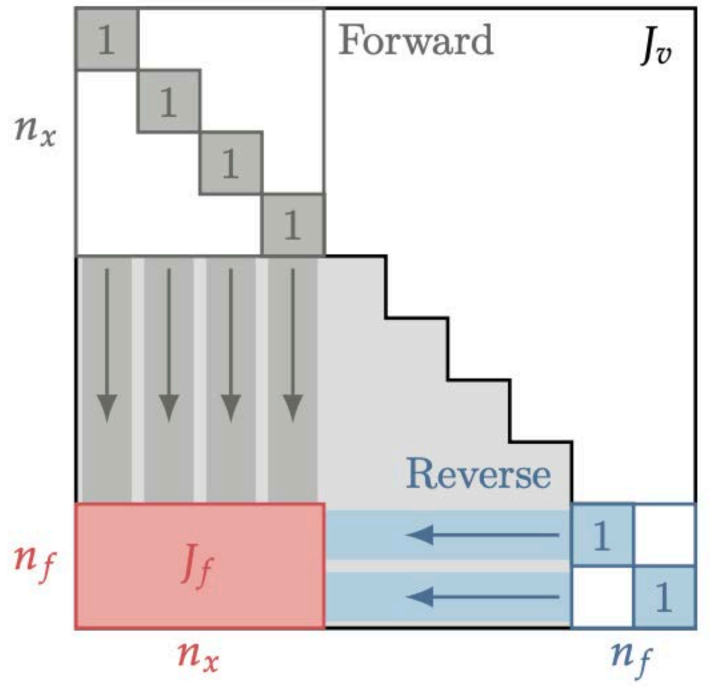
6.1 Derivatives, Gradients, and Jacobians

The derivatives we focus on are *first-order* derivatives of one or more functions of interest (f) with respect to a vector of variables (x). In the engineering optimization literature, the term *sensitivity analysis* is often used to refer to the computation of derivatives, and derivatives are sometimes referred to as *sensitivity derivatives* or *design sensitivities*. Although these terms are not incorrect, we prefer to use the more specific and concise term *derivative*.

For the sake of generality, we do not specify which functions we want to differentiate in this chapter (which could be an objective, constraints, residuals, or any other function). Instead, we refer to the functions

225





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

The unified derivatives equation (UDE) is the core of OpenMDAO

$$\frac{\partial r}{\partial u}\frac{\mathrm{d}u}{\mathrm{d}r} = I = \frac{\partial r}{\partial u}^{\intercal}\frac{\mathrm{d}u}{\mathrm{d}r}^{\intercal}$$

- The UDE was motivated by the desire to unify all methods for computing derivatives (implicit analytic methods and AD)
- The UDE concept was expanded to the modular analysis and unified derivatives (MAUD) architecture
- OpenMDAO was refactored to incorporate MAUD
- Now let's derive the UDE! (Sec. 6.9 of the book)

Martins and Hwang. **Review and unification of methods for computing derivatives of multidisciplinary computational models**. *AIAA Journal*, 2013.

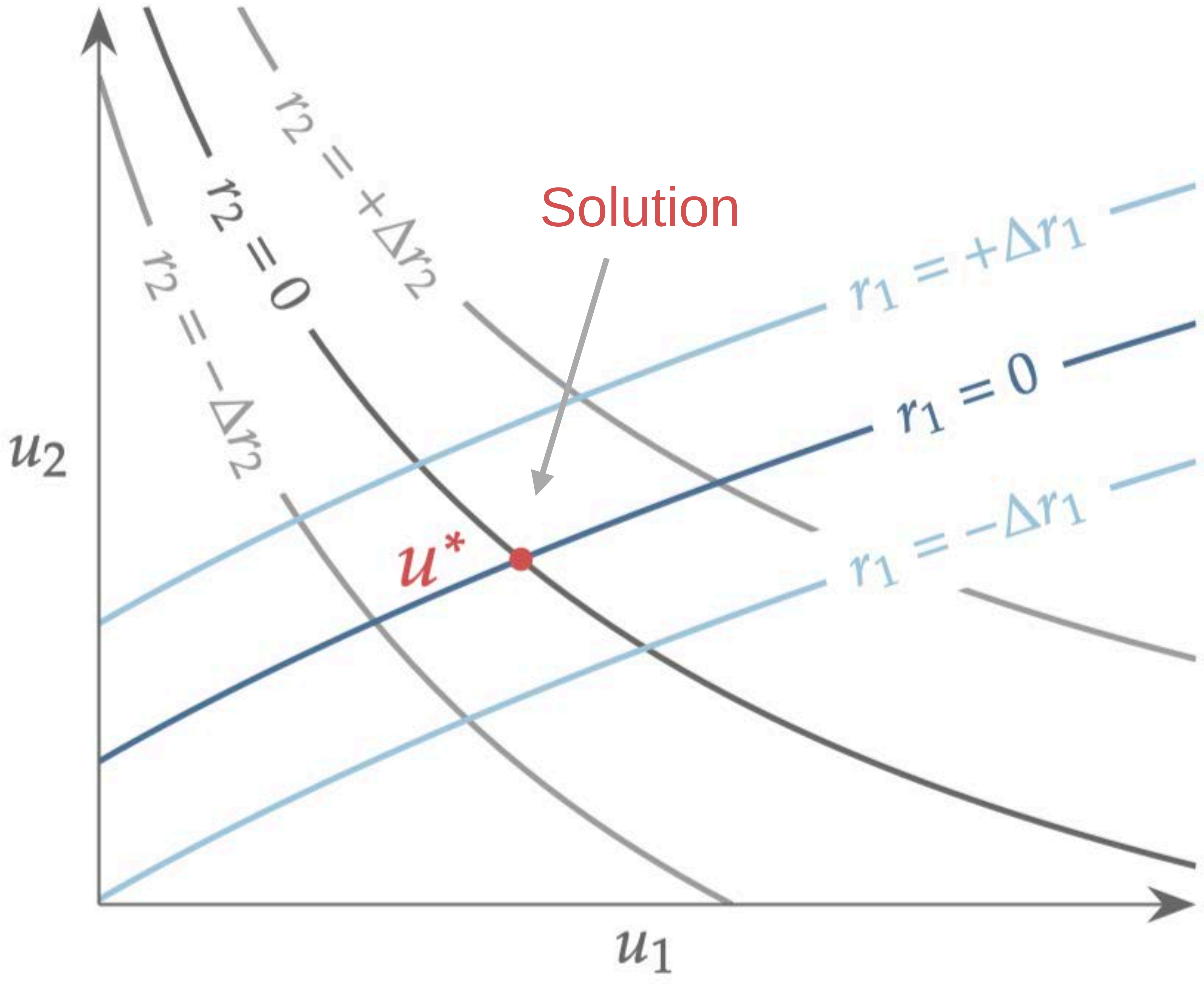
Hwang and Martins. **A computational architecture for coupling heterogeneous numerical models and computing coupled derivatives**. *ACM Transactions on Mathematical Software*, 2018

Governing equations can always be expressed as residuals

$$r_i(u_1, u_2, \ldots, u_n) = 0, \quad i = 1, \ldots, n$$

- Each residual is a scalar function of n state variables, $u_1, \ldots, u_n$
- There are n residual equations and n unknowns
- We assume that there is at least one solution
- In the $n = 2$ example, the each residual function corresponds to a contour plot
- Here we visualize a variation in the residual about zero ($\pm\Delta r$)

Example for $n = 2$

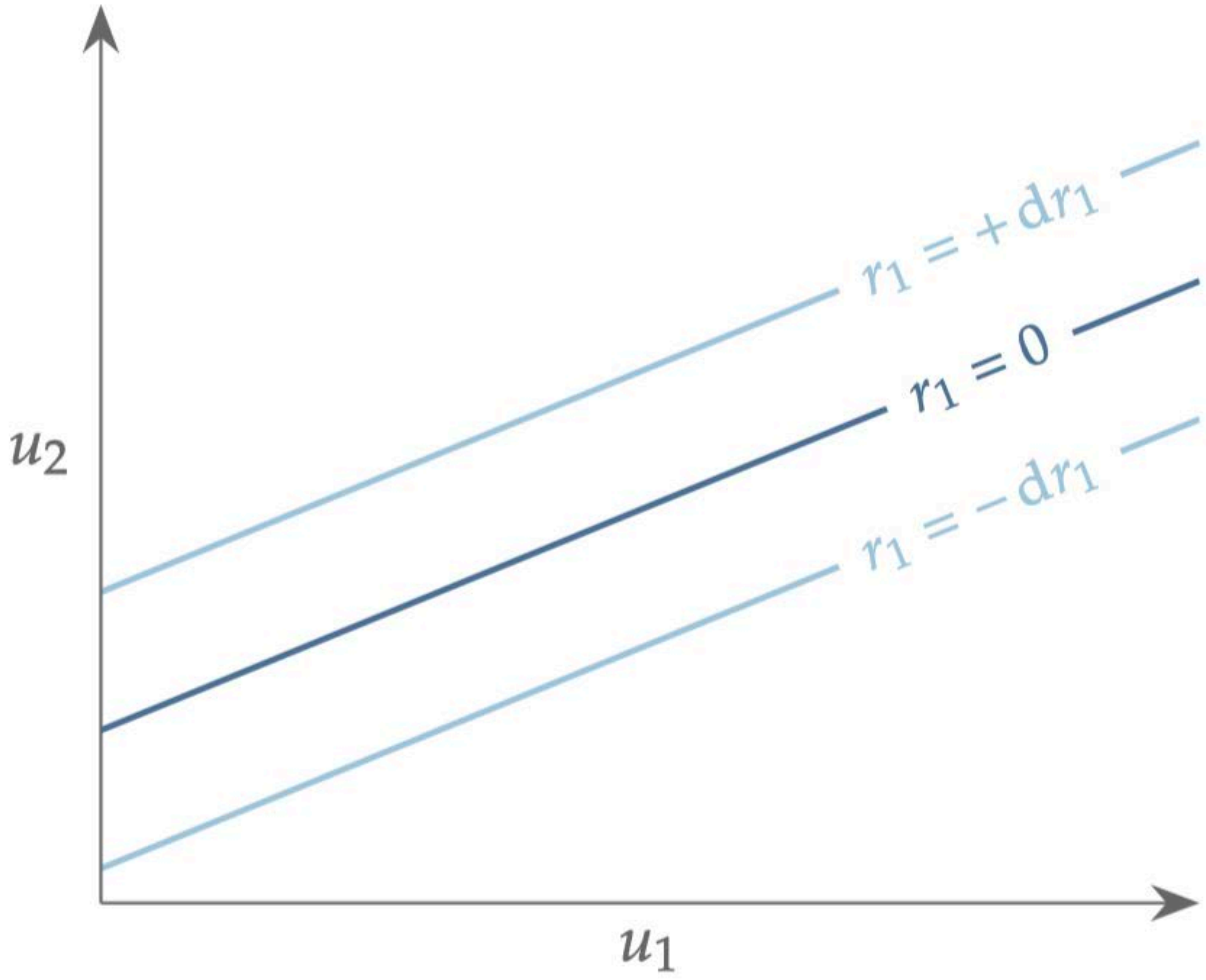


Let's consider the total differential of the residual (Why? We will see later...)

$$\mathrm{d}r_i = \frac{\partial r_i}{\partial u_1}\mathrm{d}u_1 + \ldots + \frac{\partial r_i}{\partial u_n}\mathrm{d}u_n, \quad i = 1, \ldots, n$$

- These are first-order changes in r due to perturbations in u
- In matrix form:

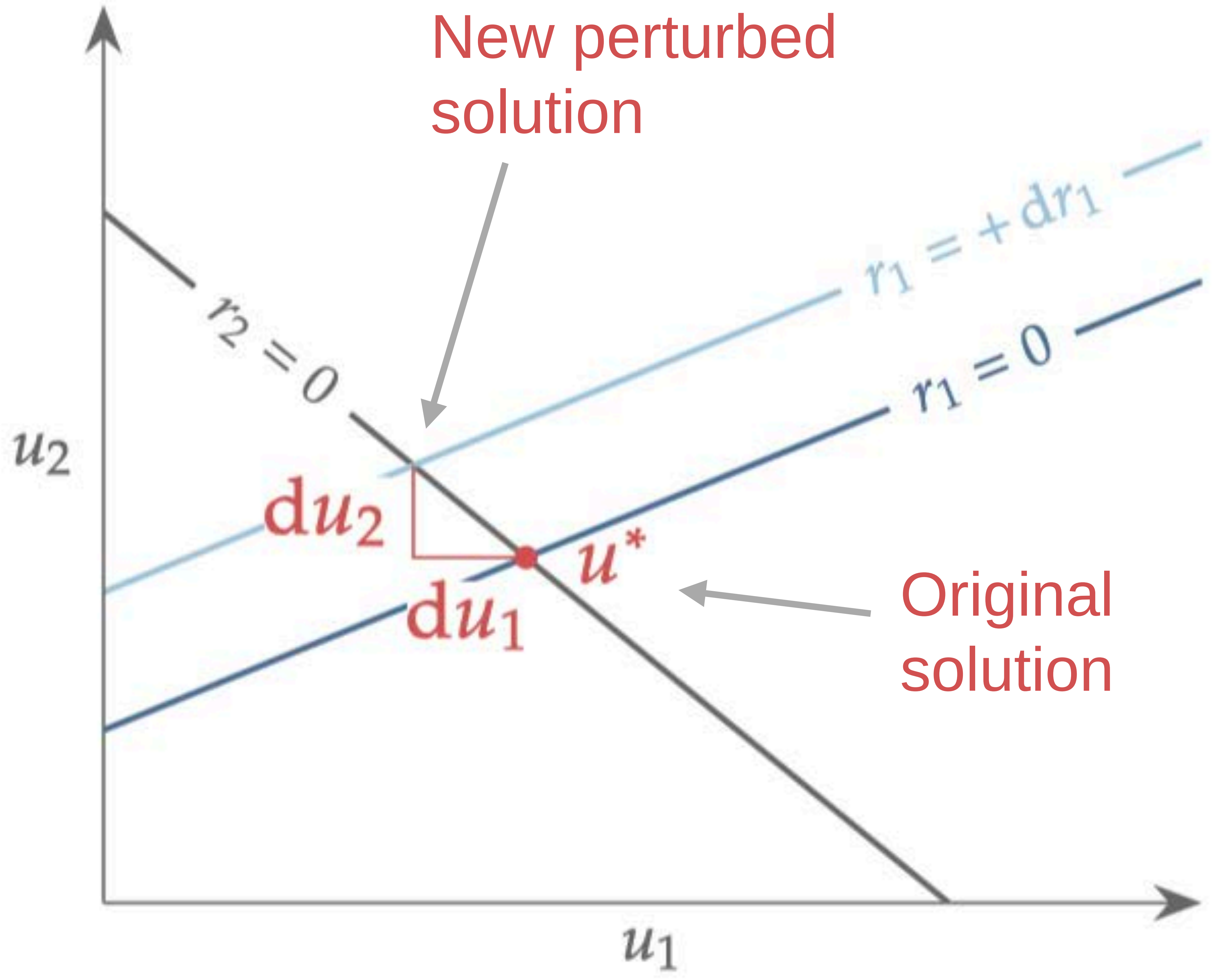
$$\begin{bmatrix} \frac{\partial r_1}{\partial u_1} & \cdots & \frac{\partial r_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_n}{\partial u_1} & \cdots & \frac{\partial r_n}{\partial u_n} \end{bmatrix} \begin{bmatrix} \mathrm{d}u_1 \\ \vdots \\ \mathrm{d}u_n \end{bmatrix} = \begin{bmatrix} \mathrm{d}r_1 \\ \vdots \\ \mathrm{d}r_n \end{bmatrix}$$



This linear system relates changes in the residuals to changes in the state variables

$$\begin{bmatrix} \frac{\partial r_1}{\partial u_1} & \cdots & \frac{\partial r_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_n}{\partial u_1} & \cdots & \frac{\partial r_n}{\partial u_n} \end{bmatrix} \begin{bmatrix} \mathrm{d}u_1 \\ \vdots \\ \mathrm{d}u_n \end{bmatrix} = \begin{bmatrix} \mathrm{d}r_1 \\ \vdots \\ \mathrm{d}r_n \end{bmatrix}$$

- Suppose we change one residual from zero to $r_1 = \mathrm{d}r_1$, while keeping all other residuals equal to zero.
- Solving for this RHS yields the corresponding changes in u that would satisfy this change

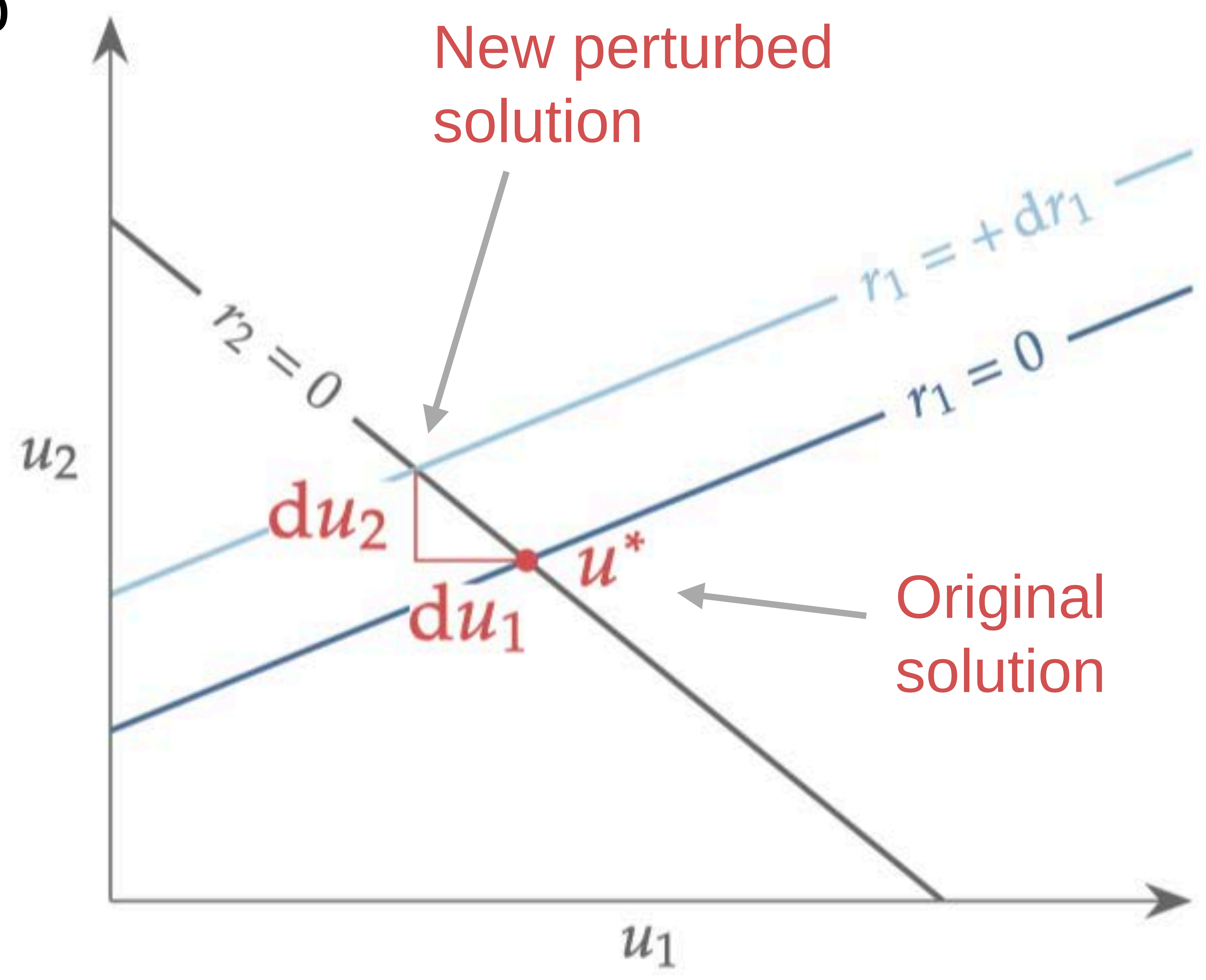


This gives the change in states for a prescribed change in residuals

By setting the appropriate RHS, we can find any total derivative

- Suppose we want the derivative $\mathrm{d}u_i/\mathrm{d}r_i$
- We set the ith entry to $\mathrm{d}r_i$ and the others to zero
- "Dividing" by $\mathrm{d}r_i$, yields the linear system:

$$\begin{bmatrix} \frac{\partial r_1}{\partial u_1} & \cdots & \frac{\partial r_1}{\partial u_i} & \cdots & \frac{\partial r_1}{\partial u_n} \\ \vdots & \ddots & \vdots & & \vdots \\ \frac{\partial r_i}{\partial u_1} & \cdots & \frac{\partial r_i}{\partial u_i} & \cdots & \frac{\partial r_i}{\partial u_n} \\ \vdots & & \vdots & \ddots & \vdots \\ \frac{\partial r_n}{\partial u_1} & \cdots & \frac{\partial r_n}{\partial u_i} & \cdots & \frac{\partial r_n}{\partial u_n} \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}u_1}{\mathrm{d}r_i} \\ \vdots \\ \frac{\mathrm{d}u_i}{\mathrm{d}r_i} \\ \vdots \\ \frac{\mathrm{d}u_n}{\mathrm{d}r_i} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$



Do this for all entries to get whole Jacobian matrix $\mathrm{d}u/\mathrm{d}r$

$$\begin{bmatrix} \frac{\partial r_1}{\partial u_1} & \cdots & \frac{\partial r_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_n}{\partial u_1} & \cdots & \frac{\partial r_n}{\partial u_n} \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}u_1}{\mathrm{d}r_1} & \cdots & \frac{\mathrm{d}u_1}{\mathrm{d}r_n} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}u_n}{\mathrm{d}r_1} & \cdots & \frac{\mathrm{d}u_n}{\mathrm{d}r_n} \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$$

This is the forward form of the UDE. In matrix form:

$$\boxed{\frac{\partial r}{\partial u}\frac{\mathrm{d}u}{\mathrm{d}r} = I}$$

How is this useful?

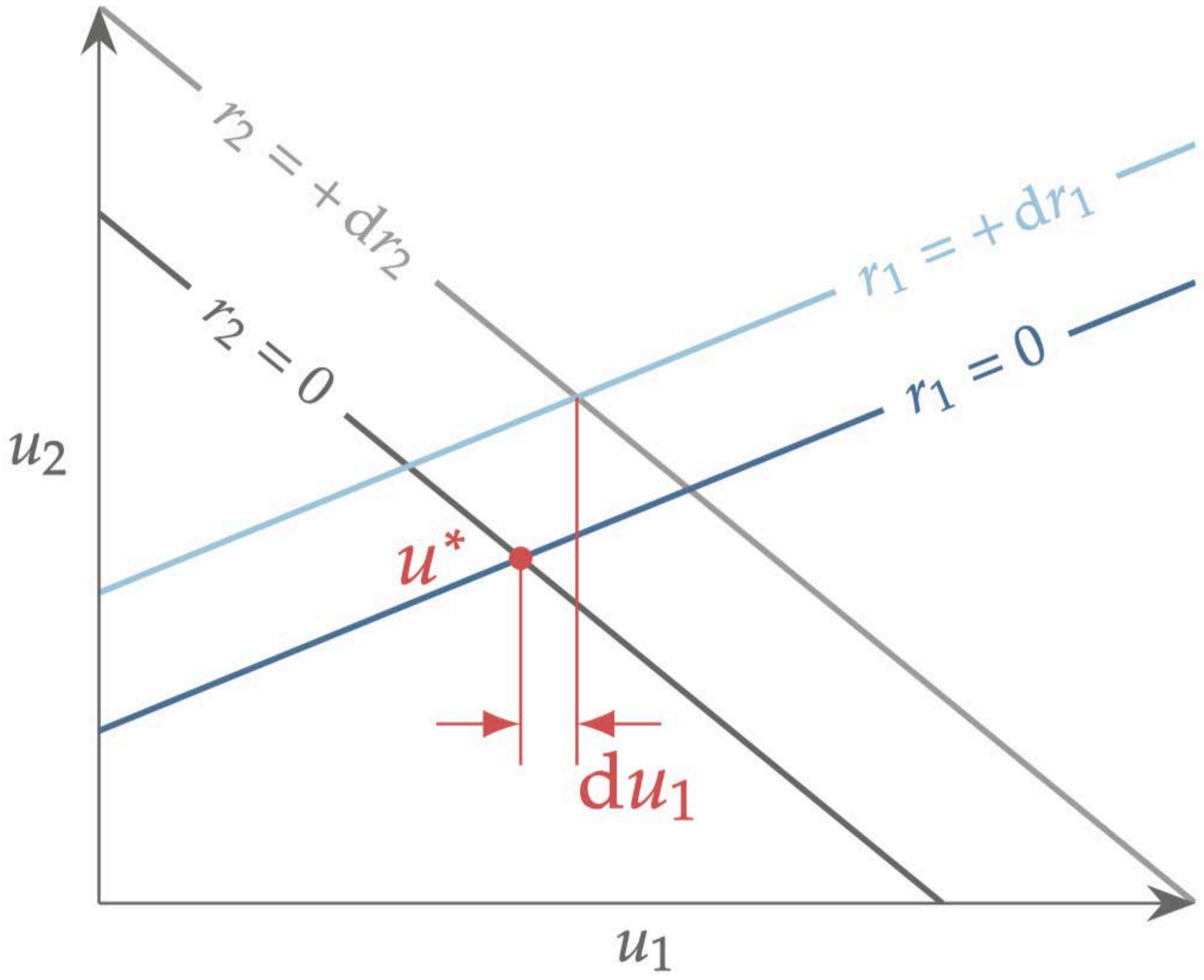
Reverse form of the UDE

$$AB = I \Rightarrow B = A^{-1} \Rightarrow B^\intercal = A^{-\intercal} \Rightarrow A^\intercal B^\intercal = I.$$

Therefore

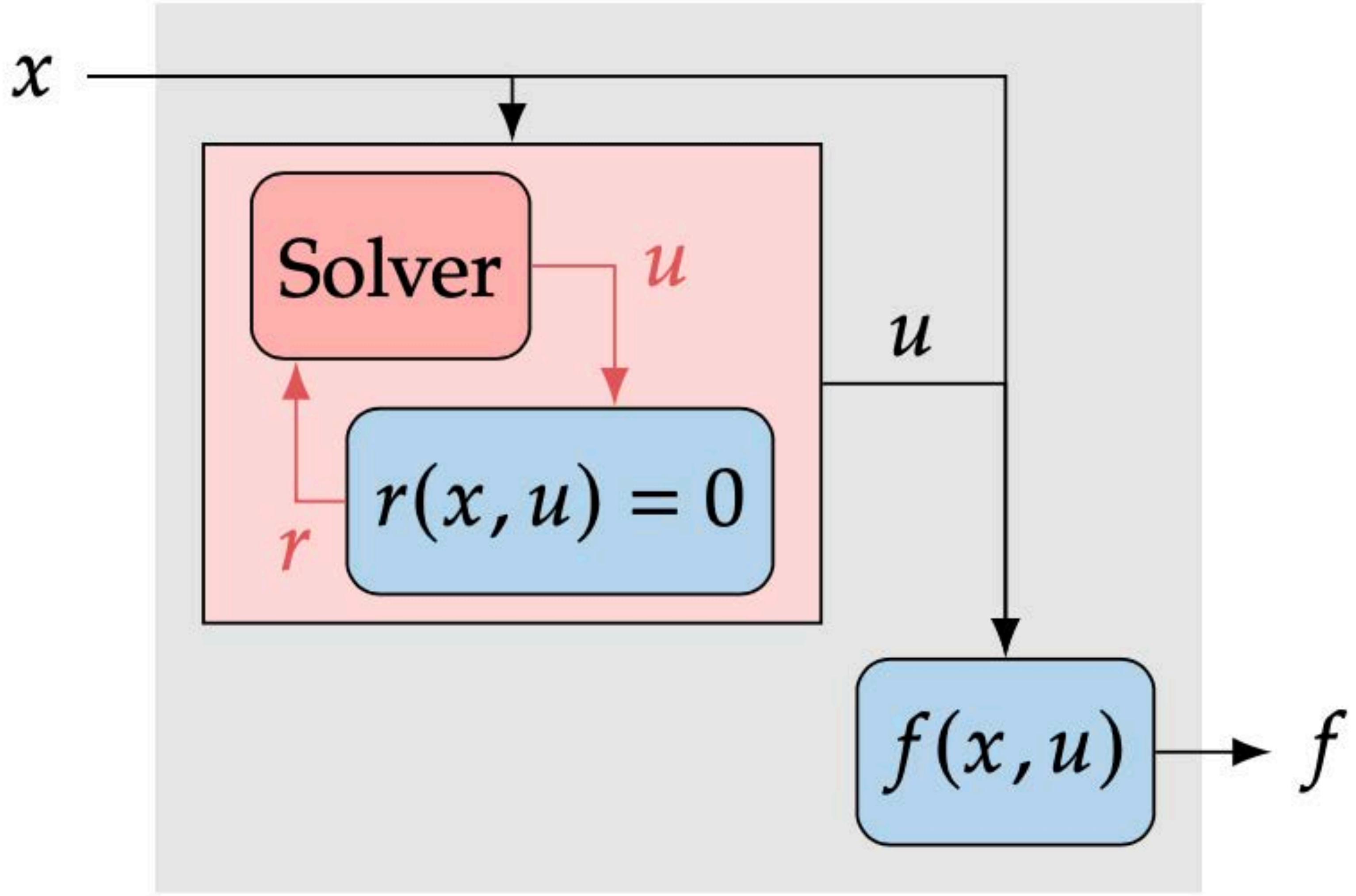
$$\boxed{\frac{\partial r}{\partial u}\frac{\mathrm{d}u}{\mathrm{d}r} = I} \Rightarrow \boxed{\frac{\partial r}{\partial u}^\intercal \frac{\mathrm{d}u}{\mathrm{d}r}^\intercal = I}$$

Again, how is this useful?



The reverse form gives the change in residuals for a prescribed change in states

UDE for mixed implicit and explicit components



- Suppose we want $\mathrm{d}f/\mathrm{d}x$, where $r(x,u) = 0$
- We can define new set of states and residuals as:

$$\hat{u} \equiv \begin{bmatrix} x \\ u \\ f \end{bmatrix} \qquad \hat{r} \equiv \begin{bmatrix} x - \check{x} \\ r - \check{r}(x,u) \\ f - \check{f}(x,u) \end{bmatrix} = 0$$

- Applying the forward and reverse UDE:

This is what we want!

$$\begin{bmatrix} I & 0 & 0 \\ -\frac{\partial \check{r}}{\partial x} & -\frac{\partial \check{r}}{\partial u} & 0 \\ -\frac{\partial \check{f}}{\partial x} & -\frac{\partial \check{f}}{\partial u} & I \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ \frac{\mathrm{d}u}{\mathrm{d}x} & \frac{\mathrm{d}u}{\mathrm{d}r} & 0 \\ \frac{\mathrm{d}f}{\mathrm{d}x} & \frac{\mathrm{d}f}{\mathrm{d}r} & I \end{bmatrix} = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} = \begin{bmatrix} I & -\frac{\partial \check{r}}{\partial x}^{\intercal} & -\frac{\partial \check{f}}{\partial x}^{\intercal} \\ 0 & -\frac{\partial \check{r}}{\partial u}^{\intercal} & -\frac{\partial \check{f}}{\partial u}^{\intercal} \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} I & \frac{\mathrm{d}u}{\mathrm{d}x}^{\intercal} & \frac{\mathrm{d}f}{\mathrm{d}x}^{\intercal} \\ 0 & \frac{\mathrm{d}u}{\mathrm{d}r}^{\intercal} & \frac{\mathrm{d}f}{\mathrm{d}r}^{\intercal} \\ 0 & 0 & I \end{bmatrix}$$

UDE yields the implicit analytic methods: direct and adjoint

$$
\begin{bmatrix} I & 0 & 0 \\ -\frac{\partial \check{r}}{\partial x} & -\frac{\partial \check{r}}{\partial u} & 0 \\ -\frac{\partial \check{f}}{\partial x} & -\frac{\partial \check{f}}{\partial u} & I \end{bmatrix}
\begin{bmatrix} I & 0 & 0 \\ \frac{\mathrm{d}u}{\mathrm{d}x} & \frac{\mathrm{d}u}{\mathrm{d}r} & 0 \\ \frac{\mathrm{d}f}{\mathrm{d}x} & \frac{\mathrm{d}f}{\mathrm{d}r} & I \end{bmatrix}
=
\begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}
=
\begin{bmatrix} I & -\frac{\partial \check{r}}{\partial x}^{\intercal} & -\frac{\partial \check{f}}{\partial x}^{\intercal} \\ 0 & -\frac{\partial \check{r}}{\partial u}^{\intercal} & -\frac{\partial \check{f}}{\partial u}^{\intercal} \\ 0 & 0 & I \end{bmatrix}
\begin{bmatrix} I & \frac{\mathrm{d}u}{\mathrm{d}x}^{\intercal} & \frac{\mathrm{d}f}{\mathrm{d}x}^{\intercal} \\ 0 & \frac{\mathrm{d}u}{\mathrm{d}r}^{\intercal} & \frac{\mathrm{d}f}{\mathrm{d}r}^{\intercal} \\ 0 & 0 & I \end{bmatrix}
$$

$$\frac{\partial r}{\partial u}\phi = \frac{\partial r}{\partial x}$$

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\partial f}{\partial x} - \frac{\partial f}{\partial u}\phi \,.$$

$$\frac{\partial r}{\partial u}^{\intercal}\psi = \frac{\partial f}{\partial u}^{\intercal}$$

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\partial f}{\partial x} - \psi^{\intercal}\frac{\partial r}{\partial x}$$

The UDE also yields algorithmic differentiation (AD)

Define states and residuals as

$$v_i = \check{v}_i(v_1, \ldots, v_{i-1}), \quad i = 1, \ldots, n$$
$$r_i = v_i - \check{v}_i(v_1, \ldots, v_{i-1})$$

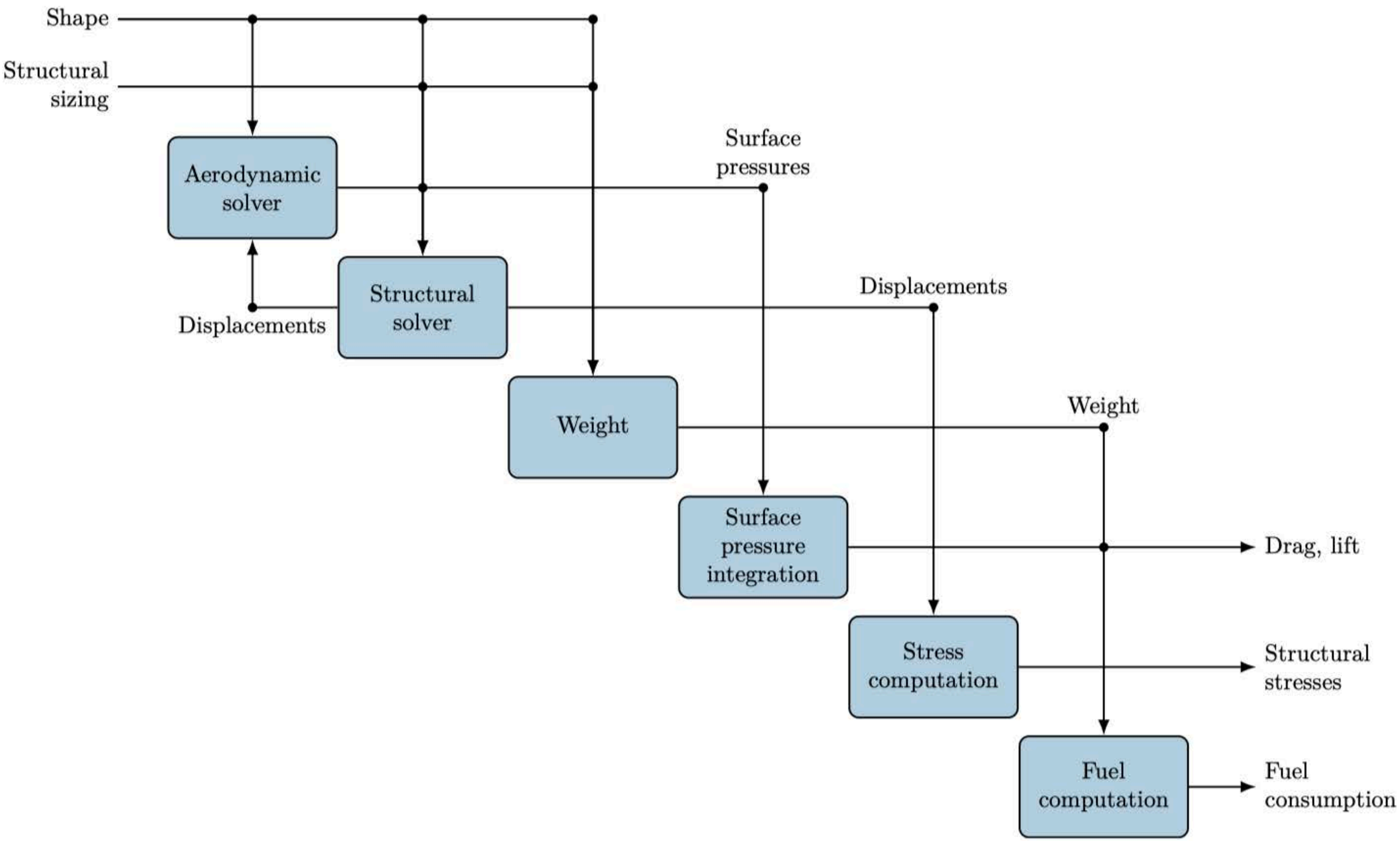
Then, the forward UDE becomes

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ -\dfrac{\partial \check{v}_2}{\partial v_1} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ -\dfrac{\partial \check{v}_n}{\partial v_1} & \cdots & -\dfrac{\partial \check{v}_n}{\partial v_{n-1}} & 1 \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}v_1}{\mathrm{d}v_1} & 0 & \cdots & 0 \\ \dfrac{\mathrm{d}v_2}{\mathrm{d}v_1} & \dfrac{\mathrm{d}v_2}{\mathrm{d}v_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \dfrac{\mathrm{d}v_n}{\mathrm{d}v_1} & \cdots & \dfrac{\mathrm{d}v_n}{\mathrm{d}v_{n-1}} & \dfrac{\mathrm{d}v_n}{\mathrm{d}v_n} \end{bmatrix} = I$$

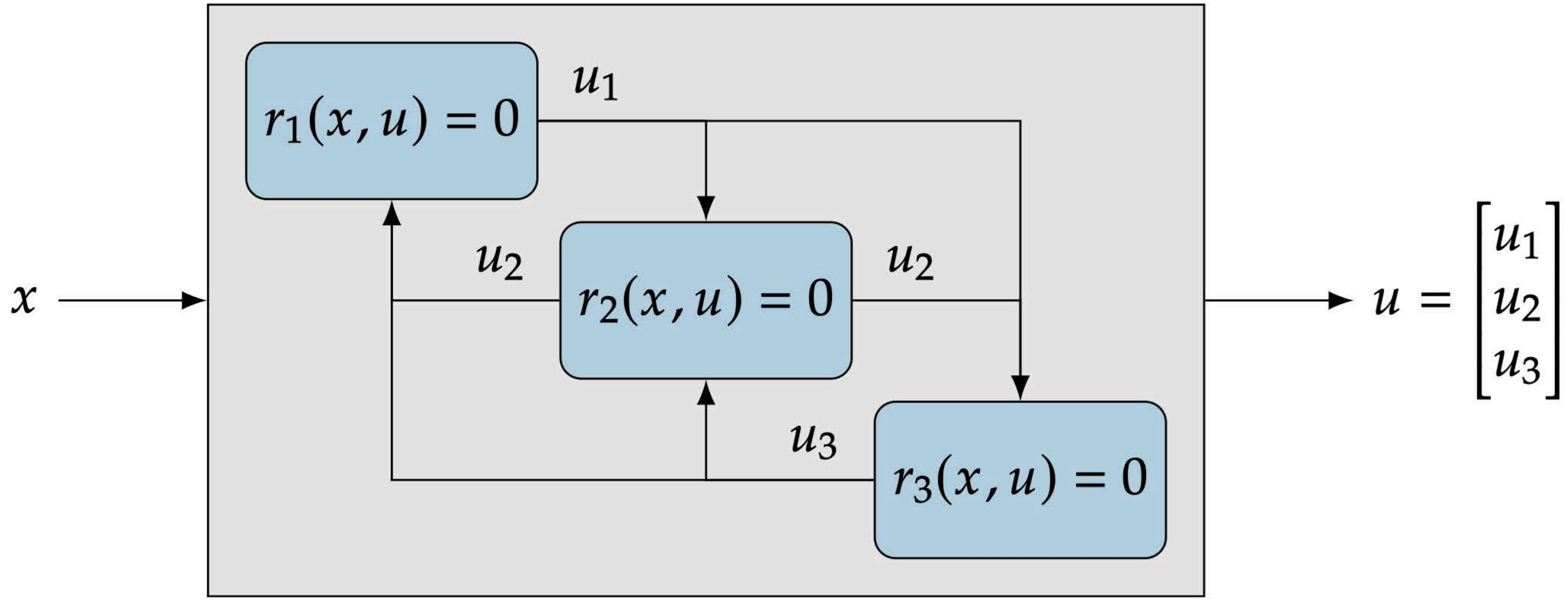
The reverse form of UDE yields reverse AD

$$\begin{bmatrix} 1 & -\frac{\partial \check{v}_2}{\partial v_1} & \dots & -\frac{\partial \check{v}_n}{\partial v_1} \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\frac{\partial \check{v}_n}{\partial v_{n-1}} \\ 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}v_1}{\mathrm{d}v_1} & \frac{\mathrm{d}v_2}{\mathrm{d}v_1} & \dots & \frac{\mathrm{d}v_n}{\mathrm{d}v_1} \\ 0 & \frac{\mathrm{d}v_2}{\mathrm{d}v_2} & \ddots & \vdots \\ \vdots & \ddots & \frac{\mathrm{d}v_{n-1}}{\mathrm{d}v_{n-1}} & \frac{\mathrm{d}v_n}{\mathrm{d}v_{n-1}} \\ 0 & \dots & 0 & \frac{\mathrm{d}v_n}{\mathrm{d}v_n} \end{bmatrix} = I$$

Now let us consider multiple components of disciplines



A coupled model has multiple residual and state subvectors



$$r(u) = 0 \equiv \begin{cases} r_1(u_1;\, u_2, \ldots, u_i, \ldots, u_n) = 0 \\ \vdots \\ r_i(u_i;\, u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_n) = 0 \\ \vdots \\ r_n(u_n;\, u_1, \ldots, u_i, \ldots, u_{n-1}) = 0 \end{cases}$$

The forward UDE yields the coupled direct method

$$r(u) \equiv \begin{bmatrix} r_1(u) \\ \vdots \\ r_n(u) \end{bmatrix}, \quad u \equiv \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial r_1}{\partial u_1} & \cdots & \frac{\partial r_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_n}{\partial u_1} & \cdots & \frac{\partial r_n}{\partial u_n} \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} \frac{\partial r_1}{\partial x} \\ \vdots \\ \frac{\partial r_n}{\partial x} \end{bmatrix}$$

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\partial f}{\partial x} - \begin{bmatrix} \frac{\partial f}{\partial u_1} & \cdots & \frac{\partial f}{\partial u_n} \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_n \end{bmatrix}$$

Solve n_x times

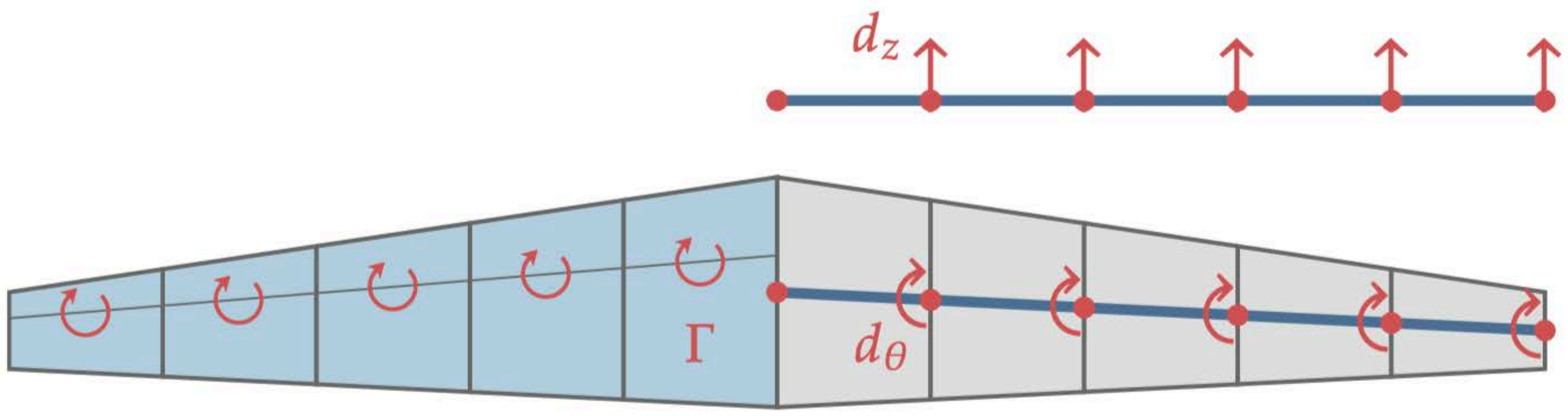
The reverse UDE yields the coupled adjoint method

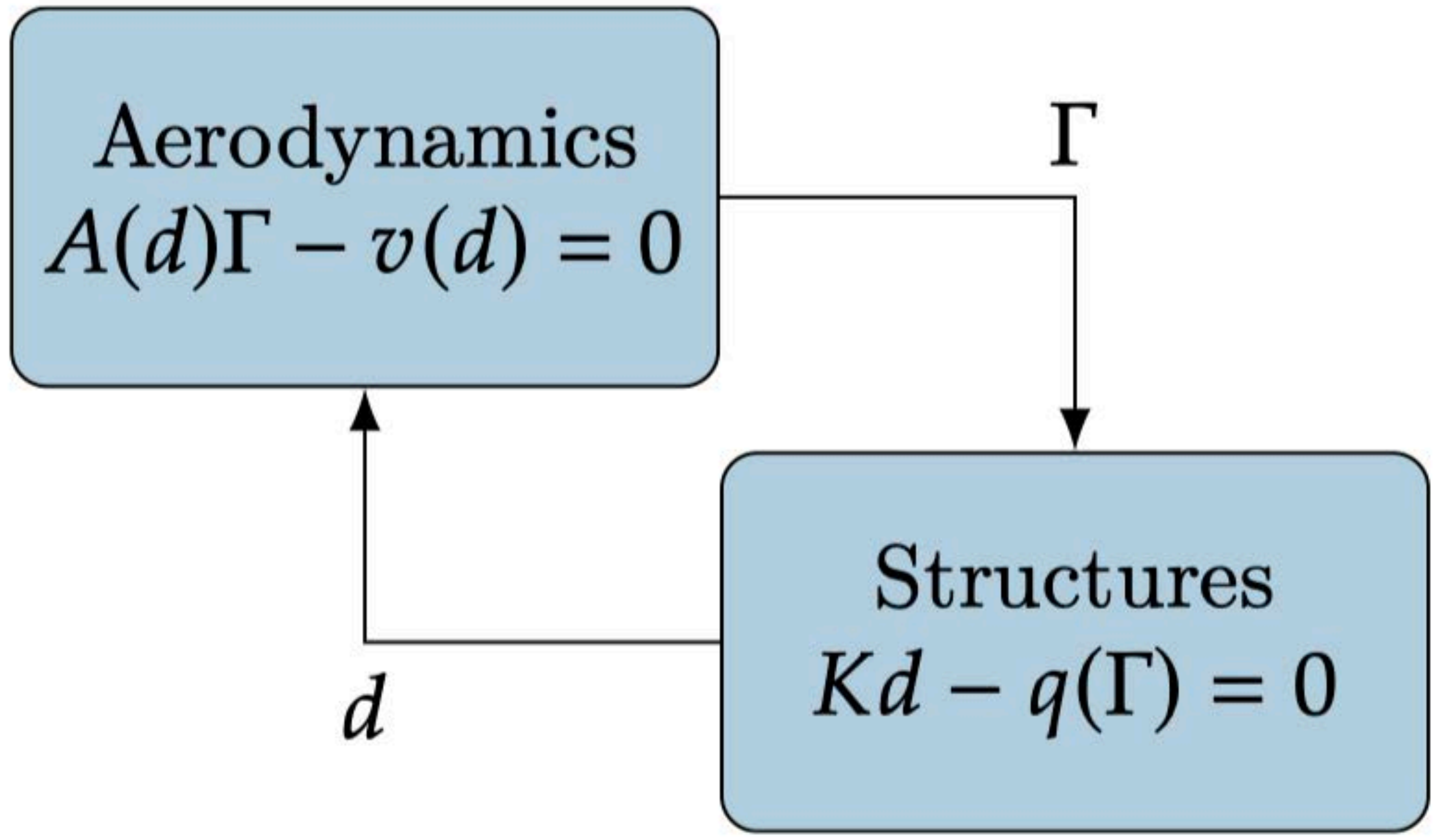
$$\begin{bmatrix} \dfrac{\partial r_1}{\partial u_1}^{\intercal} & \cdots & \dfrac{\partial r_n}{\partial u_1}^{\intercal} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial r_1}{\partial u_n}^{\intercal} & \cdots & \dfrac{\partial r_n}{\partial u_n}^{\intercal} \end{bmatrix} \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_n \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial u_1}^{\intercal} \\ \vdots \\ \dfrac{\partial f}{\partial u_n}^{\intercal} \end{bmatrix}$$

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\partial f}{\partial x} - \left[\psi_1^{\intercal} \ldots \psi_n^{\intercal}\right] \begin{bmatrix} \dfrac{\partial r_1}{\partial x} \\ \vdots \\ \dfrac{\partial r_n}{\partial x} \end{bmatrix}$$

Solve n_f times

Wing example





The states are the circulations and displacements

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \Gamma \\ d \end{bmatrix}$$

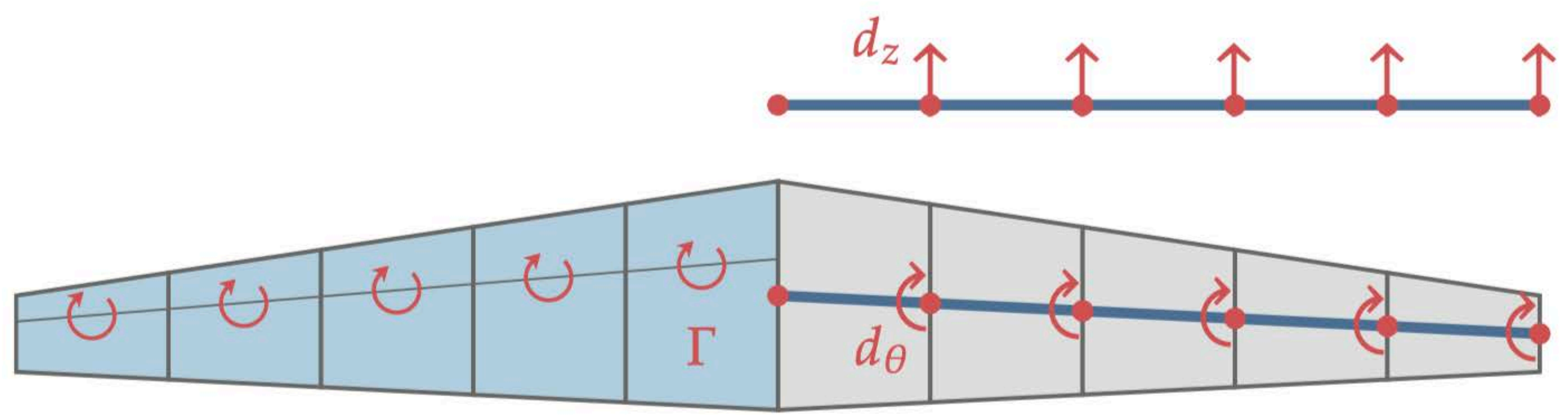
There are two corresponding sets of residual equations

$$r = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} A(d)\Gamma - v(d) \\ Kd - q(\Gamma) \end{bmatrix}$$

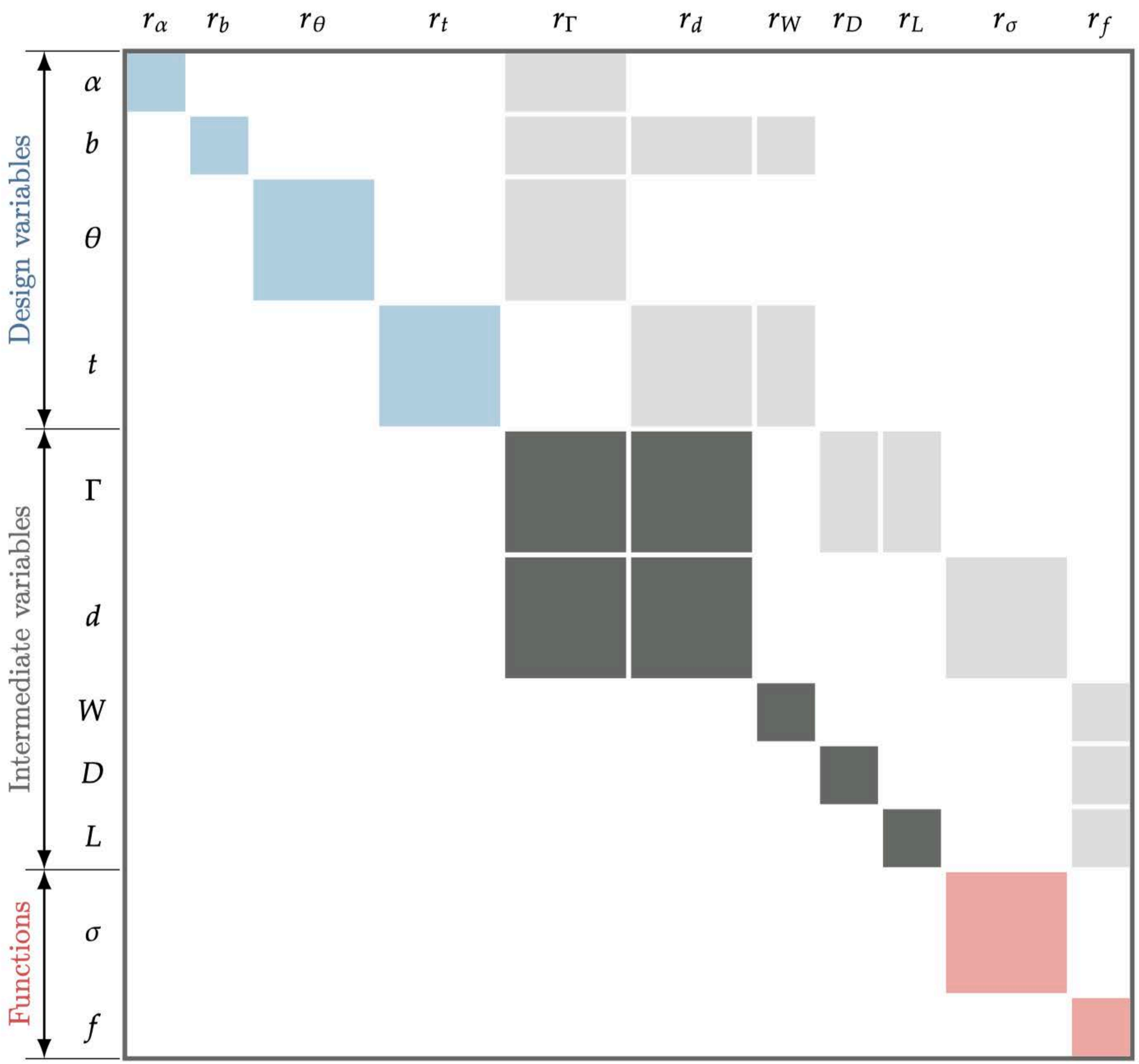
The Jacobian is

$$\frac{\partial r}{\partial u} = \begin{bmatrix} \frac{\partial r_1}{\partial u_1} & \frac{\partial r_1}{\partial u_2} \\ \frac{\partial r_2}{\partial u_1} & \frac{\partial r_2}{\partial u_2} \end{bmatrix} = \begin{bmatrix} A & \frac{\partial A}{\partial d}\Gamma - \frac{\partial v}{\partial d} \\ -\frac{\partial q}{\partial \Gamma} & K \end{bmatrix}$$

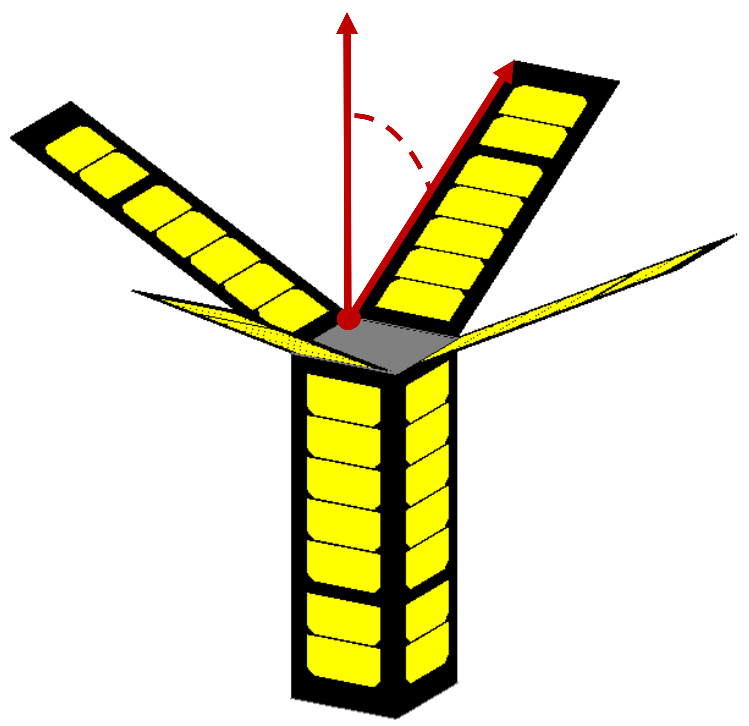
Wing example

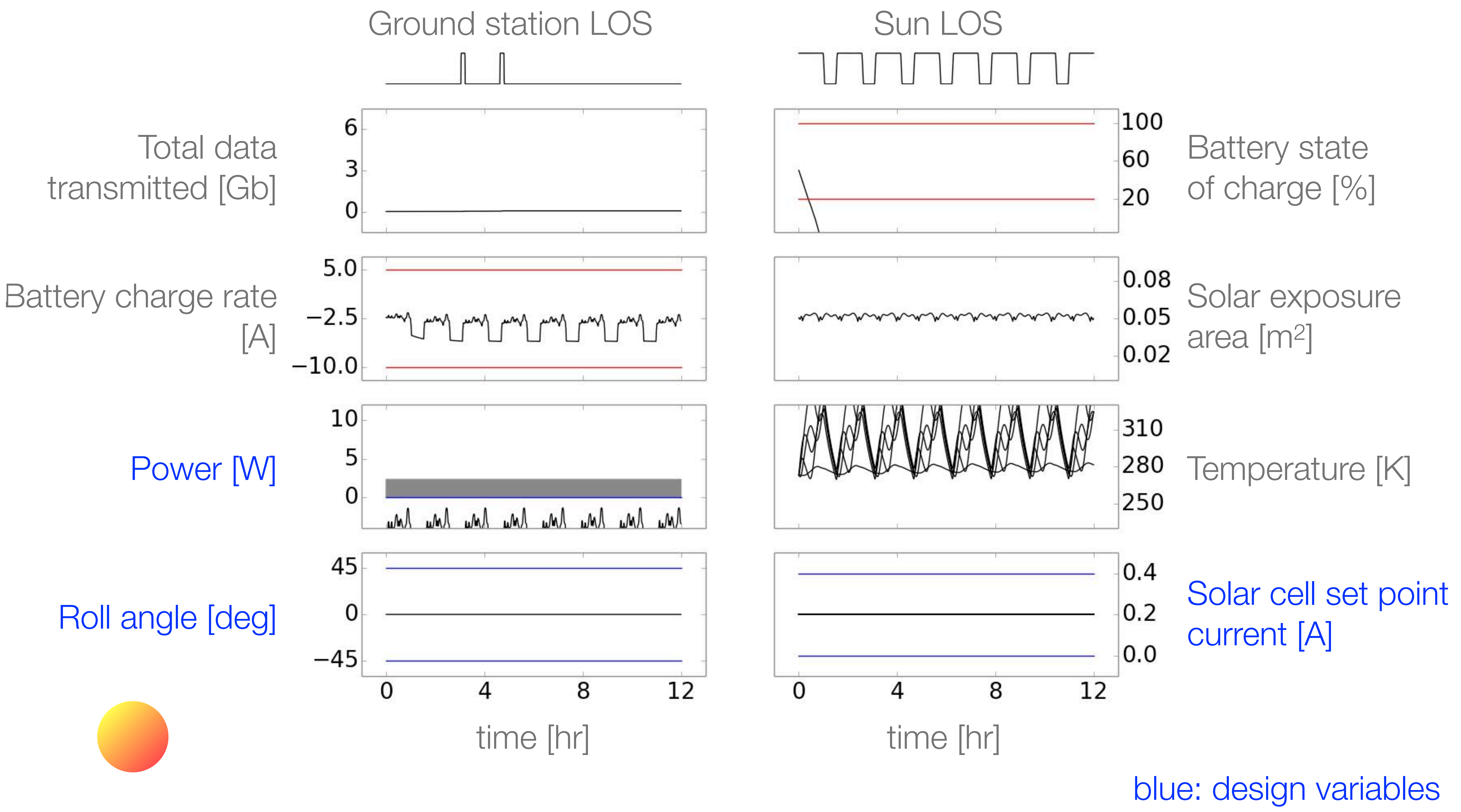


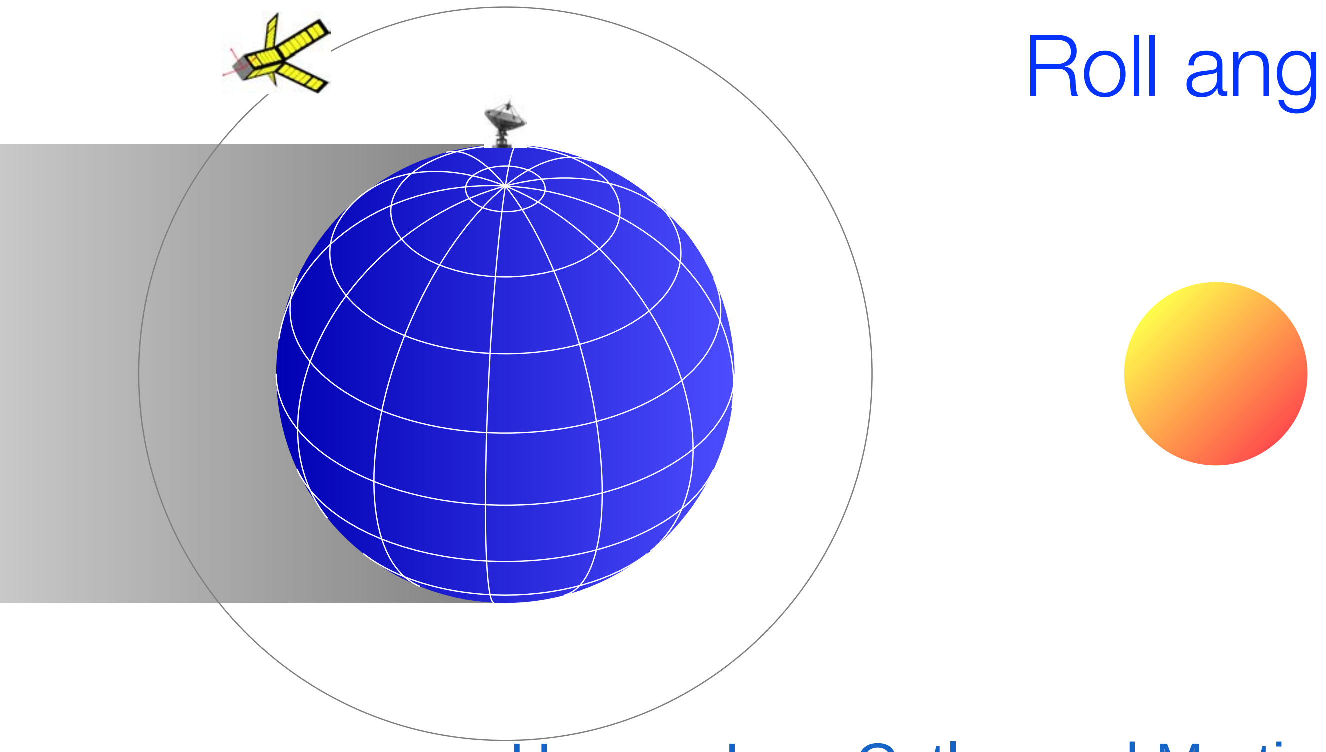
- α : angle of attack
- b : wing span
- θ : twist distribution (vector)
- t : thickness distribution (vector)
- σ : stress distribution (vector)
- f : fuel burn



MAUD was first implemented in a satellite MDO problem

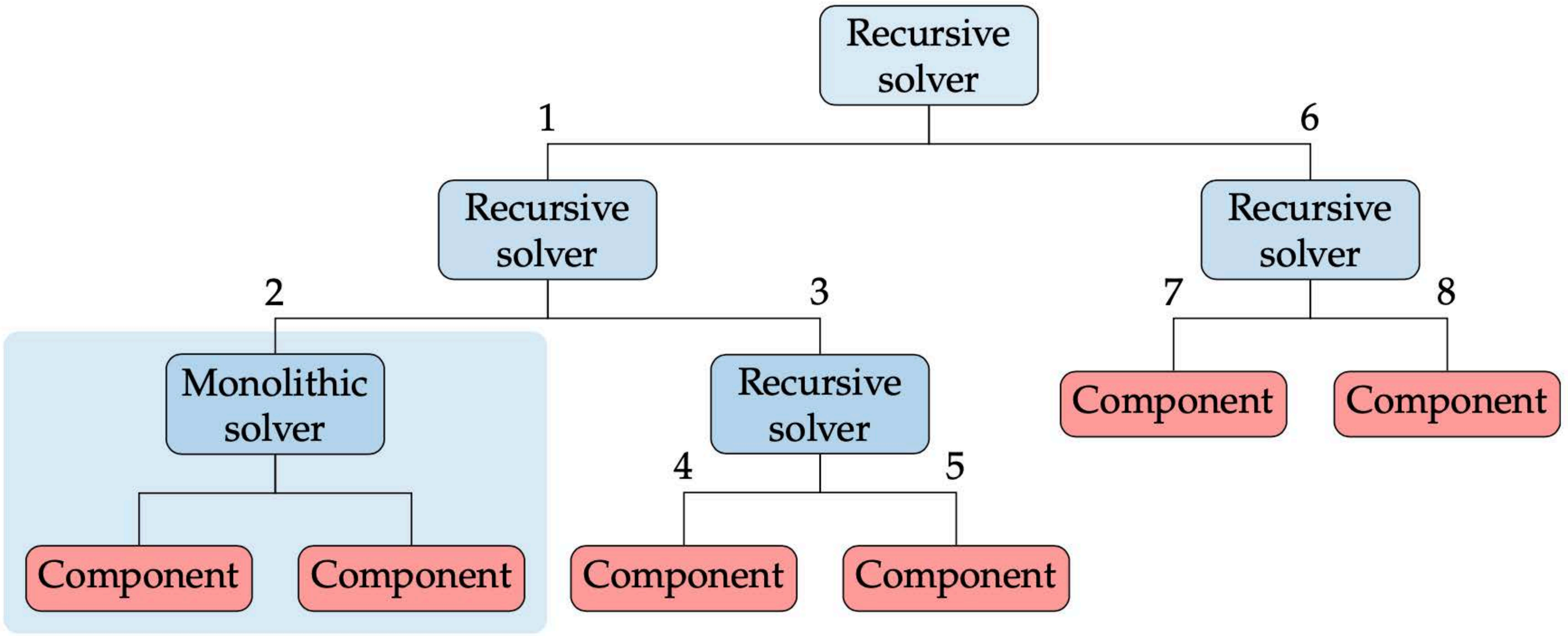


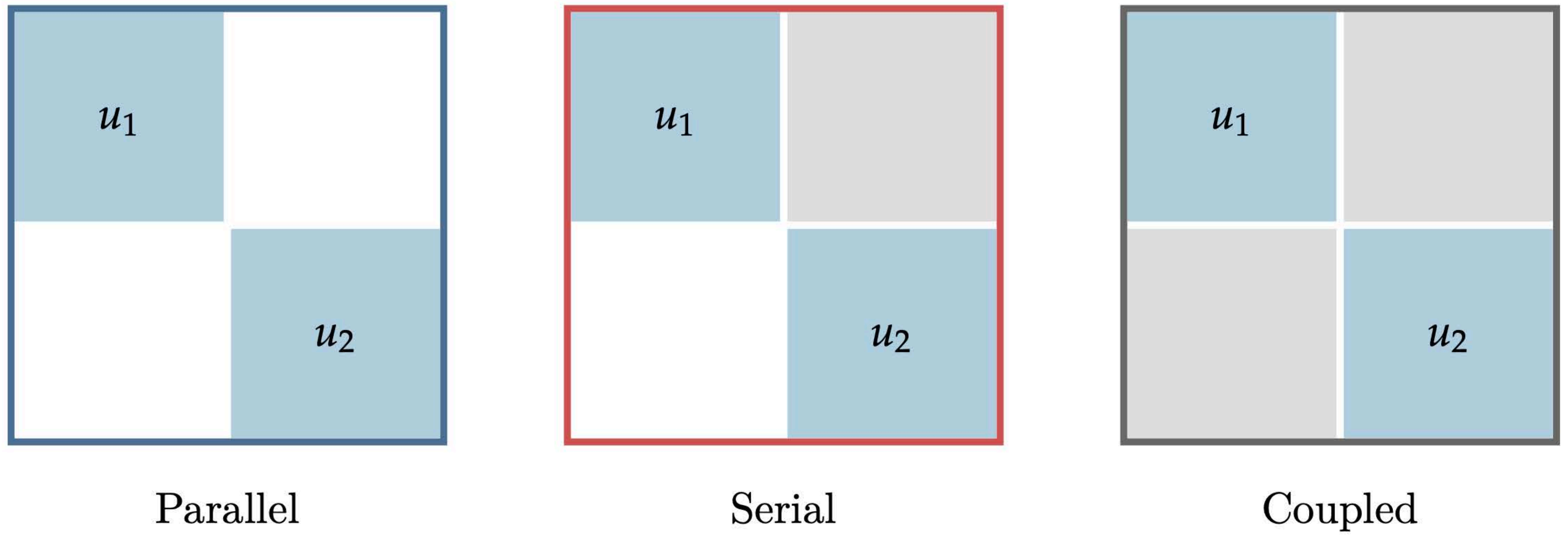


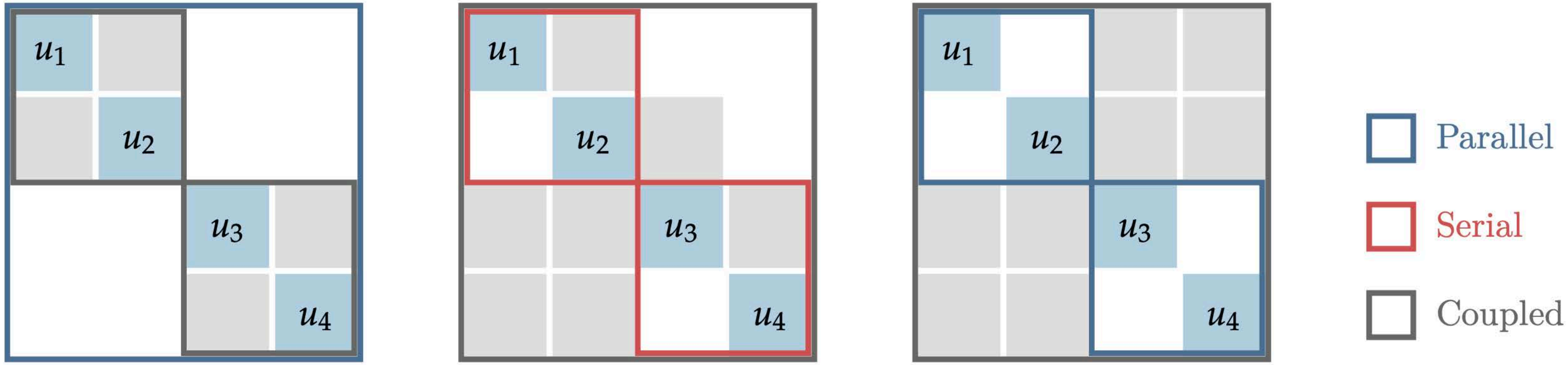


Hwang, Lee, Cutler, and Martins. **Large-scale multidisciplinary optimization of a small satellite's design and operation**. Journal of Spacecraft and Rockets, 2014

MAUD includes hierarchical solvers and coupled derivatives for complex systems



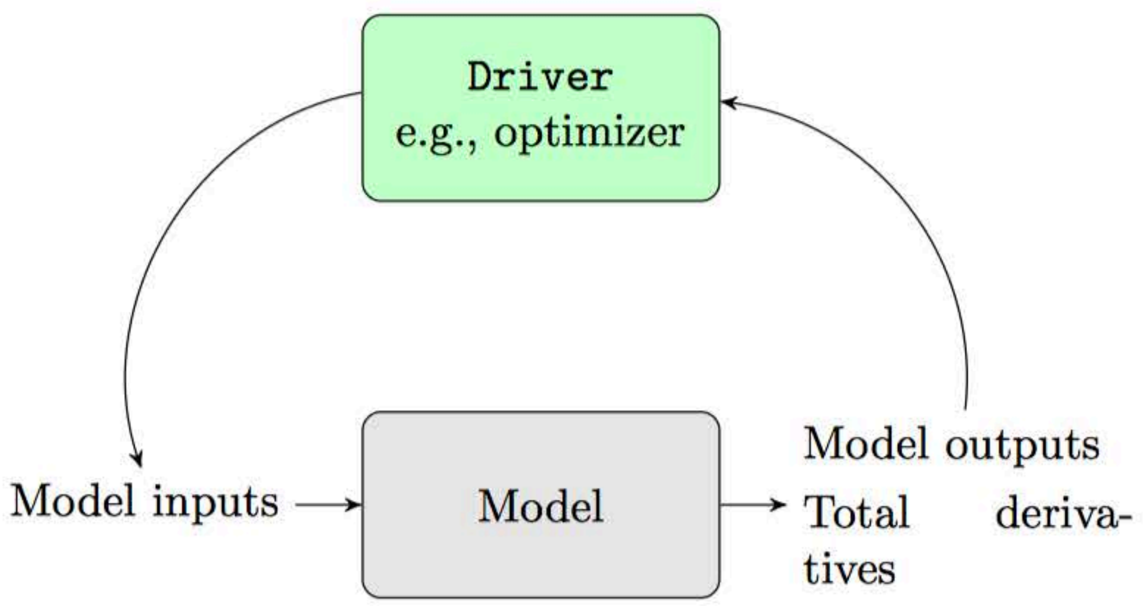


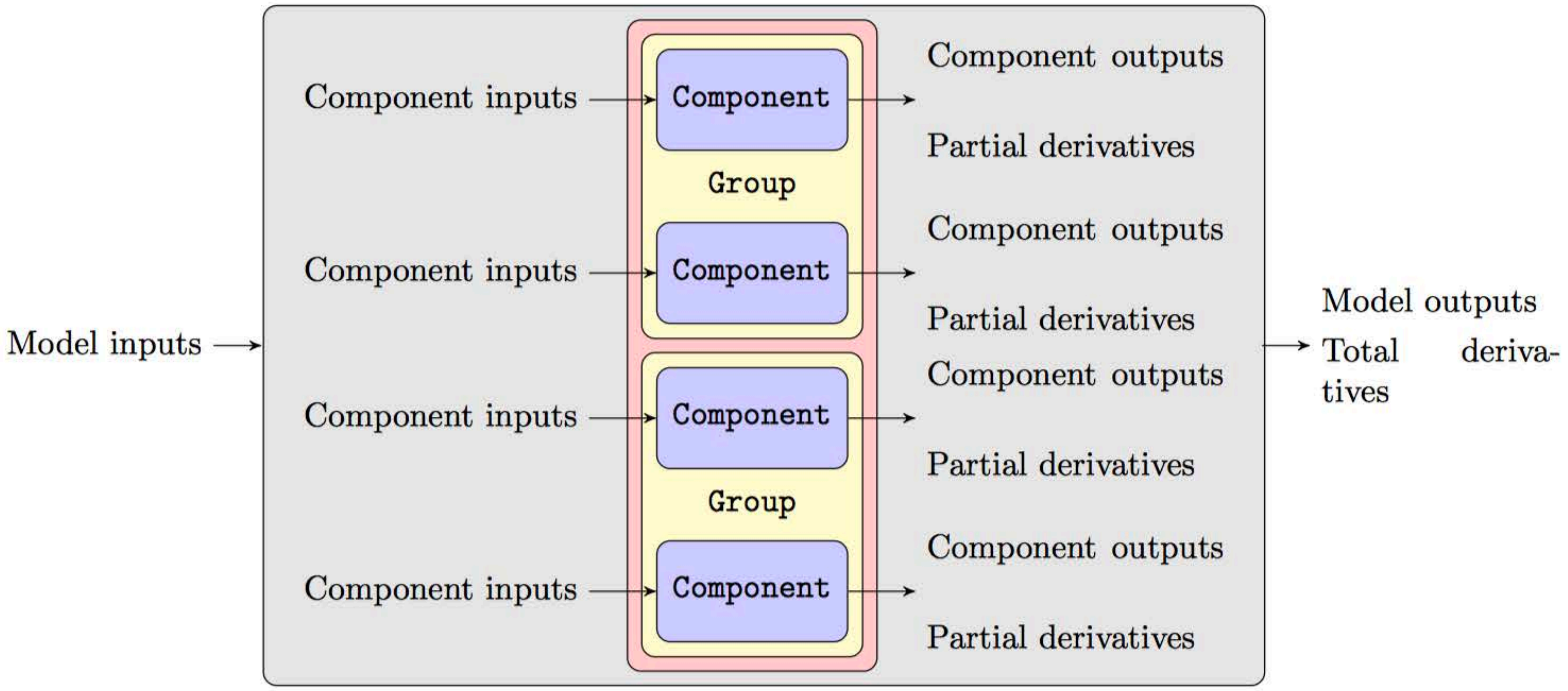


Hwang and Martins. **A computational architecture for coupling heterogeneous numerical models and computing coupled derivatives**. *ACM Transactions on Mathematical Software*, 2018

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

MAUD was implemented in openMDAO



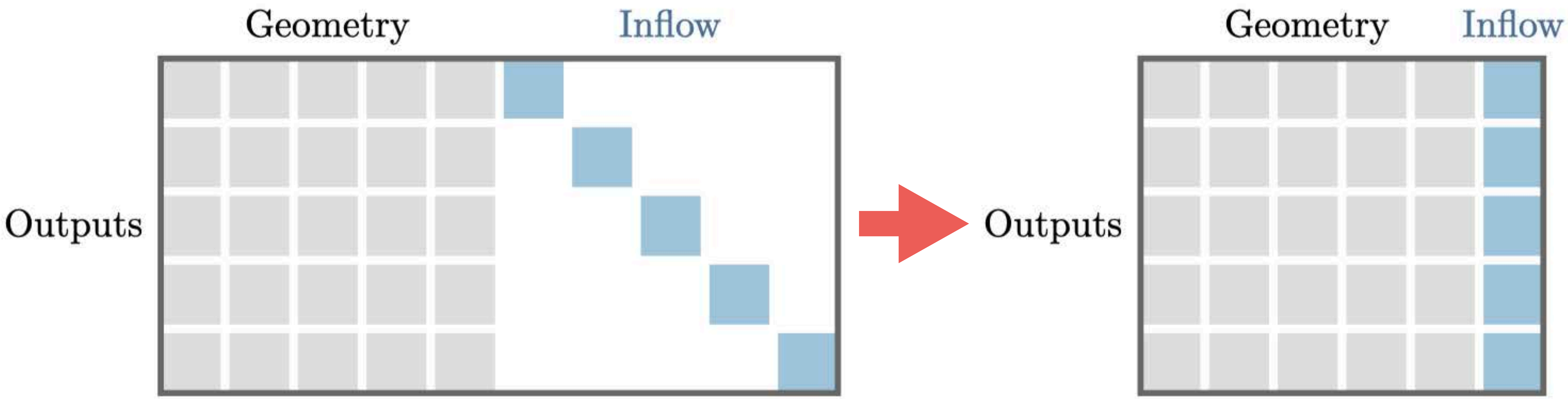


- Developed at NASA Glenn
- Python-based
- Open-source framework
- Facilitates the coupling multiple models and optimization
- Efficient coupled solution via Newton-type methods
- Efficient coupled adjoint derivative computation

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

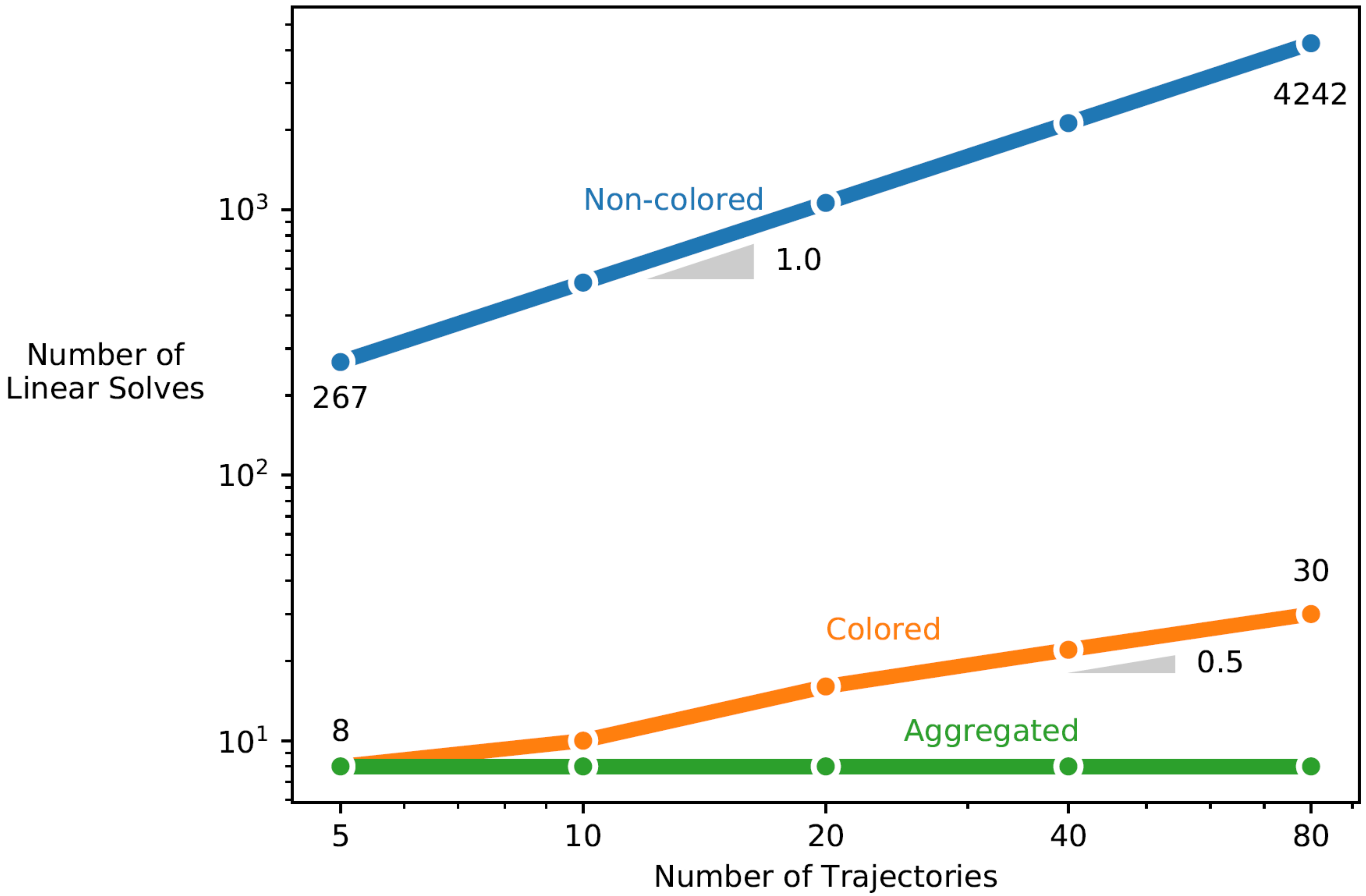
OpenMDAO performs Jacobian coloring automatically

- Consider a wind turbine optimization problem with geometry design variables and multiple inflow conditions
- The geometry variables affect the performance at all inflow conditions
- But each inflow condition only affects the performance for that condition



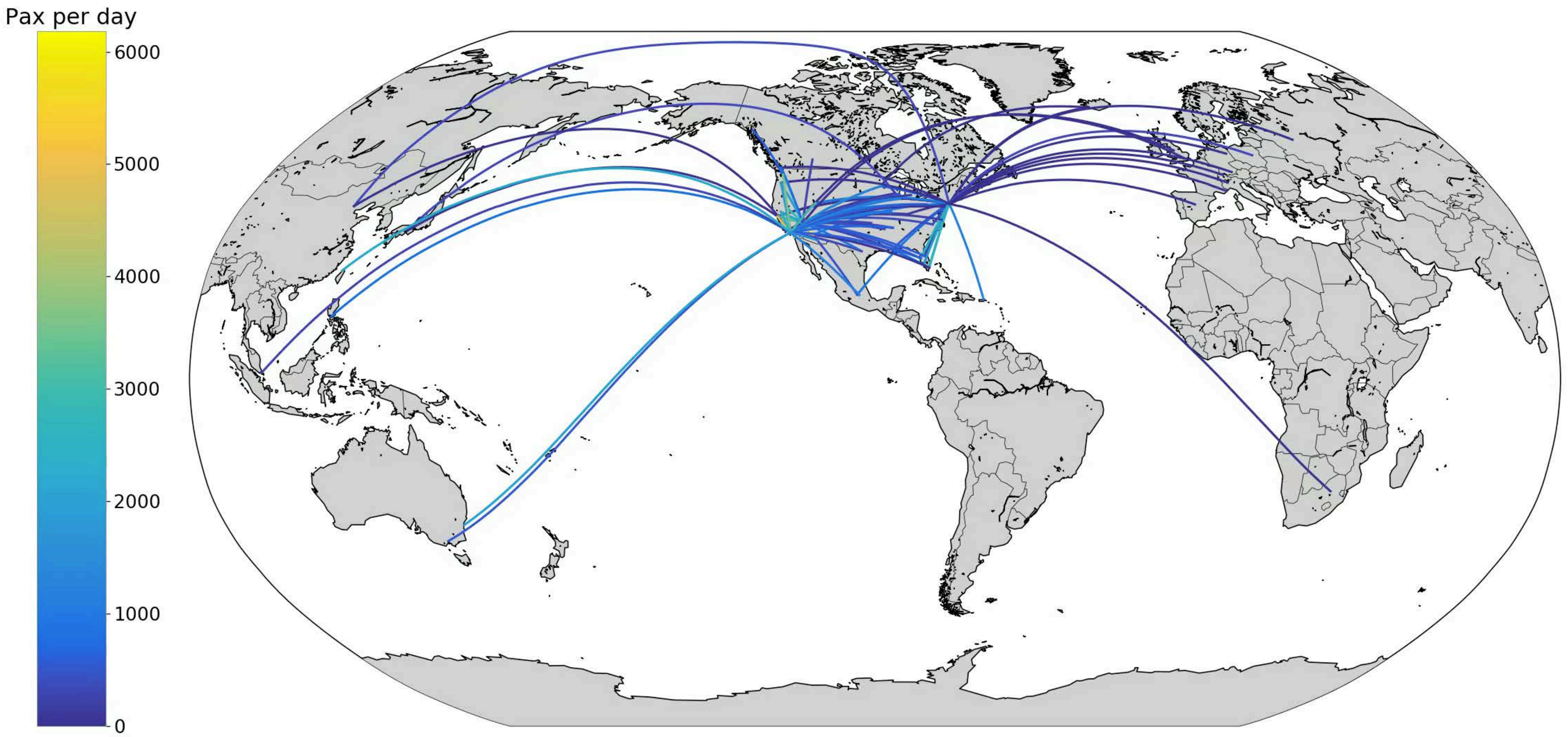
Gray, Hwang, Martins, Moore, and Naylor. **OpenMDAO: An open- source framework for multidisciplinary design, analysis, and optimization**. *Structural and Multidisciplinary Optimization*, 2019

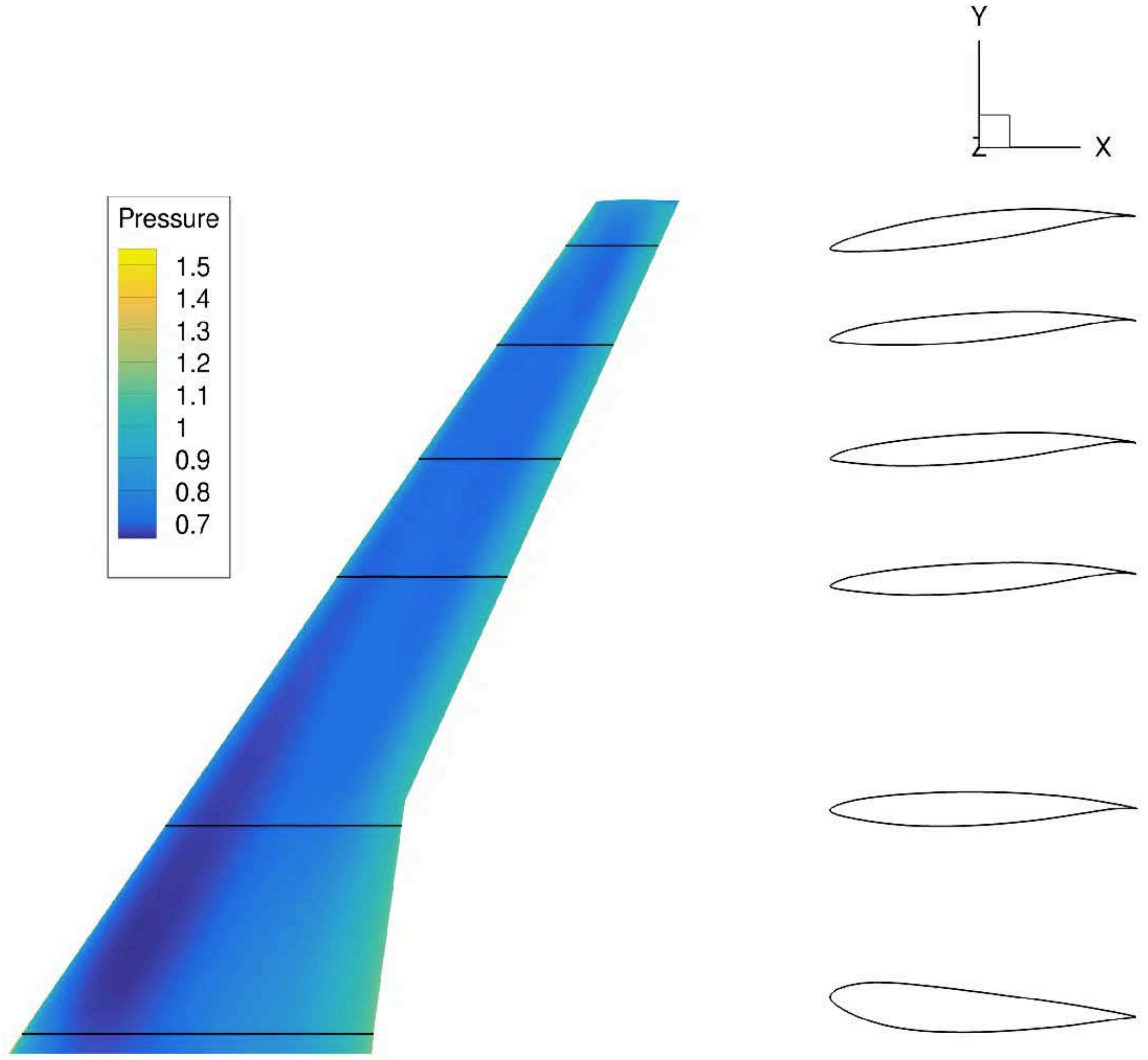
Coloring greatly increases the efficiency of derivative computation for sparse systems

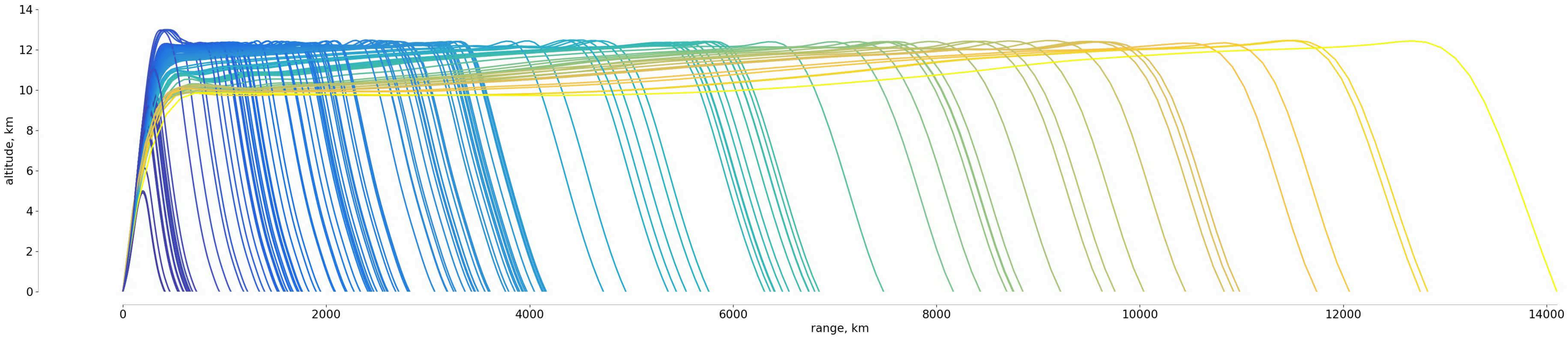


Gray, Hwang, Martins, Moore, and Naylor. **OpenMDAO: An open- source framework for multidisciplinary design, analysis, and optimization**. *Structural and Multidisciplinary Optimization*, 2019

It is possible to optimize wing, trajectory, and allocation together thanks to OpenMDAO

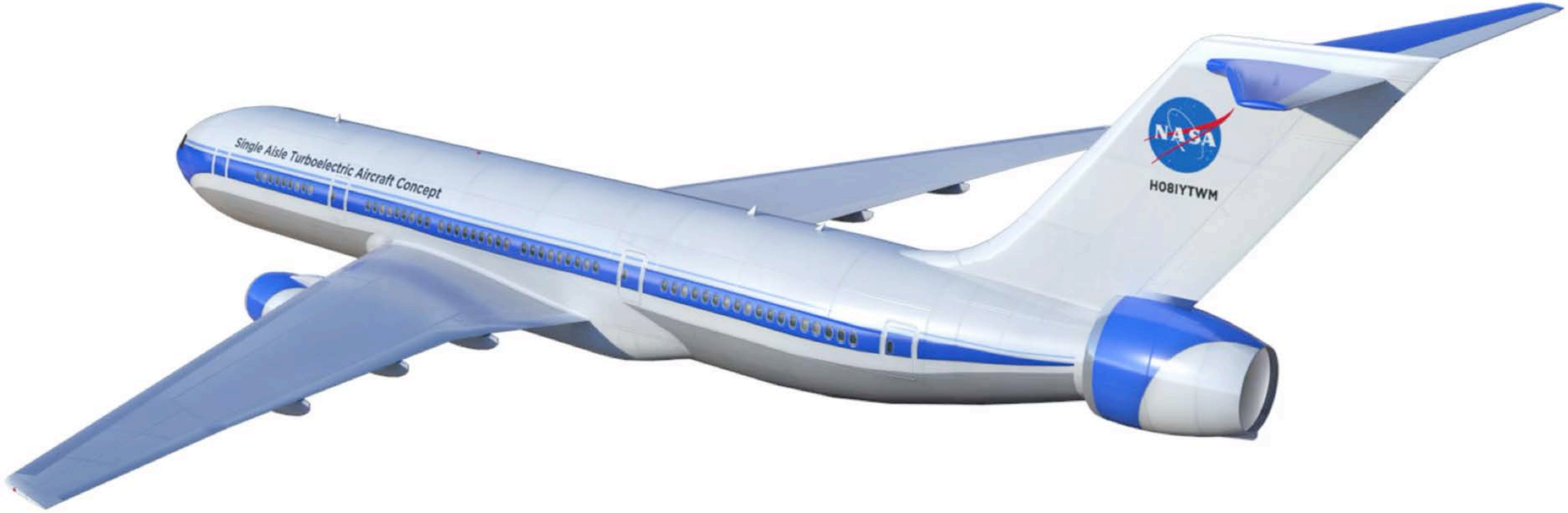






Hwang, Jasa, and Martins. **High-fidelity design-allocation optimization of a commercial aircraft maximizing airline profit**. *Journal of Aircraft*, 2019.

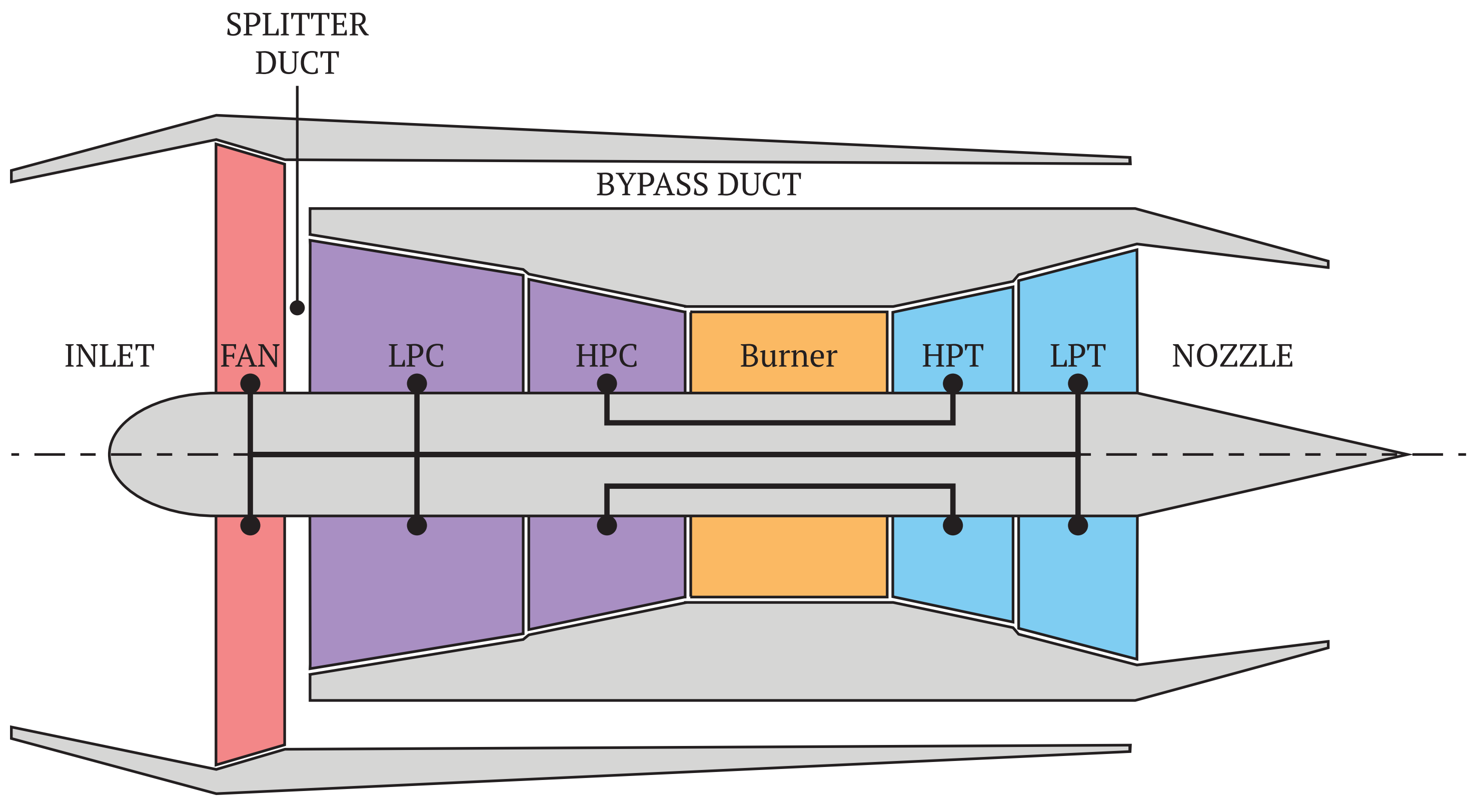
Airframe-propulsion integration demands CFD-based MDO

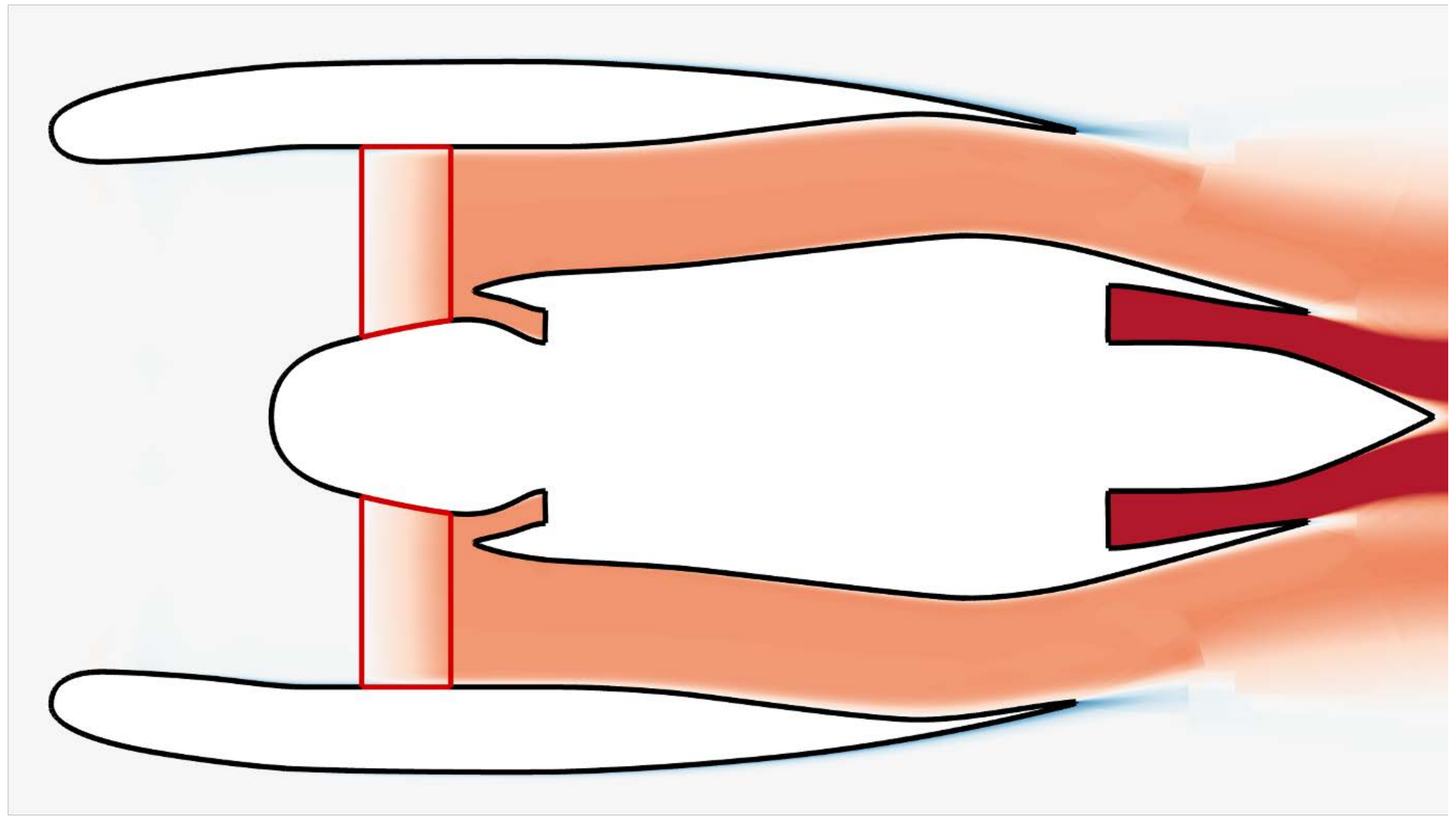


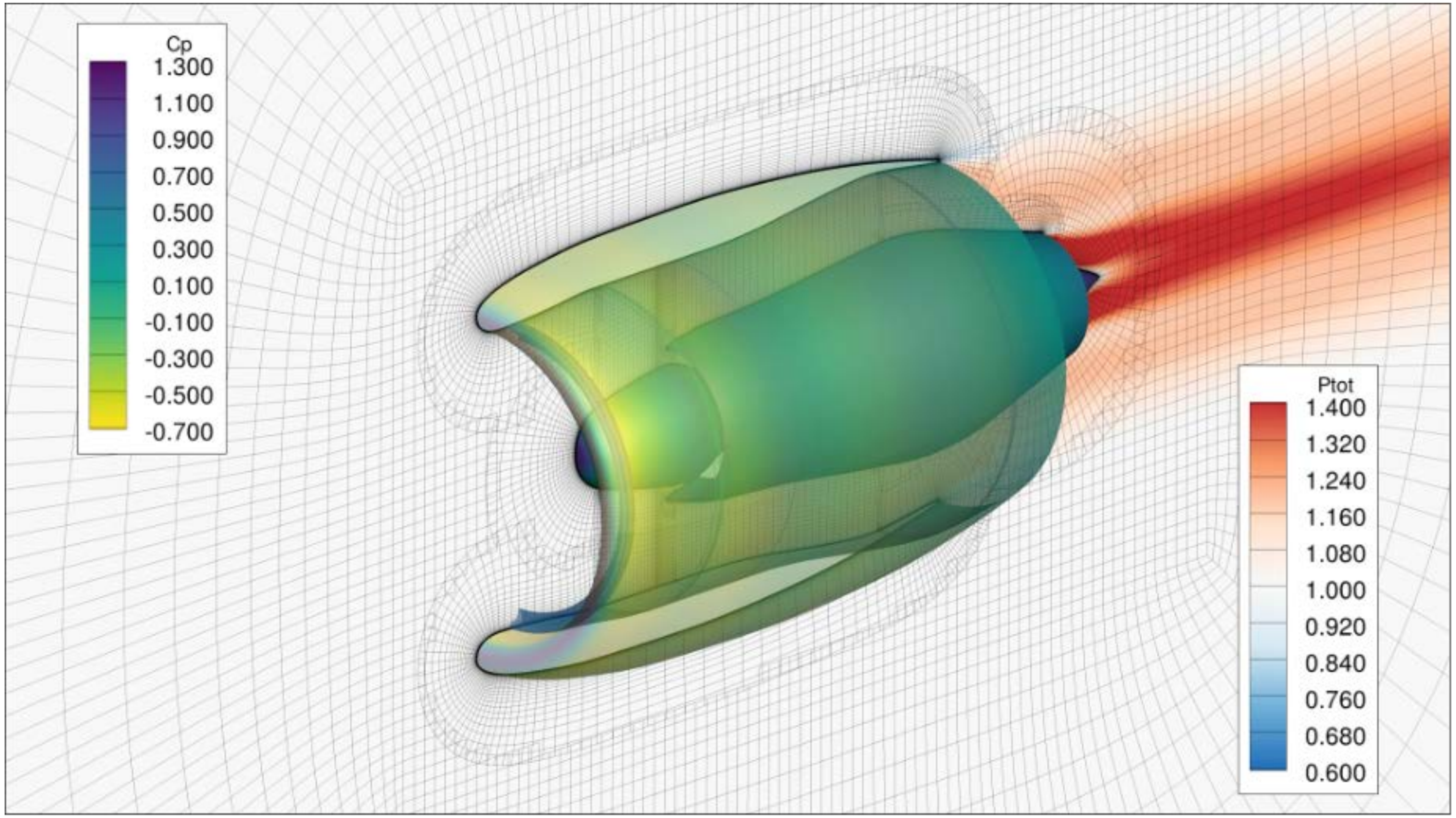
This is the STARC–ABL concept

Yildirim, Gray, Mader, and Martins. **Aeropropulsive design optimization of a boundary layer ingestion system**. *AIAA 2019- 3455*.

Coupled CFD-based optimization with pyCycle turbofan model



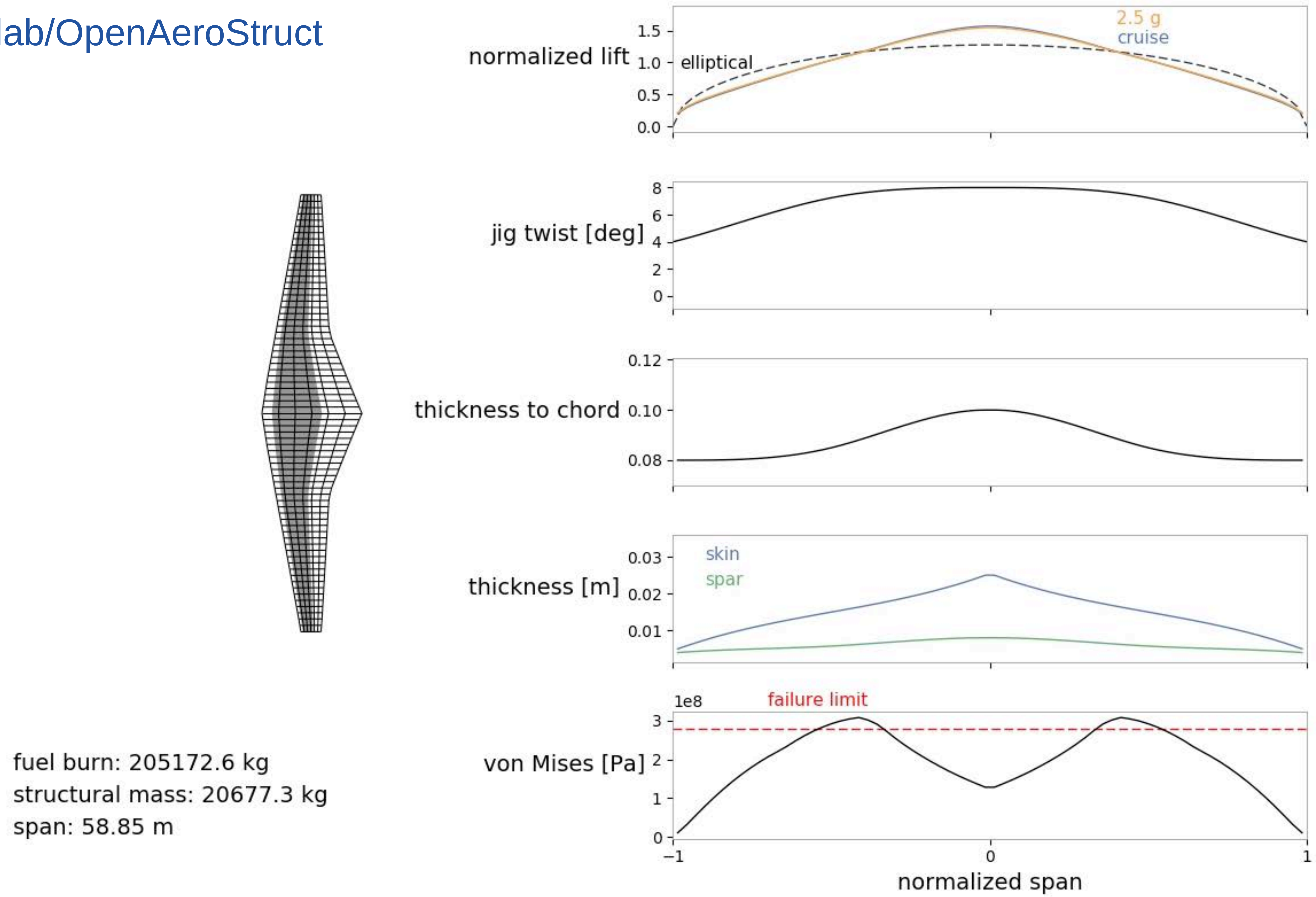




Lamkin, Yildirim, and Martins. **Coupled aeropropulsive analysis and optimization of a high bypass turbofan engine**. *ICAS*, 2022.

OpenAeroStruct is a low-fidelity OpenMDAO-based version of MACH

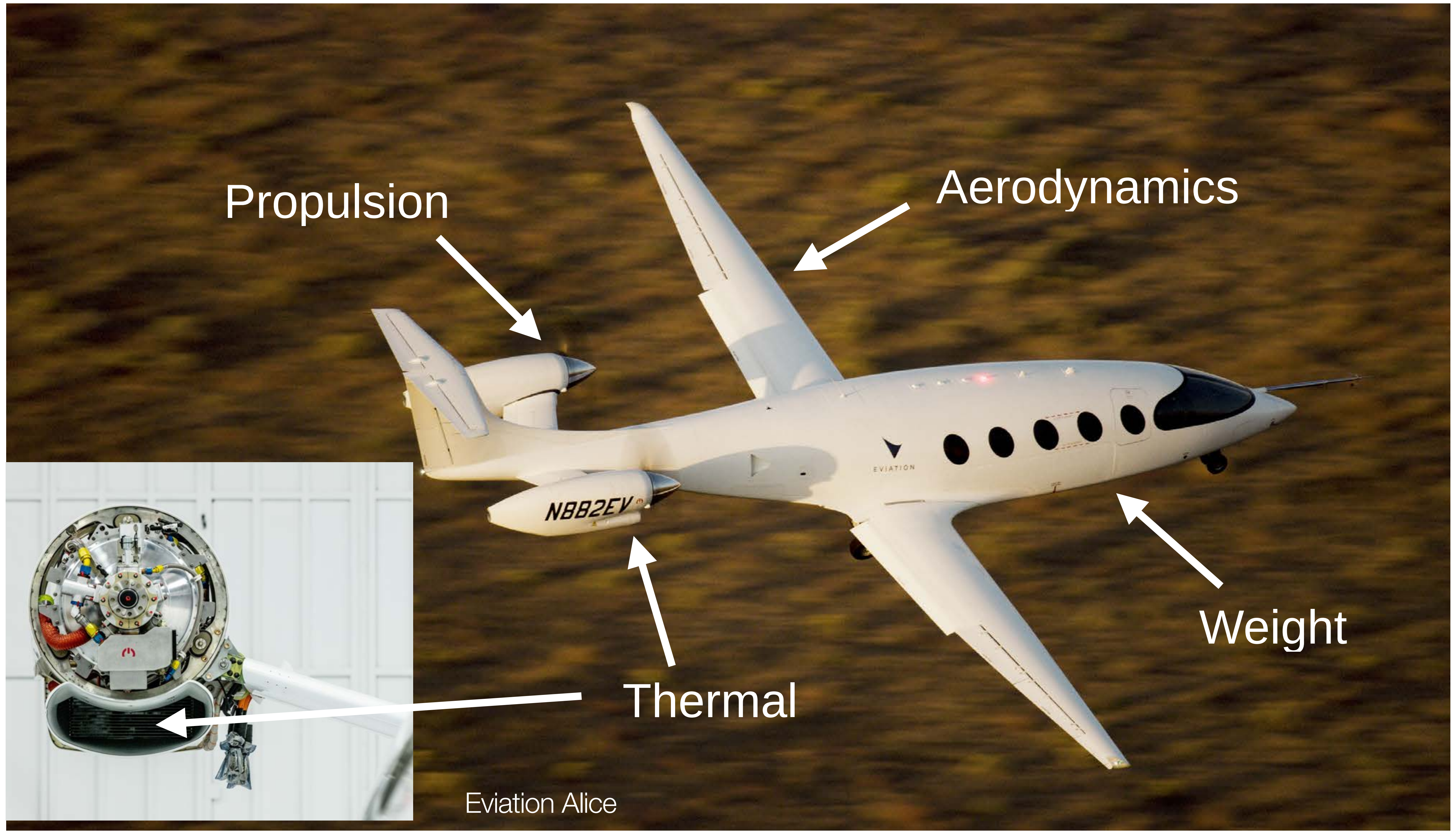
https://github.com/mdolab/OpenAeroStruct

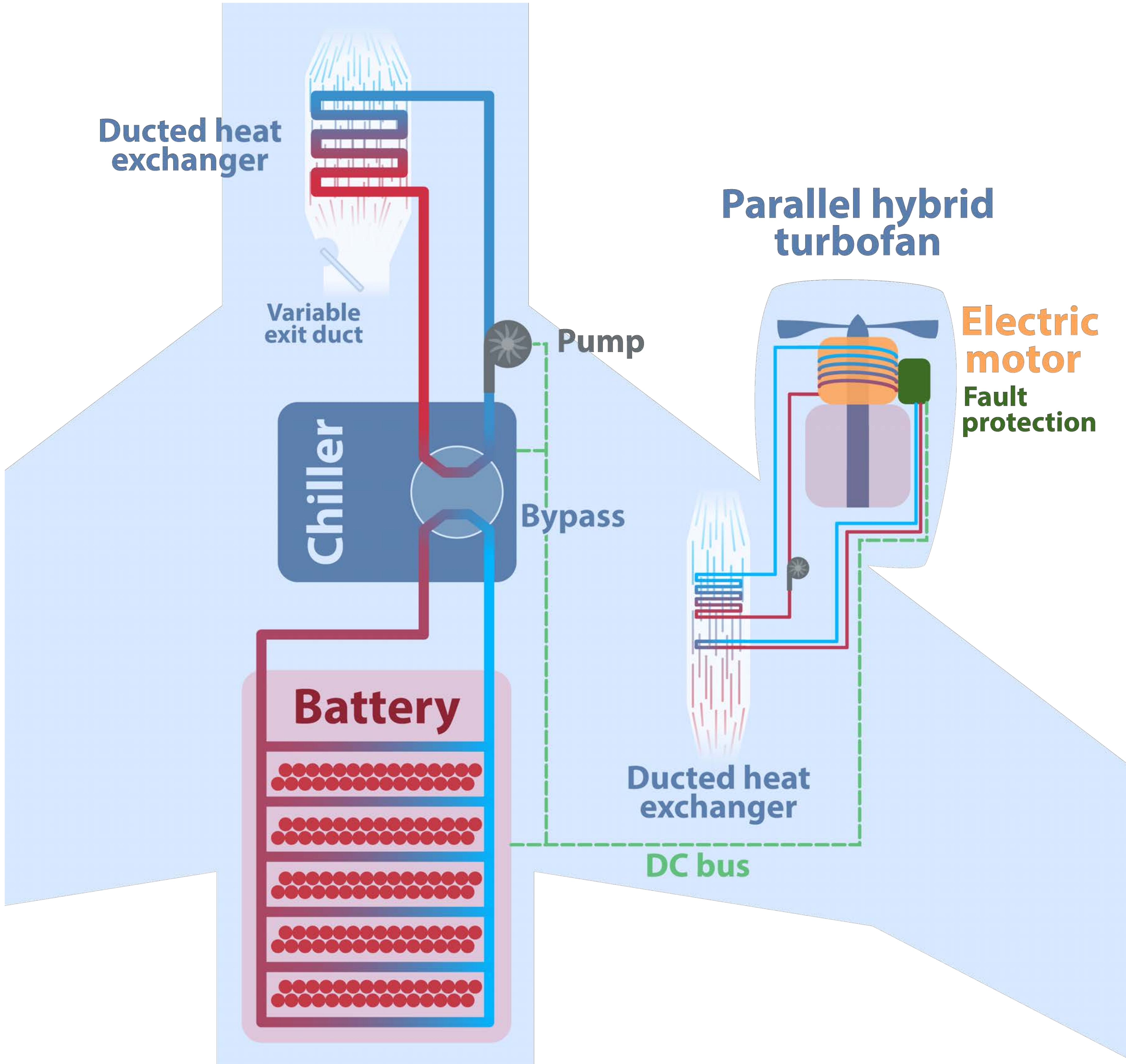


Jasa, Hwang, and Martins. **Open-source coupled aerostructural optimization using Python**. *Structural and Multidisciplinary Optimization*, 2018

Chauhan and Martins. **Low-fidelity aerostructural optimization of aircraft wings with a simplified wingbox model using OpenAeroStruct**. 2018

OpenConcept developed for electric aircraft systems

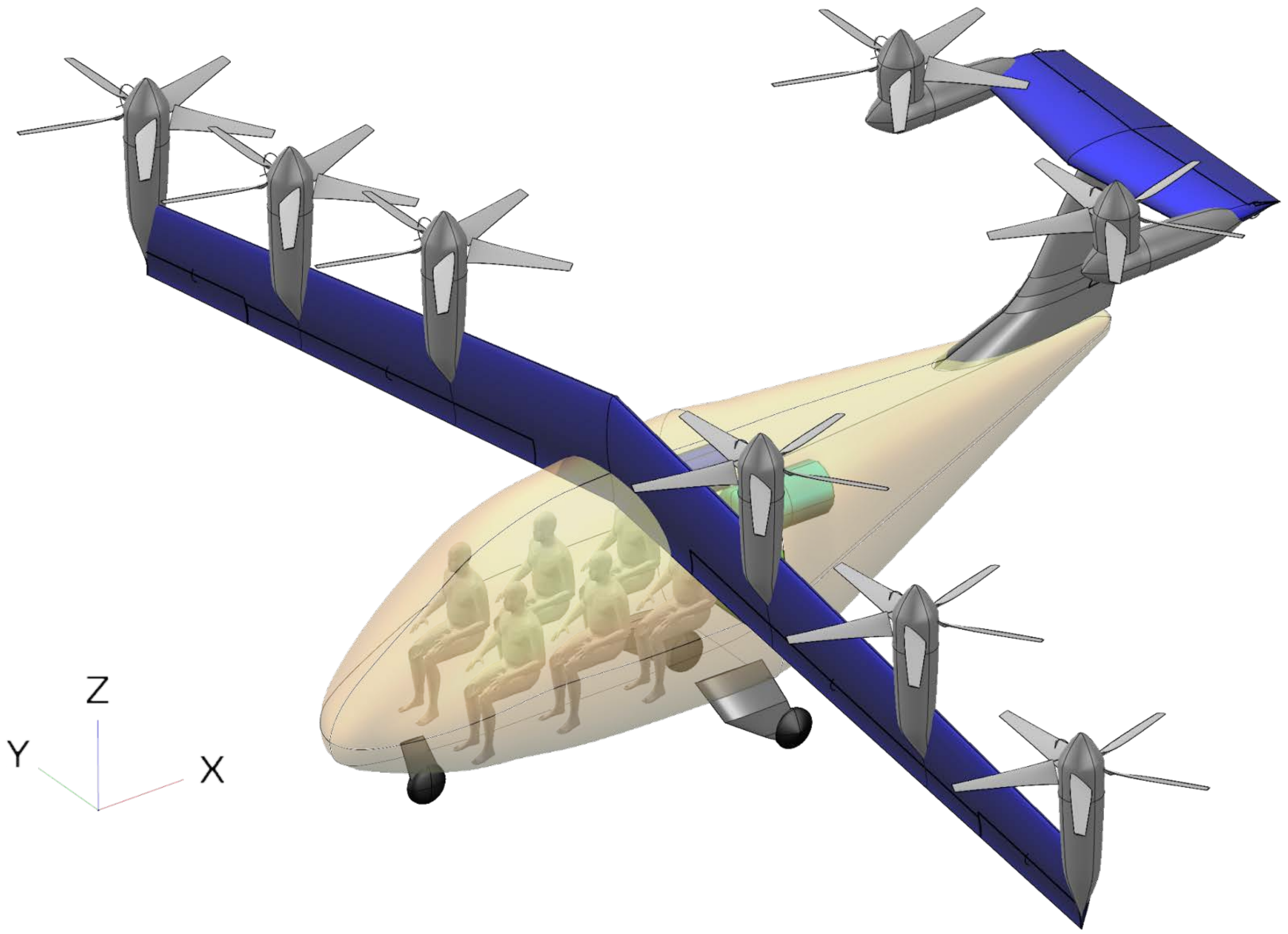


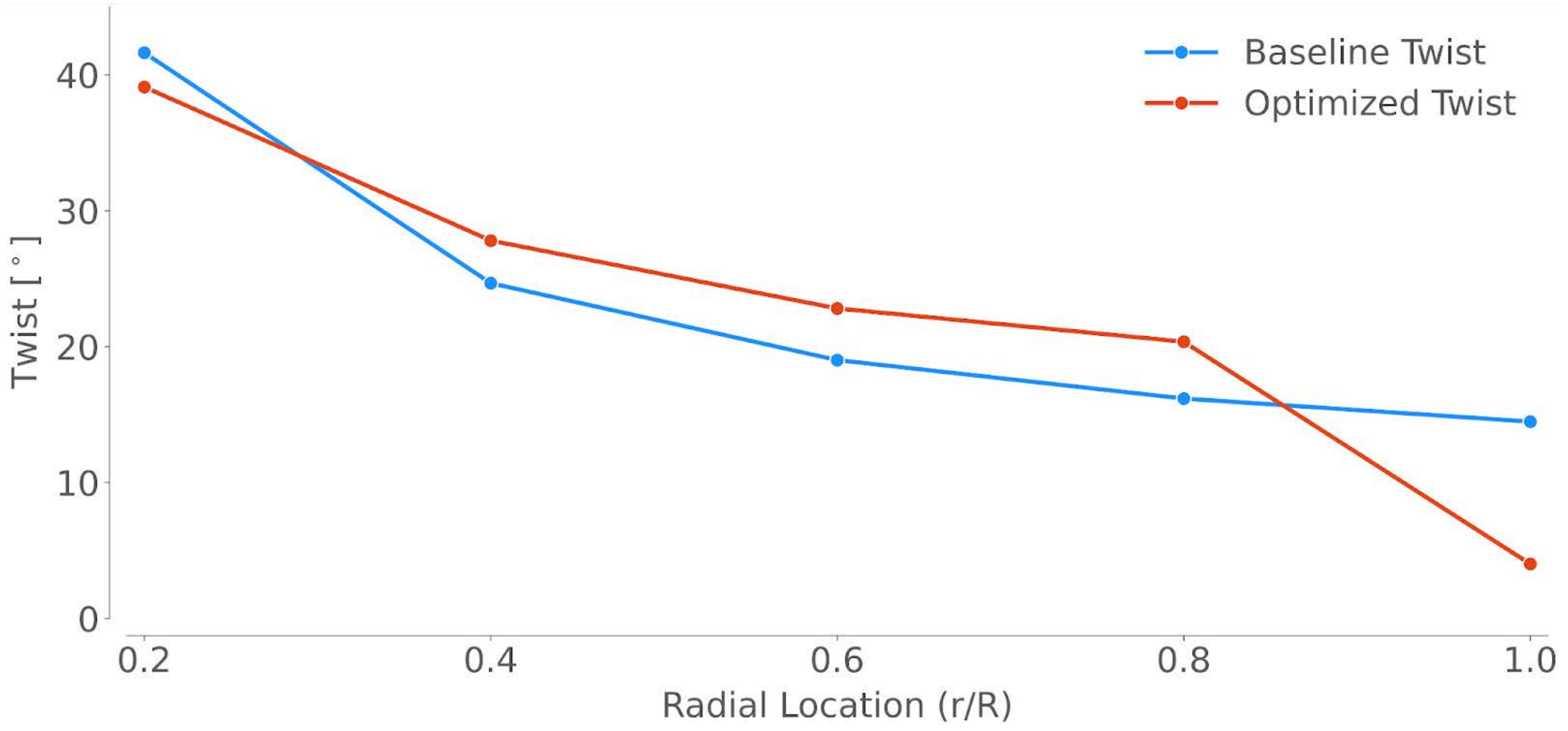


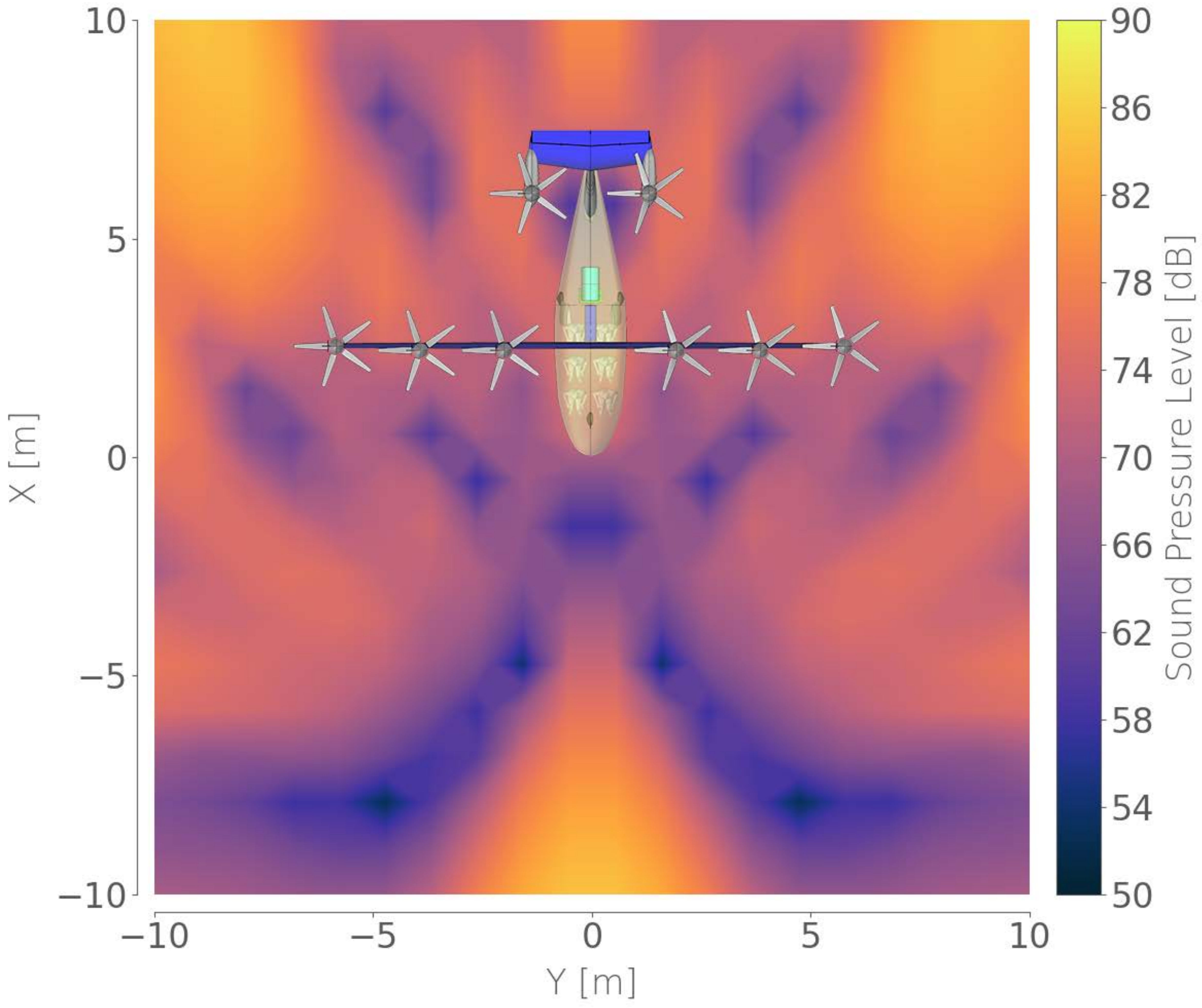
Brelje, Martins. **Electric, hybrid, and turboelectric fixed-wing aircraft: A review of concepts, models, and design approaches**. *Progress in Aerospace Sciences*, 2019

Adler, Brelje, and Martins. **Thermal management system optimization for a parallel hybrid aircraft considering mission fuel burn**. *Aerospace*, 2022.

Rotor optimization of NASA Tiltwing vehicle subject to noise constraints

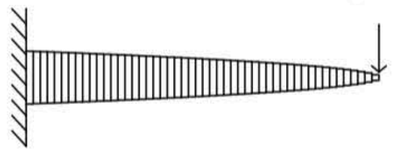
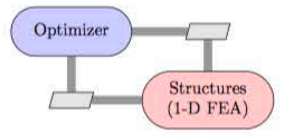
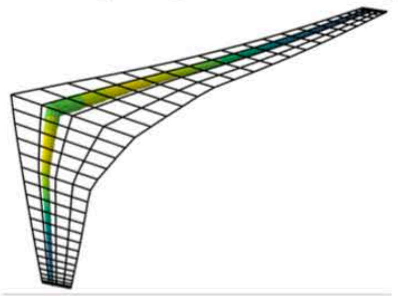
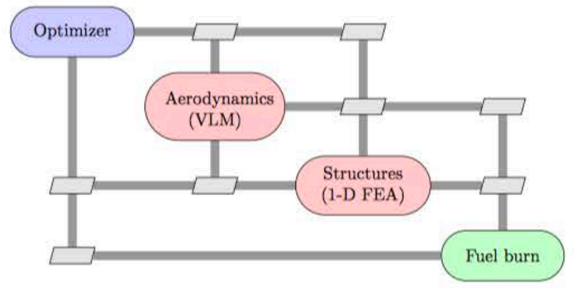
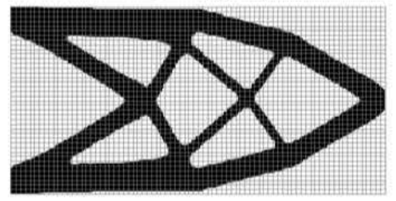
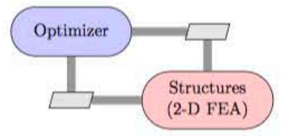
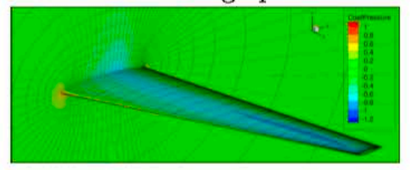
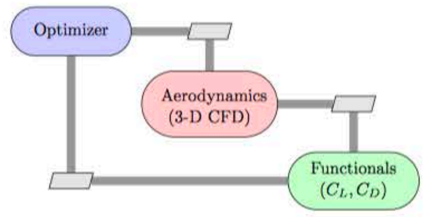
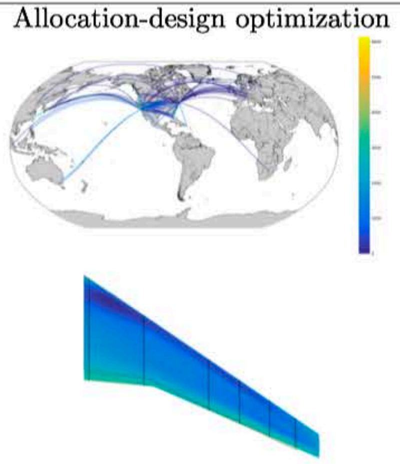
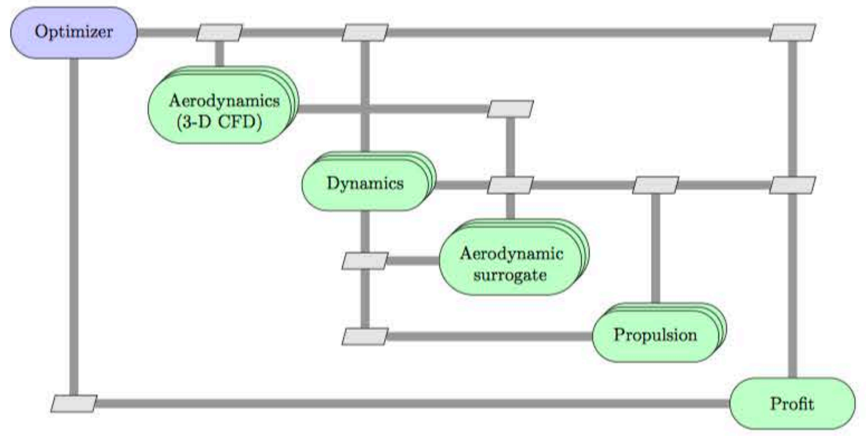
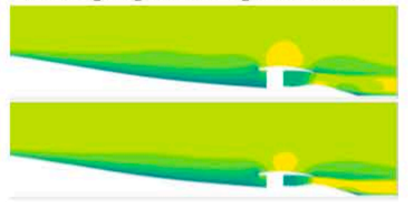
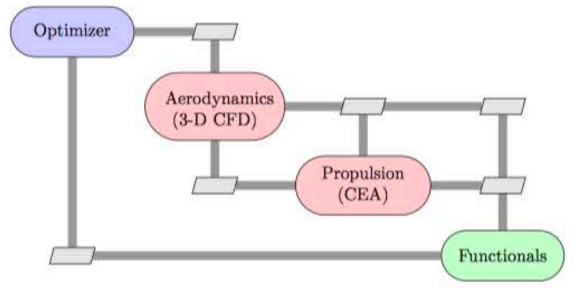
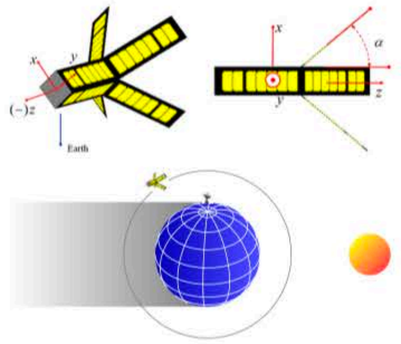
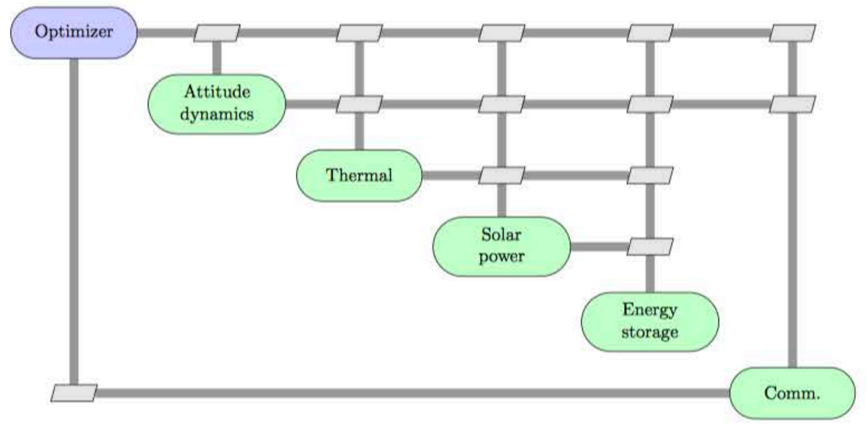
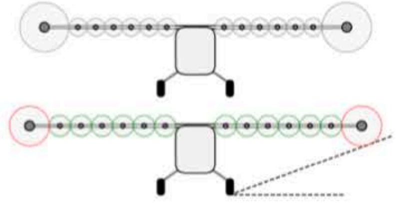
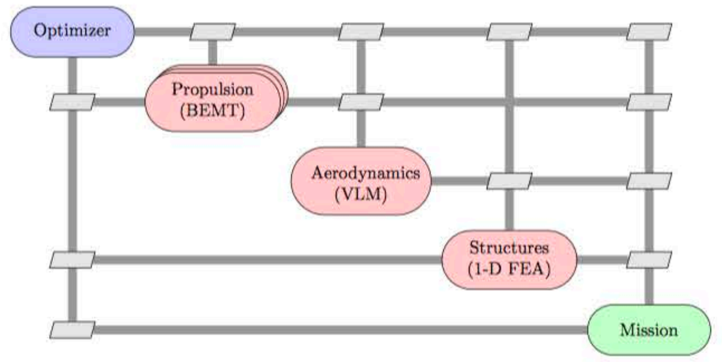






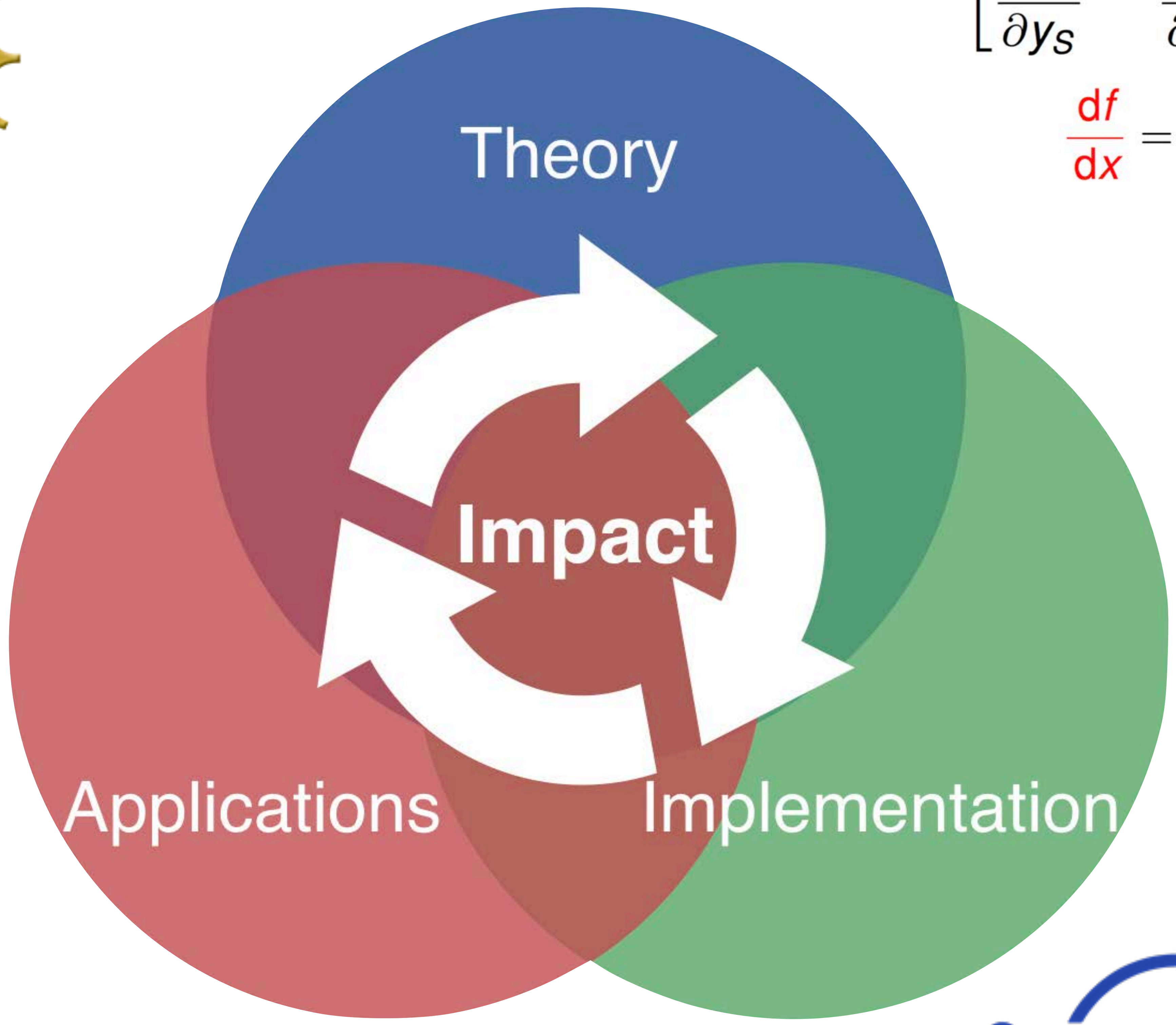
Pacini et al. **Towards Efficient Aerodynamic and Aeroacoustic Optimization for Urban Air Mobility Vehicle Design.** AIAA SciTech 2022.

Other OpenMDAO applications

Problem	Model structure	Design variables	Objective	Constraints
Cantilever beam thickness opt. 		thickness distribution	compliance	weight
Low-fidelity wing aerostructural opt. 		thickness & twist dist.	fuel burn	trim stress
Structural topology optimization 		element densities	compliance	mass fraction
RANS-based wing optimization 		shape variables	drag coefficient	lift coefficient
Allocation-design optimization 		wing variables; altitude profiles; cruise Machs; allocation vars.	profit	wing geometry; thrust limits; demand & fleet limits
Aero-propulsive optimization 		inlet shape variables	fuel burn	trim
CubeSat MDO 		solar panel angle; antenna angle; num. radiators; power distribution; attitude profile; solar panel controls	data downloaded	batt. charge rate; batt. charge level
CTOL electric aircraft MDO 		altitude prof.; velocity prof.; prop RPM profs.; prop chord; prop twist; prop diam.; wing twist; beam thickness	range	average speed; eqs. of motion; max. power; min. torque; ground clear.; tip speed; wing failure

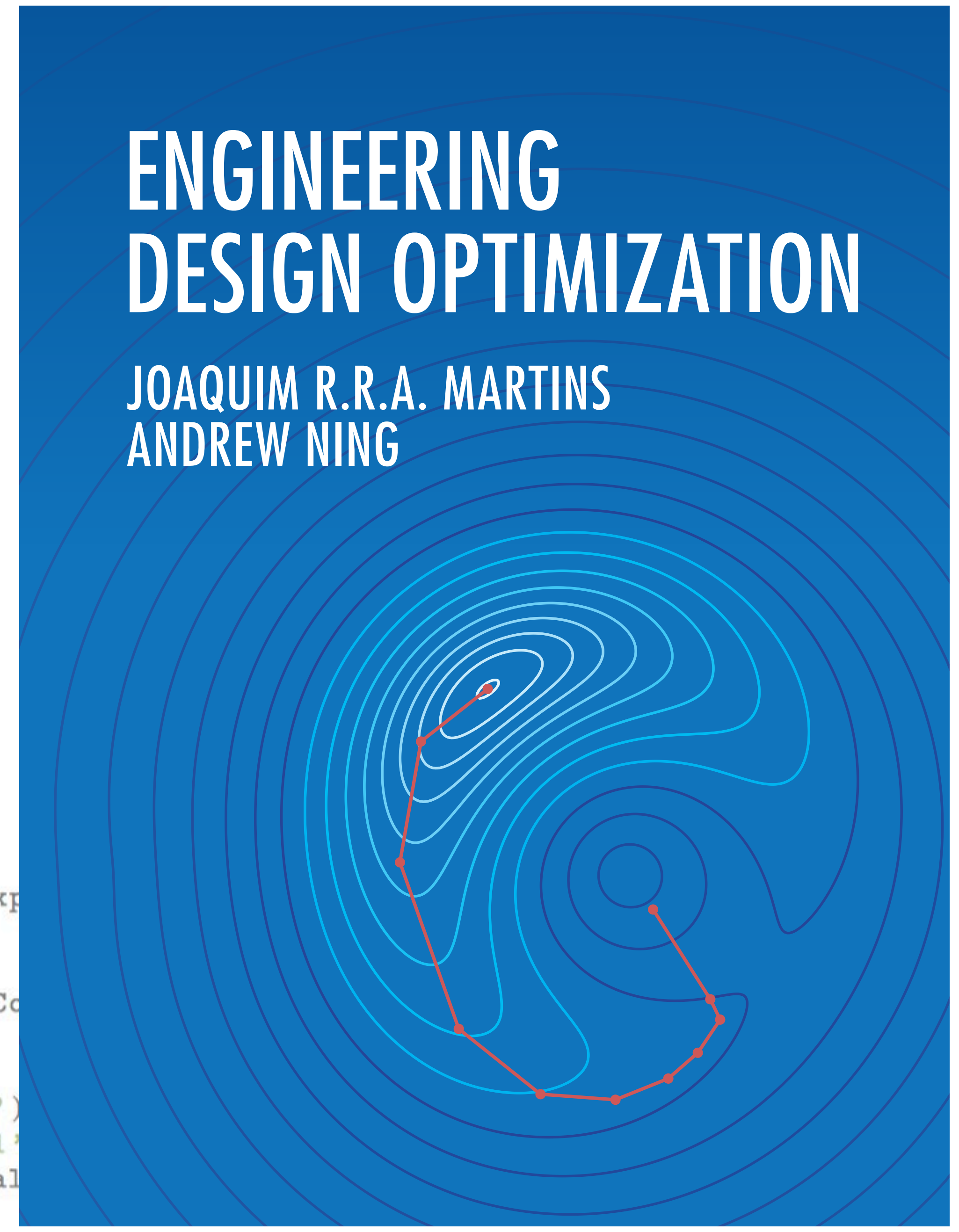
Gray, Hwang, Martins, Moore, and Naylor. **OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization**. *Structural and Multidisciplinary Optimization*, 2019

Many of the implemented theoretical developments are now available as open-source software



$$\begin{bmatrix} \dfrac{\partial R_A}{\partial y_A}^T & \dfrac{\partial R_S}{\partial y_A}^T \\ \dfrac{\partial R_A}{\partial y_S}^T & \dfrac{\partial R_S}{\partial y_S}^T \end{bmatrix} \psi = \dfrac{\partial f}{\partial y}^T$$

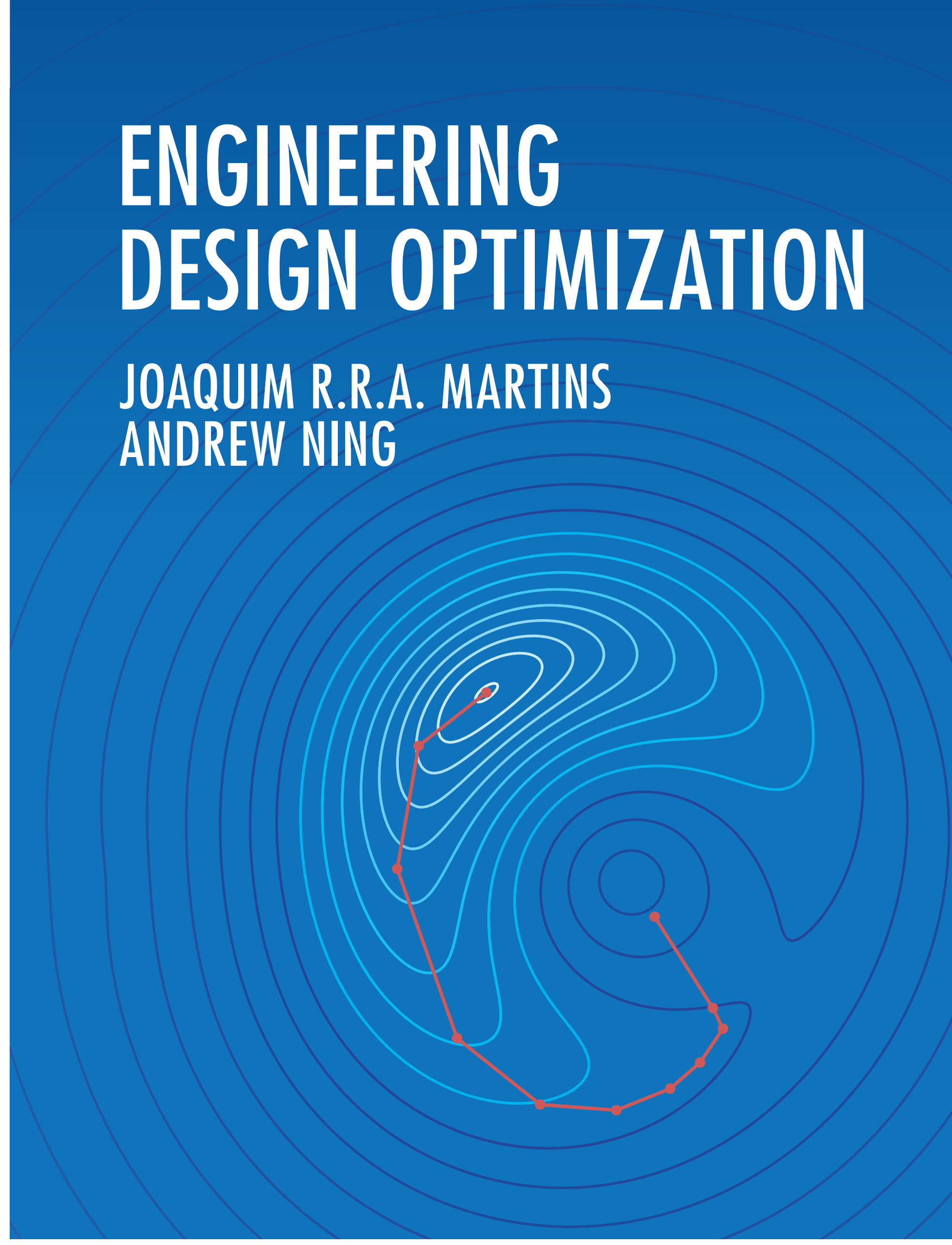
$$\frac{df}{dx} = \frac{\partial f}{\partial x} - \psi^T \frac{\partial R}{\partial x}$$




```
import numpy as np

from openmdao.api import Exp


class Discipline1(ExplicitCo

    def setup(self):
        self.add_input('y2'
        self.add_output('y1'
        self.declare_partial

    def compute(self, inputs, outputs):
        outputs['y1'] = inputs['y2'] ** 2

    def compute_partials(self, inputs, partials):
        partials['y1', 'y2'] = 2 * inputs['y2']


class Discipline2(ImplicitComponent):

    def setup(self):
        self.add_input('x')
        self.add_input('y1')
        self.add_output('y2')
        self.declare_partials('y2', 'x')
        self.declare_partials('y2', 'y1')
        self.declare_partials('y2', 'y2')

    def linearize(self, inputs, outputs, partials):
        partials['y2', 'x'] = -outputs['y2']
        partials['y2', 'y1'] = (-outputs['y2'] * np.exp(-inputs['y1'] *
                                outputs['y2']))
        partials['y2', 'y2'] = (-inputs['y1'] *
                                np.exp(-inputs['y1'] * outputs['y2']) - inputs['x'])
```


GO FORTH AND OPTIMIZE!

Useful sections for OpenMDAO users



- Sec. 1.2: Problem formulation
- Sec. 1.4.1-1.4.3: Gradient-based vs gradient-free optimization algorithms (see Fig. 1.23 for cost comparison), local vs global search, mathematical vs heuristic approaches
- Sec. 1.5: How to select an optimization algorithm
- Sec. 3.3: Introduction to residuals of governing equations and explicit/implicit functions
- Sec. 3.6: Overview of governing equation solvers
- Chapter 4 header and Sec. 4.1-4.2: Basic idea of gradient-based optimization
- Example 4.18 and Sec. 4.6: Comparison of gradient-based algorithms, summary
- Chapter 5 header and Sec. 5.1: Basic idea of constrained optimization
- Sec. 5.8: Summary of constrained optimization
- Chapter 6: Derivatives. Whole chapter recommended, but especially: Sec. 6.7 (implicit analytic methods including adjoint), Sec. 6.8 (sparse Jacobians), Sec. 6.9 (UDE), and Sec. 6.10 (summary)
- Sec. 7.1: When to use gradient-free algorithms
- Chapter 13 header and 13.1: Introduction to MDO
- Sec. 13.2: Coupled models and solvers (includes MAUD in Sec. 13.2.6)
- Sec. 13.3.3: Implicit analytic coupled derivatives
- Tip 13.4: OpenMDAO
- Sec. 13.6: MDO summary