



Using **openMDAO** to  
Solve Design Optimization Problems 10 Times Faster

Justin S. Gray and Eric S. Hendricks  
*NASA Glenn Research Center*

This work was supported by NASA ARMD's  
Transformational Tools and Technology Project



An open-source MDO framework, specializing in gradient based optimization with analytic derivatives.

Our goal is to solve design problems with many analyses and a mix both high- and low-fidelity analyses.

<https://github.com/openmdao/openmdao>

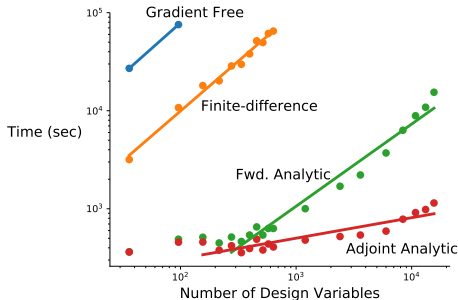
# Gradient-based optimization does have some limitations, but you should still use it!

- *Gradient-based optimizers only find local optima!*  
they are 10x-10000x faster, so we can wrap multi-start or gradient-free methods around them
- *Gradient-based optimizers can't handle discrete variables!*  
try relaxation or MINLP algorithms that mix gradient-based and integer programming\*
- *Gradient-free for life!*  
ok... you win. OpenMDAO supports gradient-free methods too

\*S. Roy, William A. Crossley, B. K. Stanford, K. T. Moore, and J. S. Gray, A Mixed Integer Efficient Global Optimization Algorithm with Multiple Infill Strategy Applied to a Wing Topology Optimization Problem, in AIAA SciTech Forum, San Diego, CA, 2019.

# Gradient-based optimization with analytic derivatives is orders of magnitude faster than other options

Computational Cost vs # of Design Variables



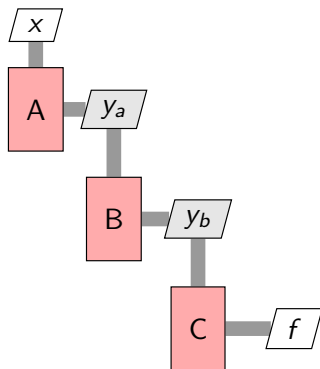
- 100x-10000x speedup for aerodynamic shape optimization vs. gradient-free [1]
- at least 5x-10x speedup vs. finite-difference [2]

[1] Z. Lyu, Xu, Z., and Martins, J. R. R. A., "Benchmarking Optimization Algorithms for Wing Aerodynamic Design Optimization", in 8th International Conference on Computational Fluid Dynamics (ICCFD8), Chengdu, China, 2014.

[2] J. S. Gray, T. A. Hearn, K. T. Moore, J. Hwang, J. Martins, and A. Ning, "Automatic Evaluation of Multidisciplinary Derivatives Using a Graph-Based Problem Formulation in OpenMDAO", in 15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2014.

# Challenge: Implementing analytic derivatives is hard, time-consuming, and highly problem specific

This one is pretty simple:

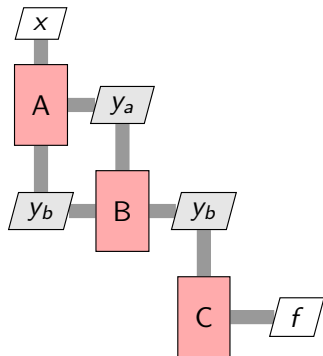


Derivative of  $f$  with respect to  $x$ :

$$\frac{df}{dx} = \frac{\partial f}{\partial y_b} \frac{\partial y_b}{\partial y_a} \frac{\partial y_a}{\partial x}$$

# Challenge: Implementing analytic derivatives is hard, time-consuming, and highly problem specific

Adding coupling makes it a bit harder:



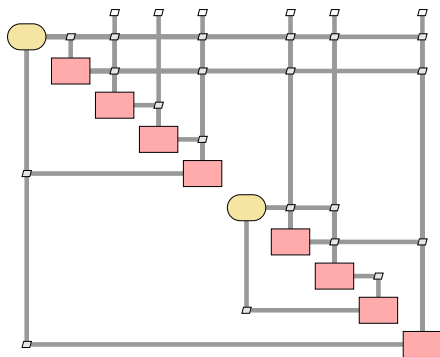
Derivative of  $f$  with respect to  $x$ :

$$\frac{df}{dx} = \frac{\partial f}{\partial y_b} \frac{\partial y_b}{\partial y_a} \frac{\partial y_a}{\partial x} + \frac{\partial f}{\partial y_b} \frac{dy_b}{dx} + \frac{\partial f}{\partial y_a} \frac{dy_a}{dx}$$

The  $\frac{dy_b}{dx}$  and  $\frac{dy_a}{dx}$  terms  
can be found with an adjoint solve

# Challenge: Implementing analytic derivatives is hard, time-consuming, and highly problem specific

here is the structure of model used with trajectory analysis for a UAM aircraft:

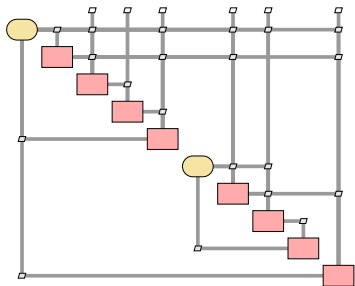


Find  $\frac{df}{dx}$ :

solvers within solvers??

let's be honest,  
we're not doing this by hand!

# Analytic derivatives require 2 steps: computing partials of each analysis and combining them into total derivatives



- 1) finding partial derivatives:  
by hand, FD/CS, algorithmic differentiation are all good options!
- 2) computing total derivatives  
from the partials:  
can be totally automated by the unified derivative equation ( $UDE$ )\*

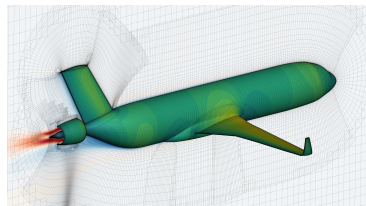
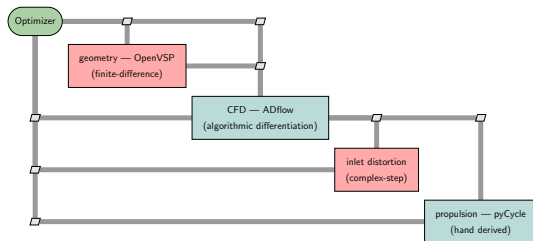
\*Martins JRR, Hwang JT. "Review and Unification of Methods for Computing Derivatives of Multidisciplinary Computational Models". AIAA Journal. 2013 ;51(11):2582–2599. DOI 10.2514/1.J052184

\*\*Hwang JT, Martins JRR. A computational architecture for coupling heterogeneous numerical models and computing coupled derivatives. ACM TOMS. 2018 ;44(4). DOI 10.1145/3182393



# OpenMDAO computes totals with mixed serial/parallel analyses and mixed analytic/numerical partials

Model for boundary layer ingestion (BLI) propulsion system mixes CFD (parallel) with 1D cycle analysis (serial) and combines all 4 ways to compute partials. Total derivatives are computed automatically.



See “Aeropropulsive Design Optimization of a Boundary Layer Ingestion System”  
Wed 2:30-3:00 (Milan)

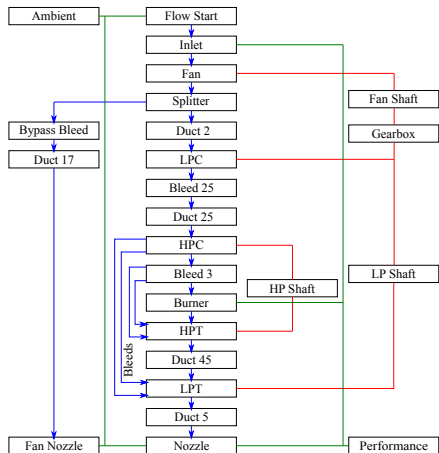
The flexibility and efficiency of OpenMDAO's analytic derivative capability makes it more than an MDO framework.

It can also be used as a low-level library for building new stand-alone analysis tools with analytic derivatives.

pyCycle and NPSS are essentially identical analysis libraries, just implemented on top of different foundational software layers

This presented the opportunity to benchmark an optimization using an existing analysis against a close cousin written in OpenMDAO

# We built the same high-bypass turbofan model in pyCycle and NPSS to compare optimization performance



- advanced technology reference cycle model\*
- 4-point coupled analysis:
  - sea level static (SLS)
  - top of climb (TOC)
  - rolling take off (RTO)
  - cruise (CRZ)

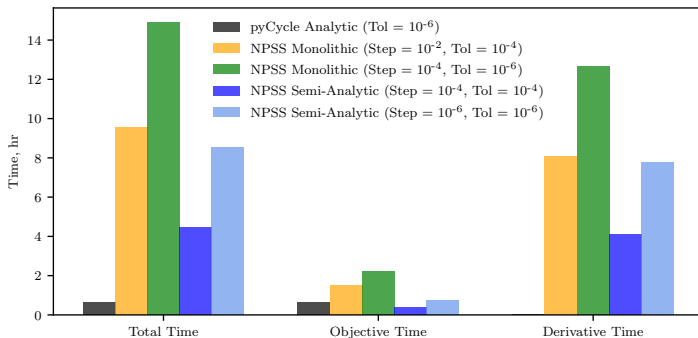
\*Jones, S. M., Haller, W. J., and Tong, M. T., An N+3 Technology Level Reference Propulsion System, May 2017. NASA TM-2017-219501

We verified that pyCycle and NPSS can produce the same answer within solver tolerance

Point	Parameter	NPSS	pyCycle	Error
TOC	TSFC, lbm/hr/lbf	0.43856	0.43860	0.00868%
	Net Thrust, lbf	6125.95	6126.66	0.01171%
	Bypass Ratio	23.74639	23.74246	0.01656%
RTO	TSFC, lbm/hr/lbf	0.27364	0.27368	0.01303%
	Net Thrust, lbf	22800.00	22800.00	0.00001%
	Bypass Ratio	25.58064	25.57669	0.01546%
SLS	TSFC, lbm/hr/lbf	0.16632	0.16635	0.01910%
	Net Thrust, lbf	28620.84	28620.84	0.00000%
	Bypass Ratio	27.34323	27.33845	0.01745%
CRZ	TSFC, lbm/hr/lbf	0.44075	0.44079	0.00883%
	Net Thrust, lbf	5513.35	5514.00	0.01171%
	Bypass Ratio	24.34212	24.33803	0.01681%

- largest errors were under 0.02%
- we considered this an exact match to within solver tolerance

# It was at least 10x faster to optimize using pyCycle with analytic derivatives than NPSS with finite-differences

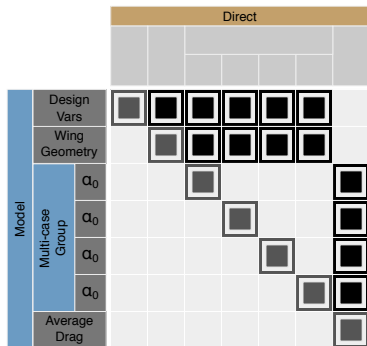


- many combinations of solver tolerance and FD step size were tested
- tried methods to finite-difference NPSS: monolithic FD over the whole model and semi-analytic derivatives with FD-partial

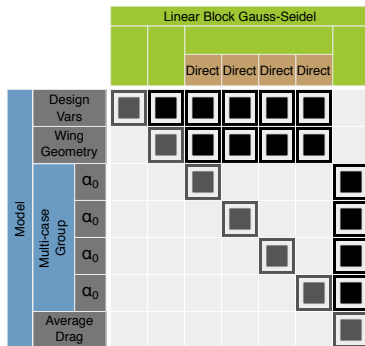
pyCycle uses a direct factorization to solve for total derivatives, but we also have other algorithms that work well for different problems

# A mixed solver setup can speed up total derivative solves for multipoint problems

Multipoint OpenAeroStruct problems see large computational savings from more complex solver setups



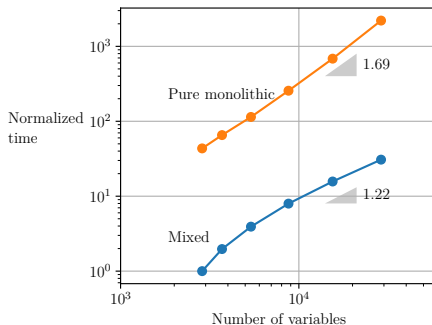
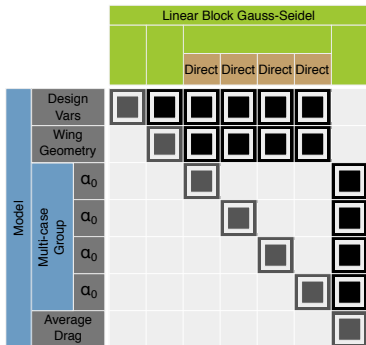
Monolithic solver setup



Mixed solver setup




# The mixed solver setup is 10x faster for a multipoint aerostructural optimization



The mixed solver setup performs even better as the number of points in the *Multi-case Group* increases

# You can get a 10x-1000x speed up too!

First US OpenMDAO workshop: Cleveland OH, October 28, 29 2019!

 makes it much easier to leverage gradient based optimization with analytic derivatives

- Get all of the advanced analytic derivative functionality “for free” if you build your tools and models in OpenMDAO
- Develop stand alone analysis tools on top of OpenMDAO and integrate them into your own framework

— or —

- Use OpenMDAO as a reference implementation and pull these methods into your own tools and frameworks