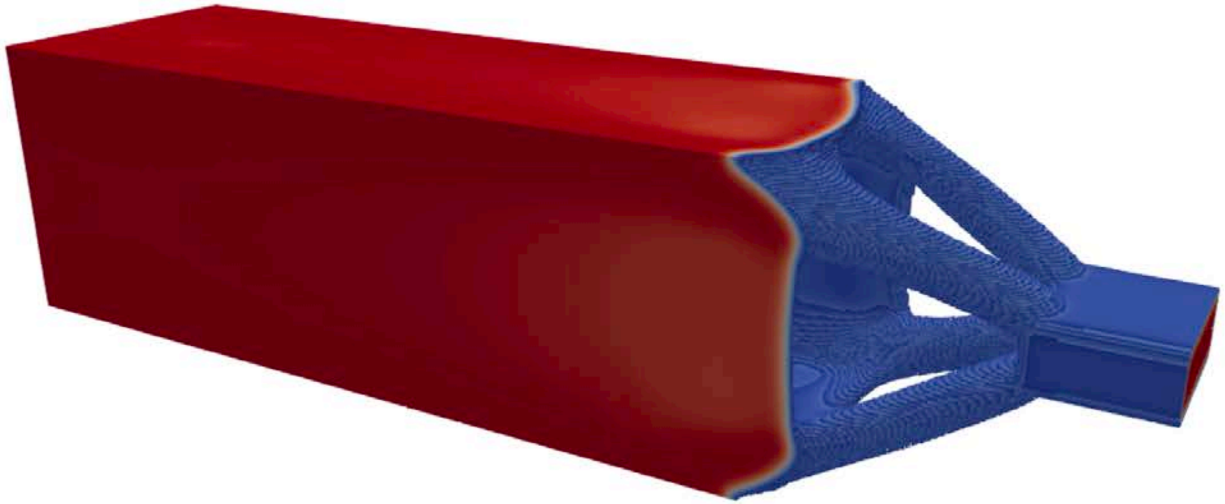
Topology optimization, second derivatives and OpenMDAO

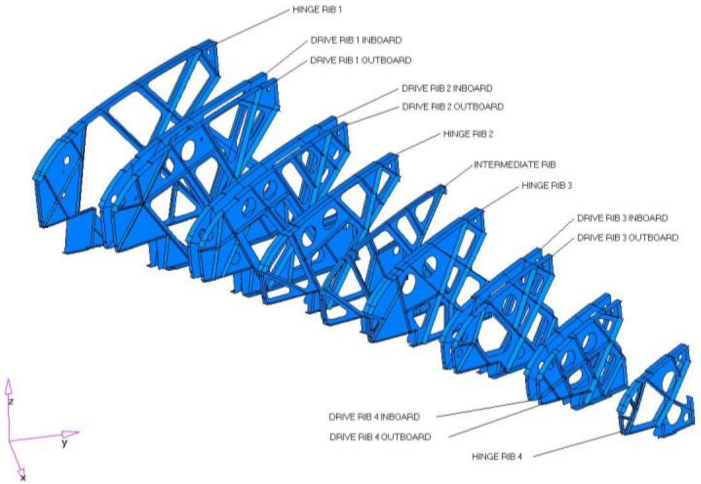


2022 OpenMDAO workshop

Graeme J. Kennedy

Georgia Institute of Technology

Topology optimization applications



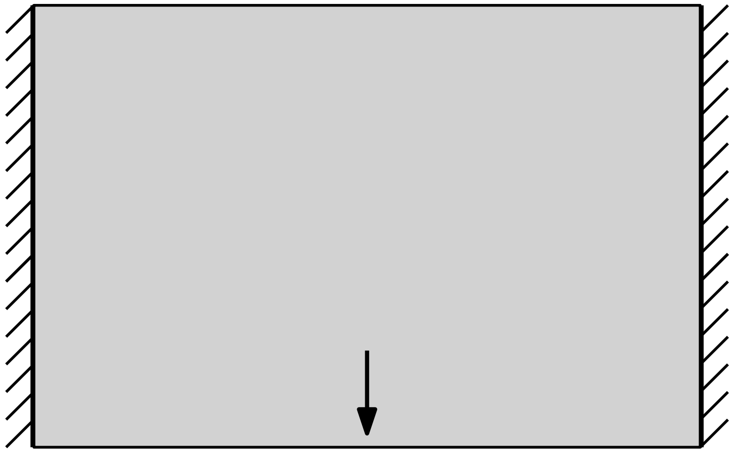
AIRBUS: 13 A380 leading edge ribs
Credit: AIRBUS

Prototype “A” slab, 80% mass reduction
Credit: Andrei Jipa et al.

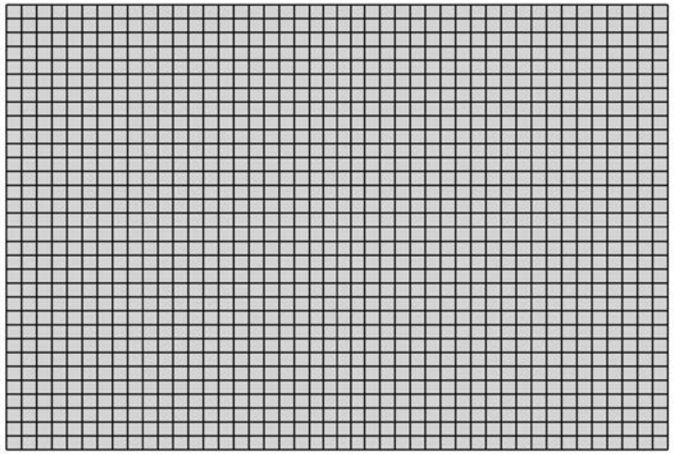
Topologically optimized chassis
Credit: SIEMENS

Topology optimization

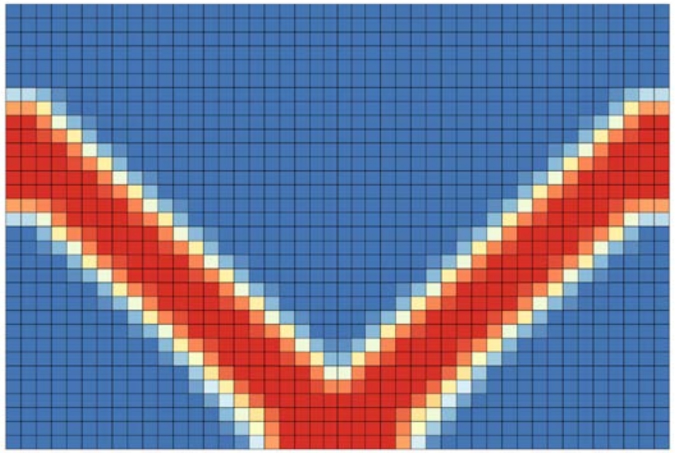
- Optimized structural design with few geometric constraints



Problem Definition

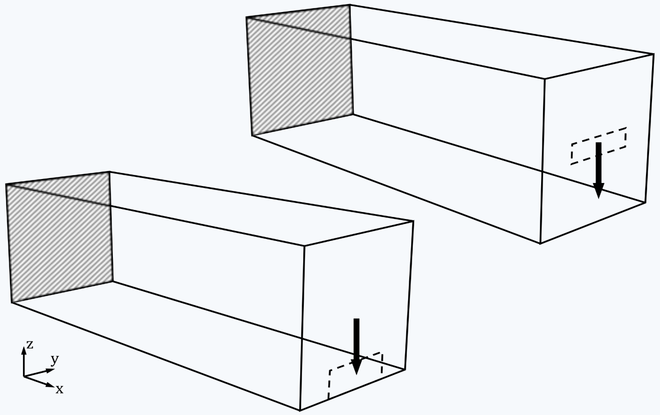


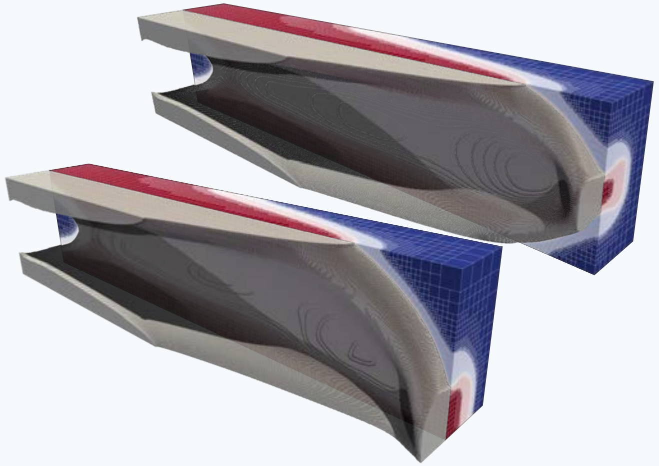
Discretize



Optimize

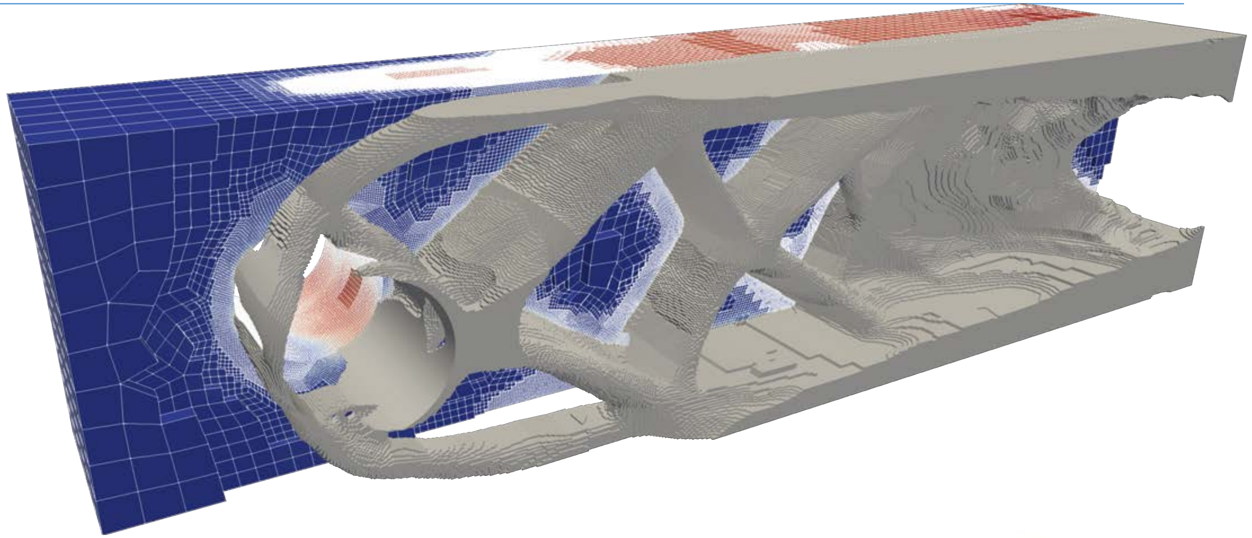
$$
\begin{aligned}
\min_{\mathbf{x}} \quad & c(\mathbf{x}) = \mathbf{f}^T \mathbf{u} \\
\text{such that} \quad & \mathbf{x} \in (0,1]^n \\
& \mathbf{m}^T \mathbf{x} \leq m_0 \\
\text{governed by} \quad & \mathbf{K}(\boldsymbol{\rho})\mathbf{u} = \mathbf{f} \\
& \boldsymbol{\rho} = \mathbf{F}\mathbf{x}
\end{aligned}
$$

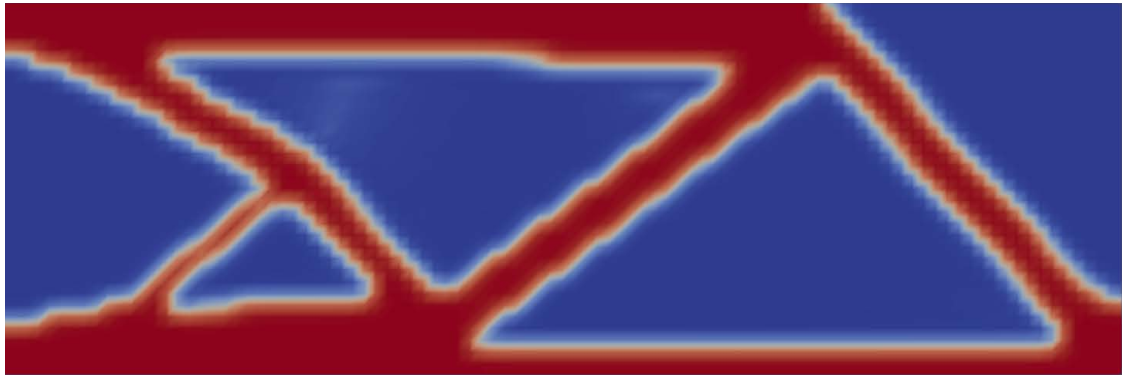


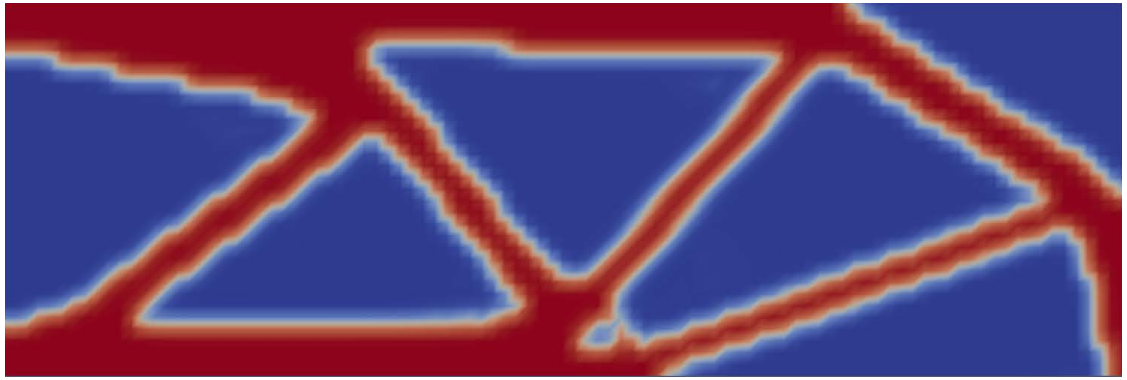


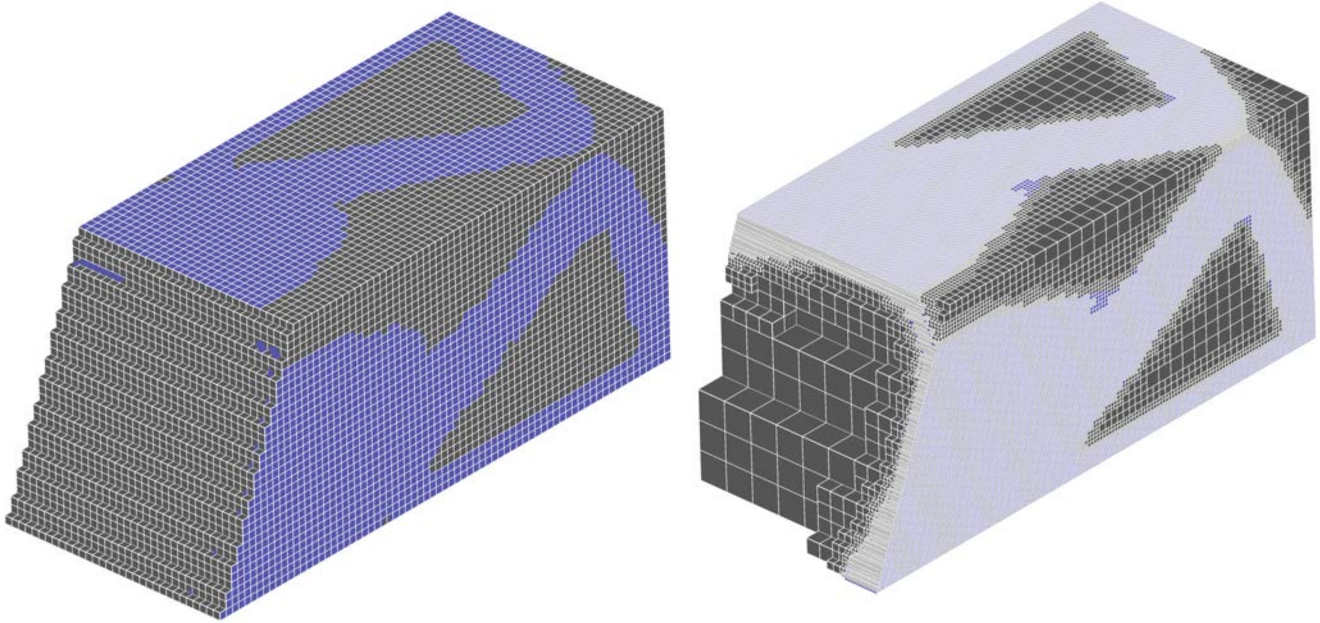
What we're trying to solve next

- Improve optimization algorithms
- Include more nonlinear physics
- Solve multiphysics problems
- Coupling with other disciplines





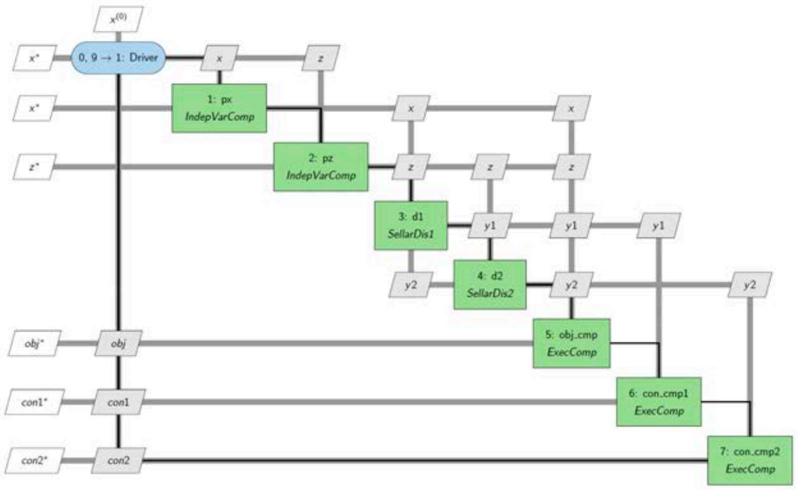


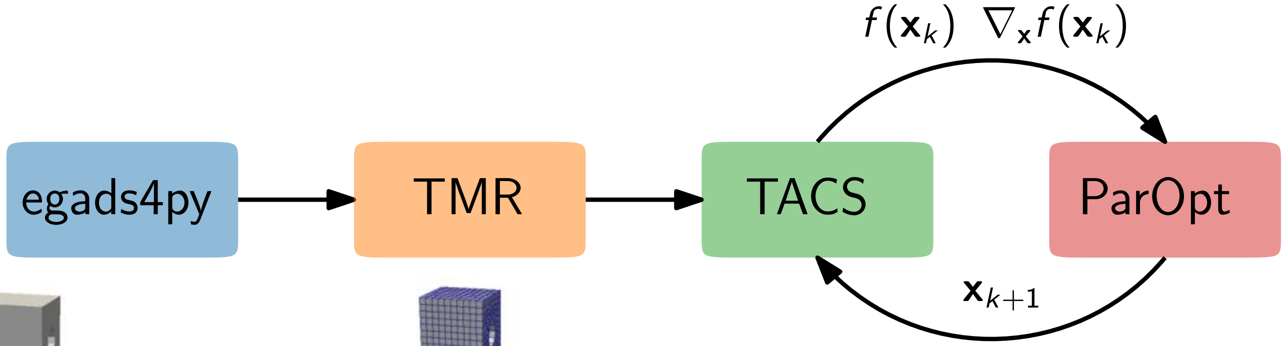


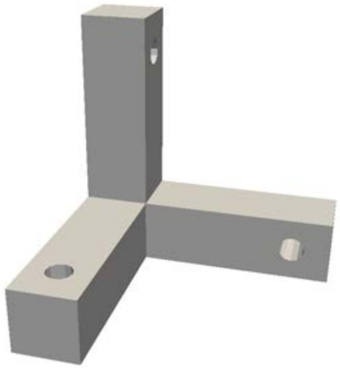
Where does OpenMDAO come in?

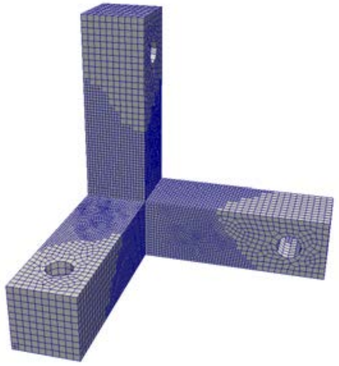
Where we're using it now:

- Structural optimization with objectives and constraints from system performance
- Integration with mphys







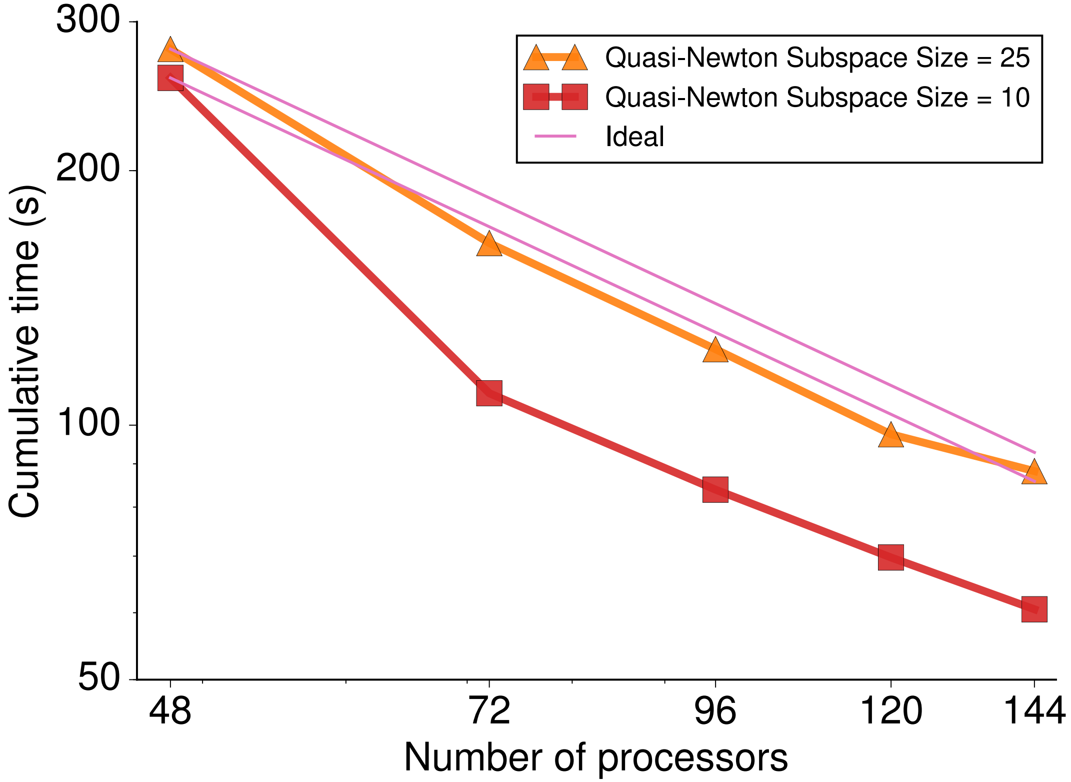


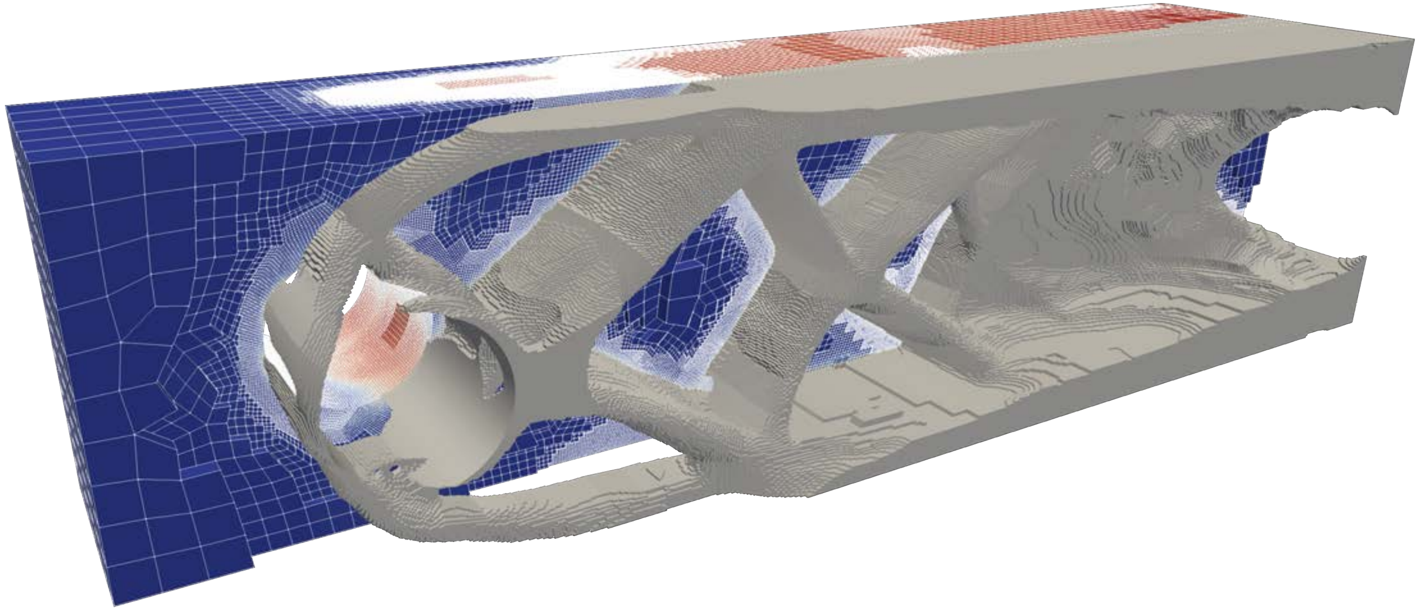
Where we're going to use it:

- Improve modularity of our own codes that are coupled together
- Integrate with other disciplines
- Include derivatives/adjoint-compatibility for all coupling

ParOpt: Driver and in pyOptSparse

https://github.com/smdogroup/paropt





Using exact Hessian-vector products

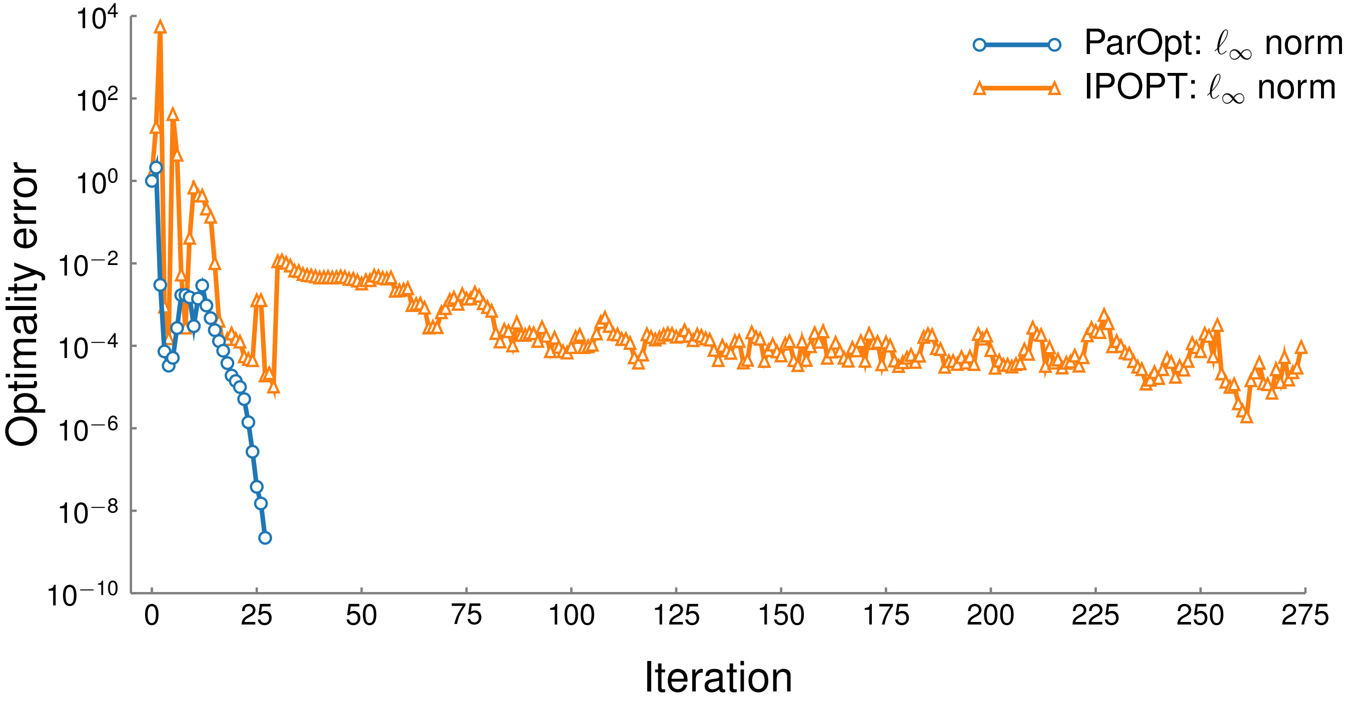
- Hessian-vector products can speed up solution
- Can be used as a globalization strategy

Second-order adjoint

$$\mathbf{K}\psi = \frac{\partial \mathbf{K}\mathbf{u}}{\partial \mathbf{x}}\mathbf{p}_x$$

$$\mathbf{H}\mathbf{p}_x = 2\psi^T \frac{\partial \mathbf{K}\mathbf{u}}{\partial \mathbf{x}}$$

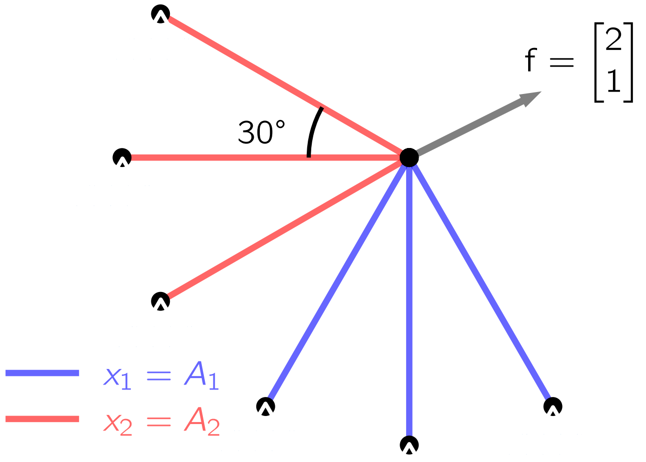
Hessian-vector product



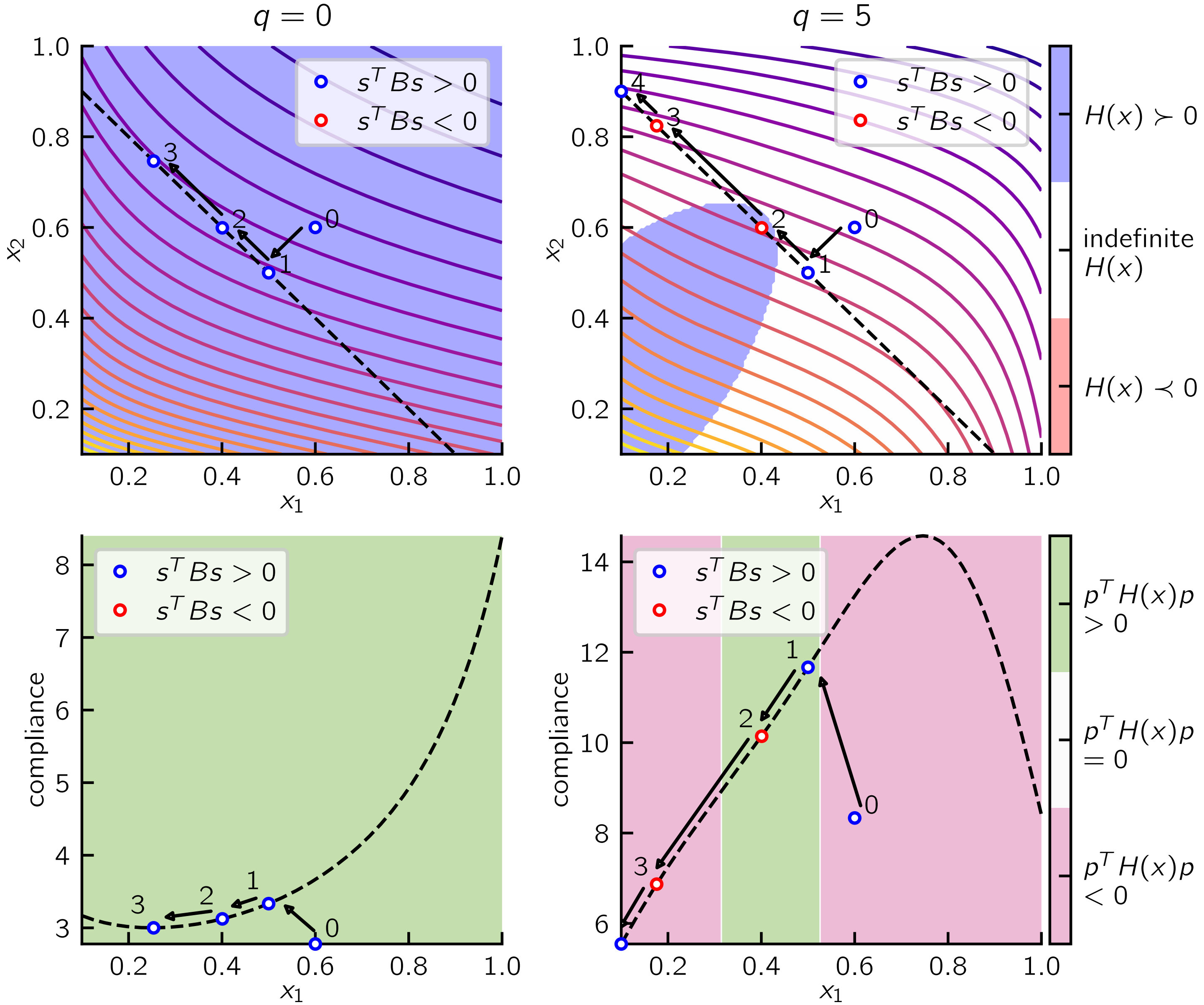
Curvature condition failures for compliance optimization

Compliance minimization

$$\begin{aligned} \min_{x_1, x_2} \quad & c(x_1, x_2) = \mathbf{f}^T \mathbf{u} \\ \text{such that} \quad & x_1, x_2 \in [0.1, 1] \\ & x_1 + x_2 \leq 1 \\ \text{governed by} \quad & \mathbf{K}(x_1, x_2)\mathbf{u} = \mathbf{f} \end{aligned}$$



Stolpe-Svanberg 6-bar truss system



Compliance contours, definiteness contours and constraint subspace

Approximate only the positive part of the Hessian

Negative part

$$\frac{\partial^2 f(\mathbf{x})}{\partial \mathbf{x}^2} = \mathbf{P} - \mathbf{N}$$

Positive part

Causes curvature condition failures

Maximize stiffness and optimize for frequency

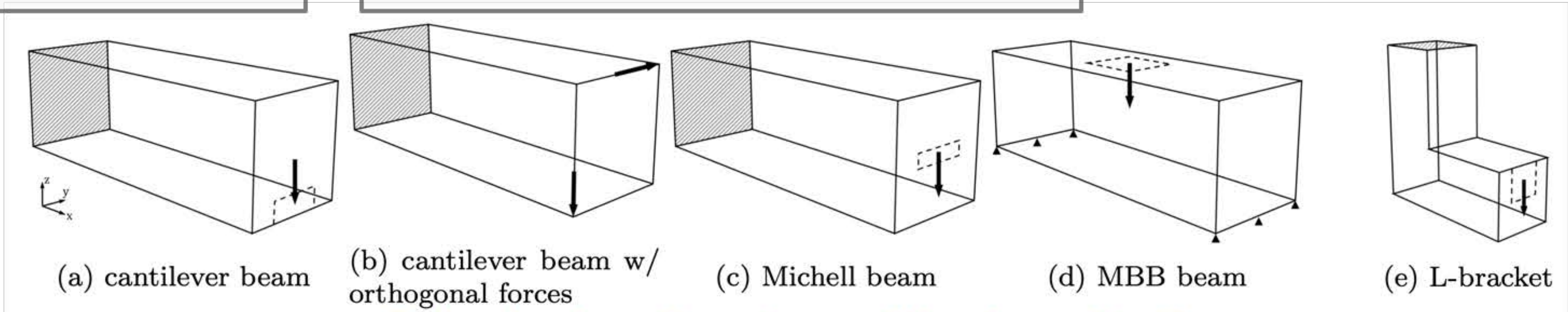
Test problem 1: compliance minimization under a linear constraint

$$
\begin{aligned}
\min_{\mathbf{x}} \quad & c(\mathbf{x}) = \mathbf{f}^T\mathbf{u} \\
\text{such that} \quad & \mathbf{x} \in (0,1]^n \\
& \mathbf{m}^T\mathbf{x} \le m_0 \\
\text{governed by} \quad & \mathbf{K}(\boldsymbol{\rho})\mathbf{u} = \mathbf{f} \\
& \boldsymbol{\rho} = \mathbf{F}\mathbf{x}
\end{aligned}
$$

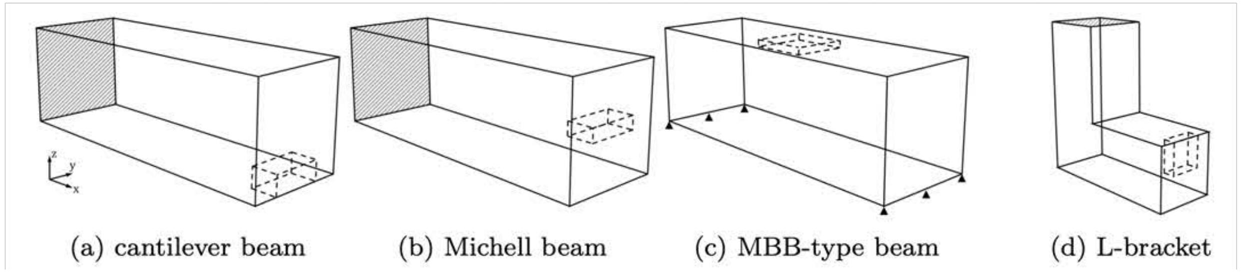
Test problem 2: mass minimization under natural frequency constraint

$$
\begin{aligned}
\min_{\mathbf{x}} \quad & \mathbf{m}^T\mathbf{x} \\
\text{such that} \quad & \mathbf{x} \in (0,1]^n \\
& g(\mathbf{x}; p) \ge 0 \\
\text{governed by} \quad & \mathbf{A}\boldsymbol{\Phi} = \boldsymbol{\Phi}\boldsymbol{\Lambda} \\
& \boldsymbol{\Phi}^T\boldsymbol{\Phi} = \mathbf{I} \\
& \boldsymbol{\rho} = \mathbf{F}\mathbf{x}
\end{aligned}
$$

Optimizer	Method
SNOPT	SQP active-set line search method
IPOPT	Interior point method
ParOpt	SQP trust region method
ParOpt w/ correction	SQP trust region with quasi-Newton correction
MMA	Method of moving asymptotes

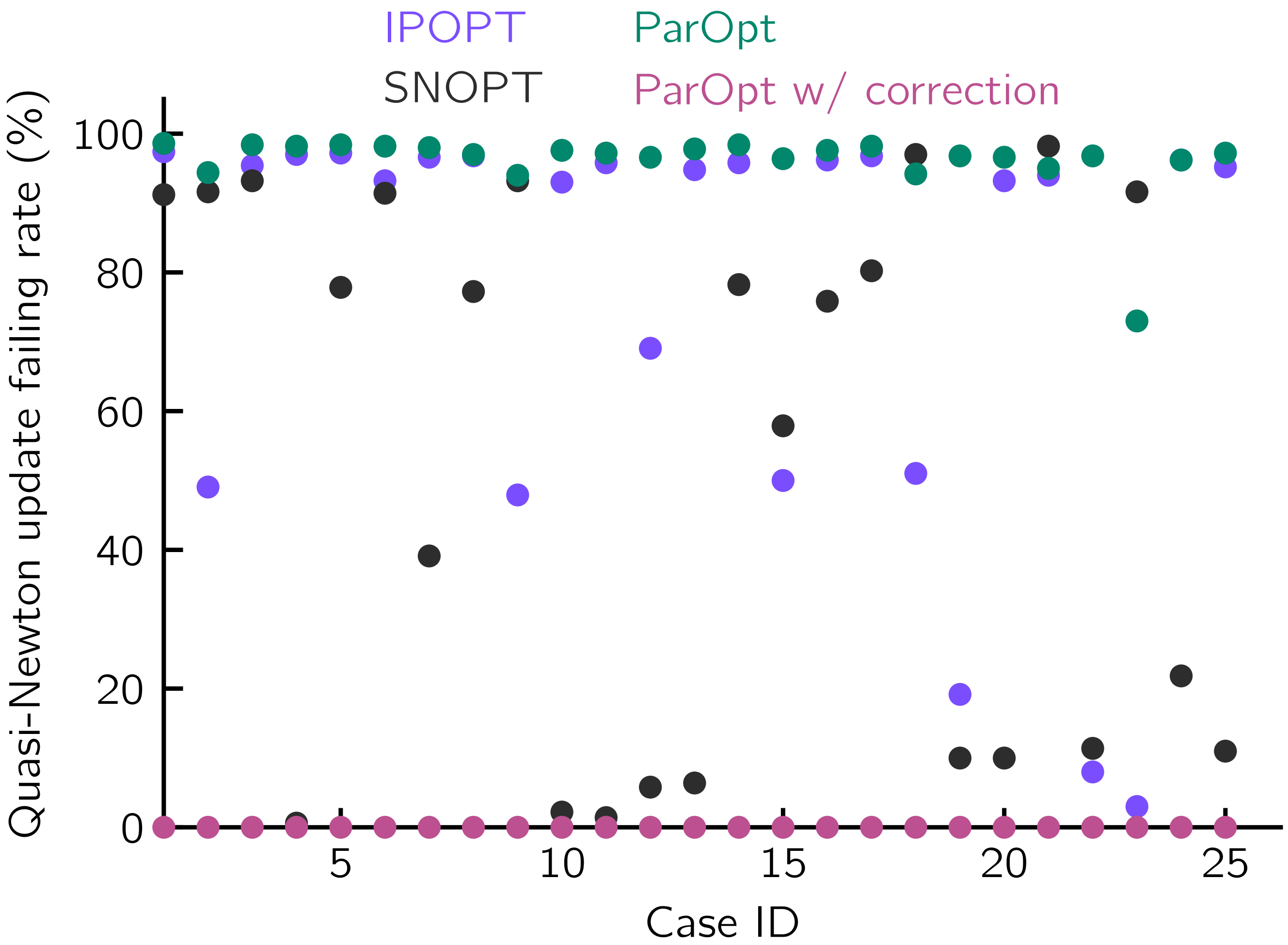


Design domains and boundary conditions for problem 1

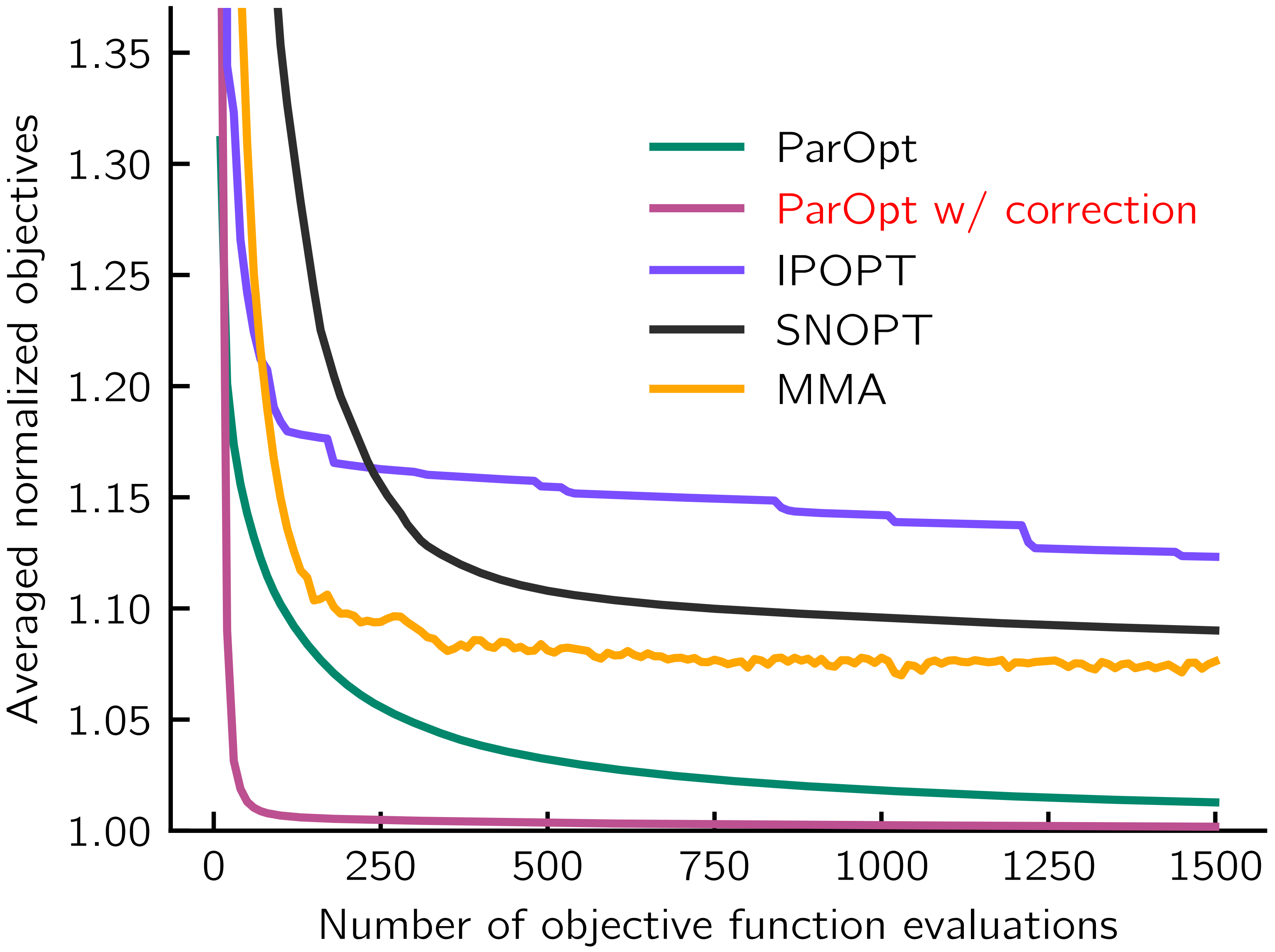


Curvature condition failures

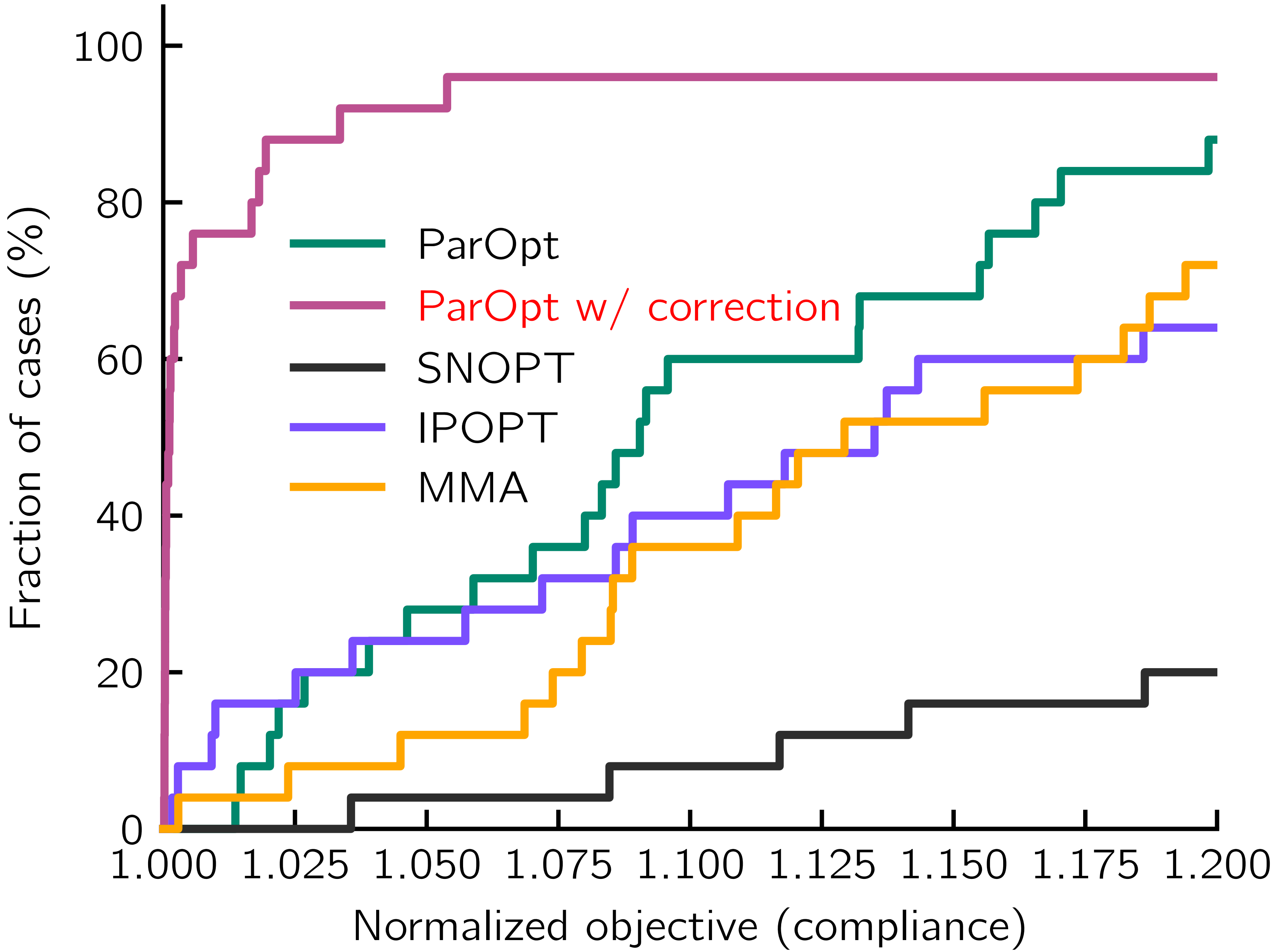
- Curvature condition fails on average 50% – 90% of the time
- Very few failures with correction



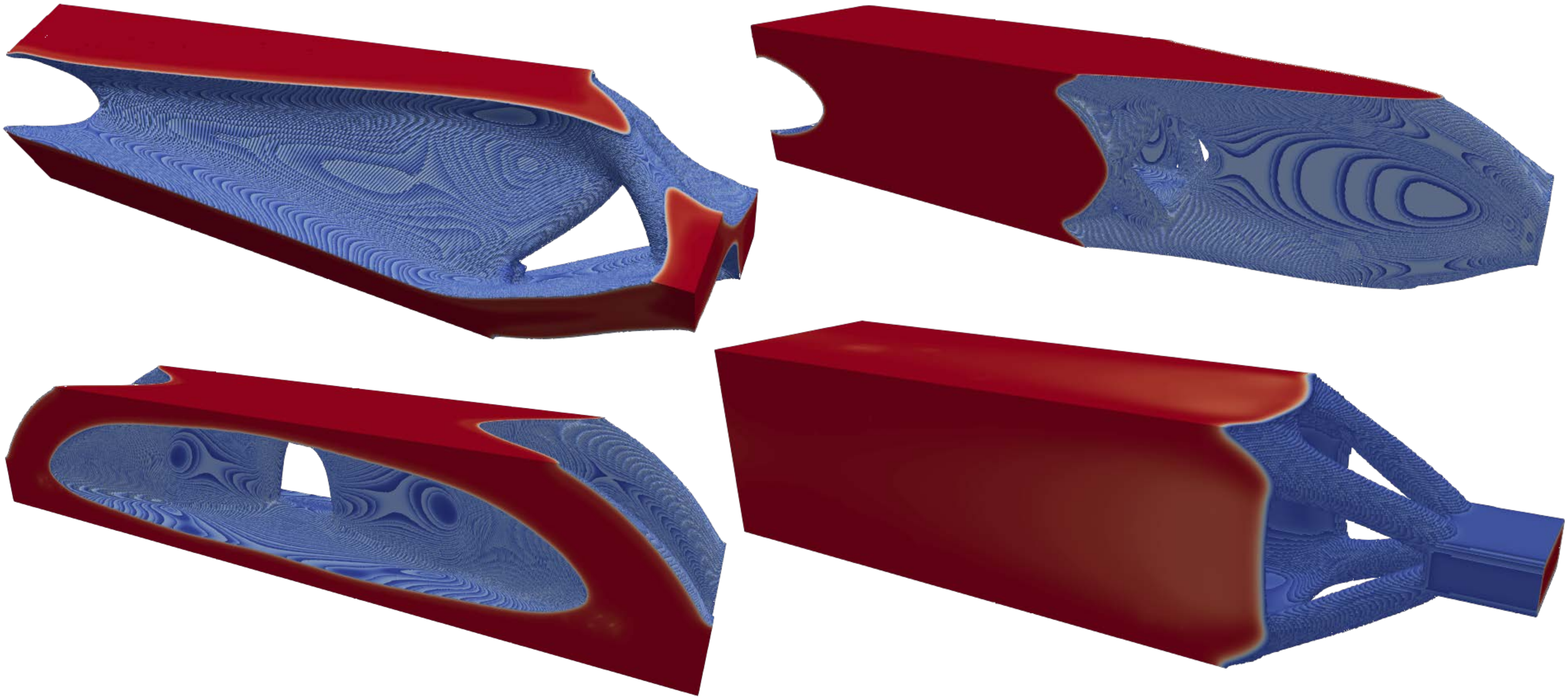
Correction performs better across **150** problems



Performance profile after 100 function evaluations

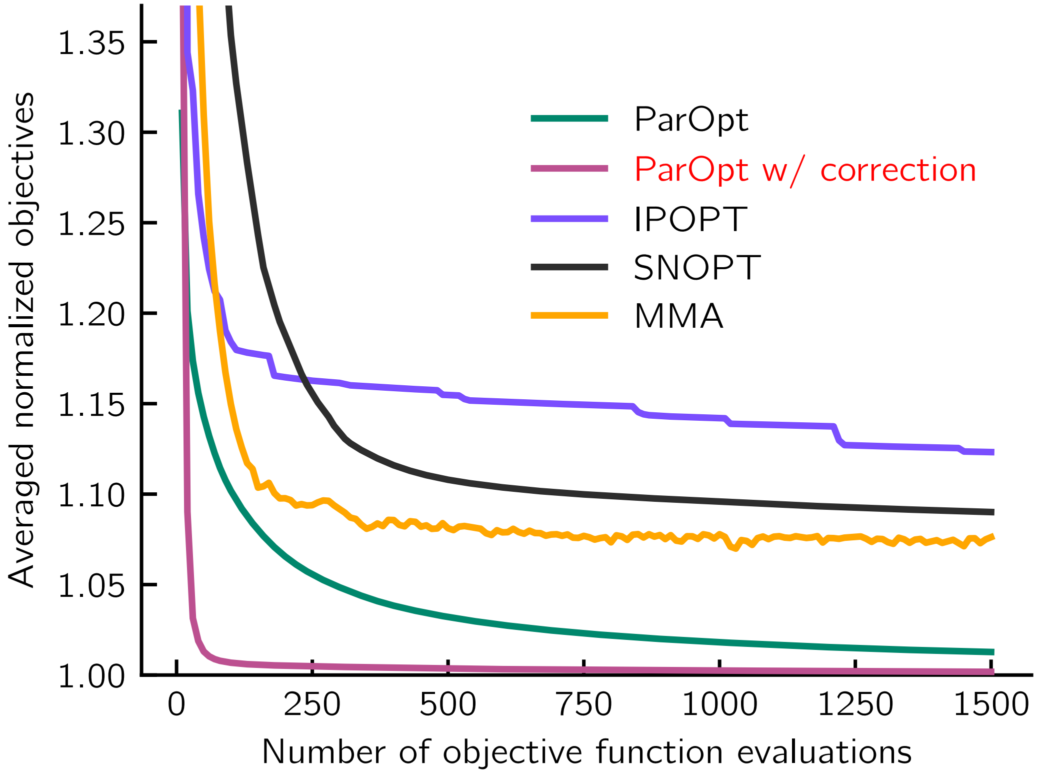


Large-scale results: 90+ million dof



Second derivative conclusions

- First-order derivatives need to be accurate
- Second-order derivatives generally do not – positive curvature is more important
  - We make our "Hessian approximation" worse and the optimizer converges faster



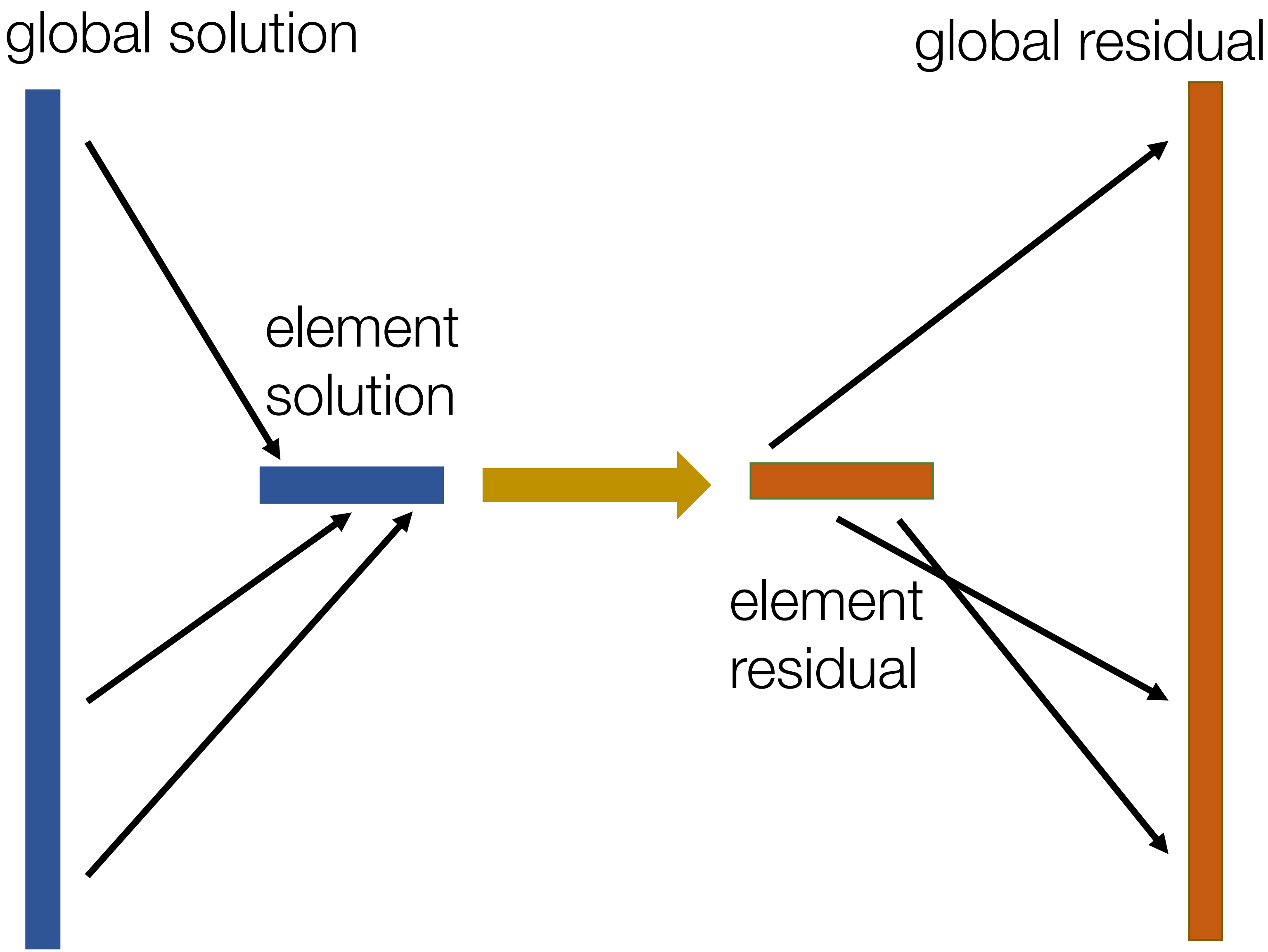
TACS and pthreads: A cautionary tale

- Around 2011 I added pthreads to TACS
- This was actually a lot of fun to do, but tedious

```
if (thread_info->getNumThreads() > 1) {
  // Set the number of completed elements to zero
  numCompletedElements = 0;
  tacsPInfo->assembler = this;

  // Create the joinable attribute
  pthread_attr_t attr;
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

  for (int k = 0; k < thread_info->getNumThreads(); k++) {
    pthread_create(&threads[k], &attr, TACSAssembler::assembleRes_thread,
                   (void *)tacsPInfo);
  }

  // Join all the threads
  for (int k = 0; k < thread_info->getNumThreads(); k++) {
    pthread_join(threads[k], NULL);
  }

  // Destroy the attribute
  pthread_attr_destroy(&attr);
} else {
```

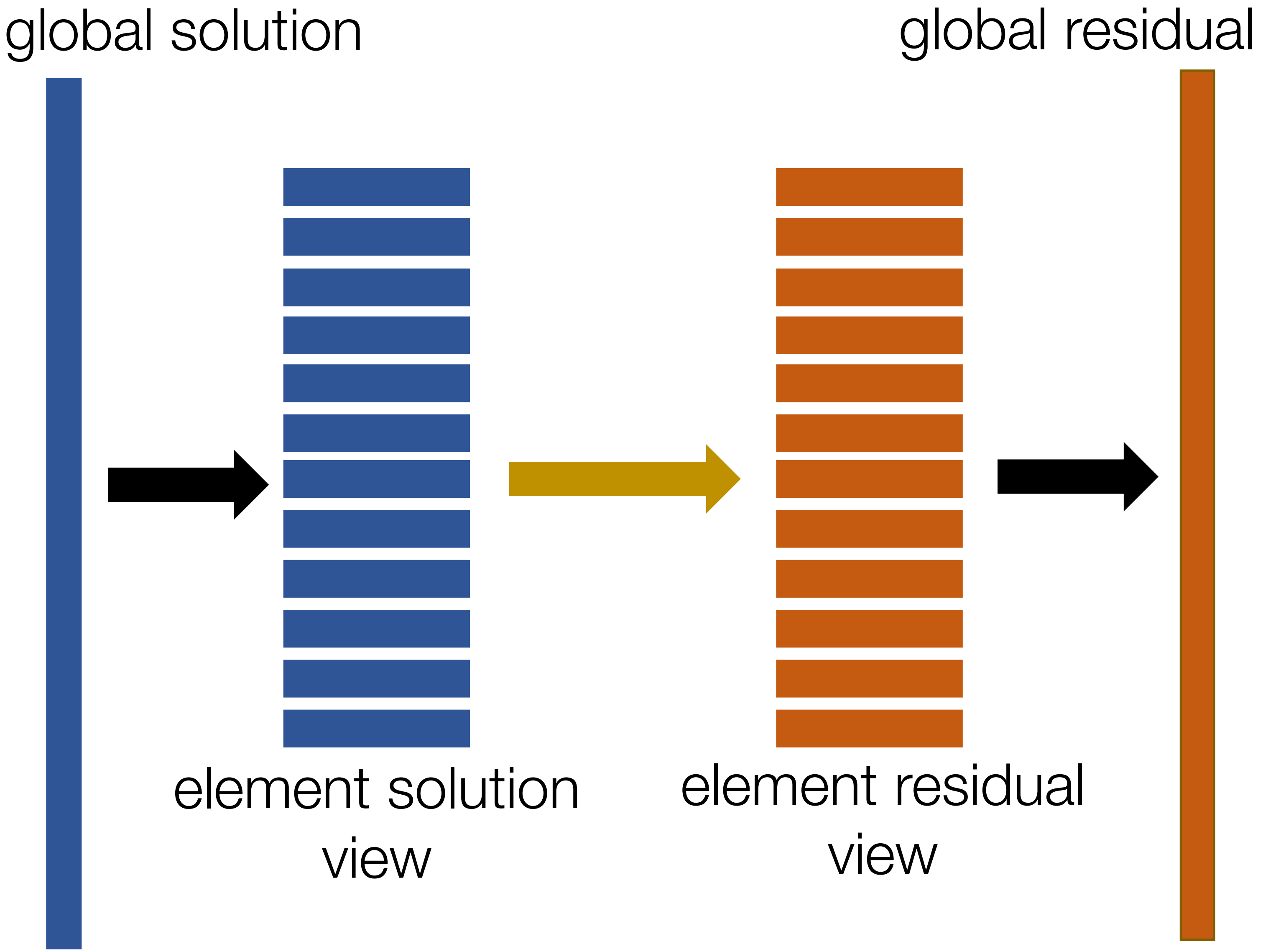
- This was before c++11 so functors/lambdas weren't widely available yet
- Shared memory – all threads work on the same memory
- Lots of unnecessary control over the thread behavior
- Not portable code

Vector access and memory layout



- Contribution from a single element residual
- Read/write to random locations within the solution and residual vectors
- When you parallelize vectors you implement some buffering magic so that non-local components can be accessed
  - For instance Petsc vectors

Better parallelism, more memory



- From the element perspective, the view of the vector has changed
- Fewer cache misses since the variables are stored in the correct view
- This is a generalization of the vector magic that Petsc implements

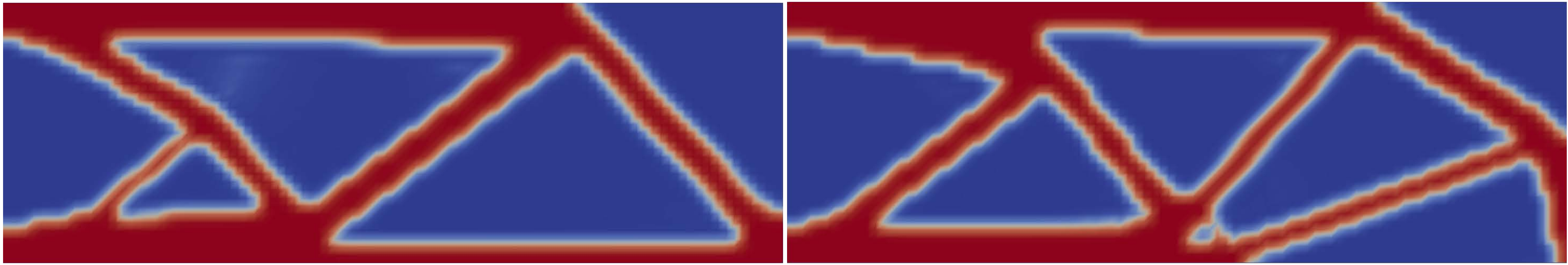
Two abstractions and programming efficiency

- Abstraction 1: *Vector views and access*
  - I want to express the finite-element equations in a generic way without worrying about how memory is accessed
- Abstraction 2: *Execution pattern*
  - I don't want to deal with pthreads
  - Implementation should express an algorithm, not a specific implementation
- Programming efficiency: *Automatic differentiation for everything*
  - I never want to compute a derivative again
  - But I don't want to give up performance

- *We're developing A2D (Almost Automatic Differentiation) to achieve these goals*

A2D: Almost Automatic Differentiation

- Straightforward to implement new tightly coupled multiphysics analysis
- Derivatives computed using automatic differentiation
    - We need first and second derivatives
- Target different HPC architectures
    - We use Kokkos to abstract the vectors and execution space
- Path towards integration with TACS



Why second derivatives?

- Total potential energy:

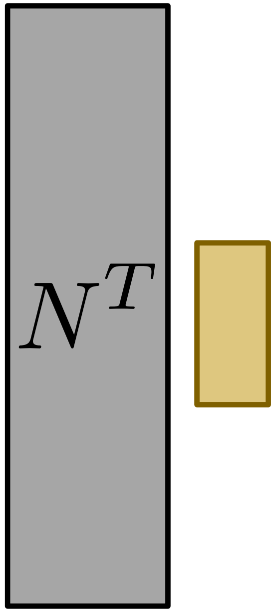
$$\Phi = \sum_i w_i \Phi_i(\nabla u)$$

Computed from the element solution

$$\nabla u = N u_e$$

- Residual is the derivative of energy:

$$R = \sum_i w_i N^T \left[\frac{\partial \Phi_i}{\partial \nabla u} \right]^T =$$



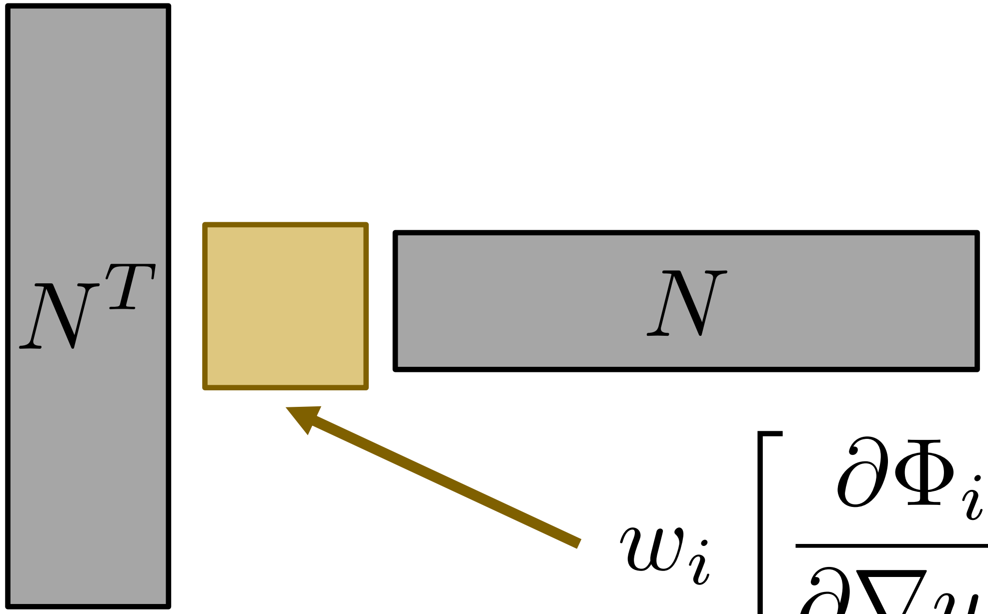
$$w_i \left[\frac{\partial \Phi_i}{\partial \nabla u} \right]^T$$

AD applied here

Why second derivatives?

- The Jacobian is the second derivative of energy:

$$J = \sum_i w_i N^T \left[\frac{\partial^2 \Phi_i}{\partial \nabla u^2} \right]^T N =$$



$$w_i \left[\frac{\partial \Phi_i}{\partial \nabla u^2} \right]^T$$

AD applied here

- Adjoint terms are Hessian-vector products

$$\psi^T \frac{\partial R}{\partial x} = \sum_i w_i \psi_i^T \left[\frac{\partial^2 \Phi_i}{\partial \nabla u \partial x} \right]^T$$

How the second derivatives are computed

- Original code

$$x_i \to y_j \to f(y(x))$$

- Reverse mode AD

$$\bar{y}_j = \frac{\partial f}{\partial y_j} \Rightarrow \bar{x}_i = \bar{y}_j \frac{\partial y_j}{\partial x_i}$$

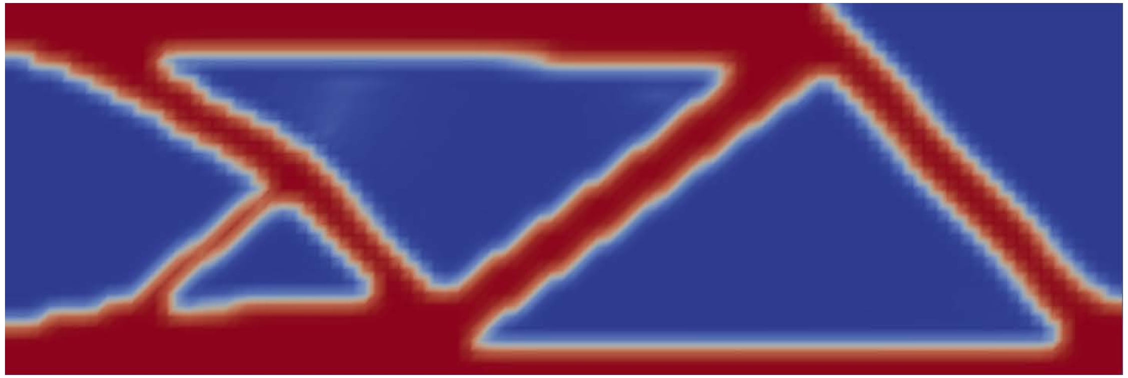
- Forward and reverse mode for Hessian

$$\hat{x}_i = \hat{y}_k \frac{\partial y_k}{\partial x_i} + \bar{y}_k \frac{\partial^2 y_k}{\partial x_i \partial x_j} \dot{x}_j = \frac{\partial^2 f}{\partial x_i \partial x_j} \dot{x}_j$$

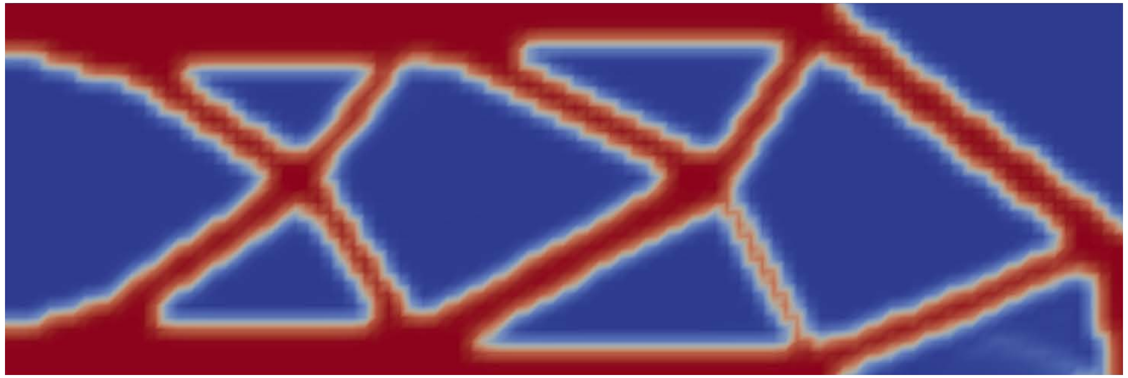
```
// Express energy using Uxi
auto mult = A2D::MatMatMult(Uxi, Jinv0, Ux);
auto strain = A2D::MatGreenStrain(Ux, E);
auto energy = A2D::SymmIsotropicEnergy(mu, lambda, E, output);

// Reverse sweep
energy.reverse();
strain.reverse();
mult.reverse();

// Forward and reverse sweep
mult.hforward();
strain.hforward();
energy.hreverse();
strain.hreverse();
mult.hreverse();  // Jacobian is available
```

Initial optimization demonstration with A2D

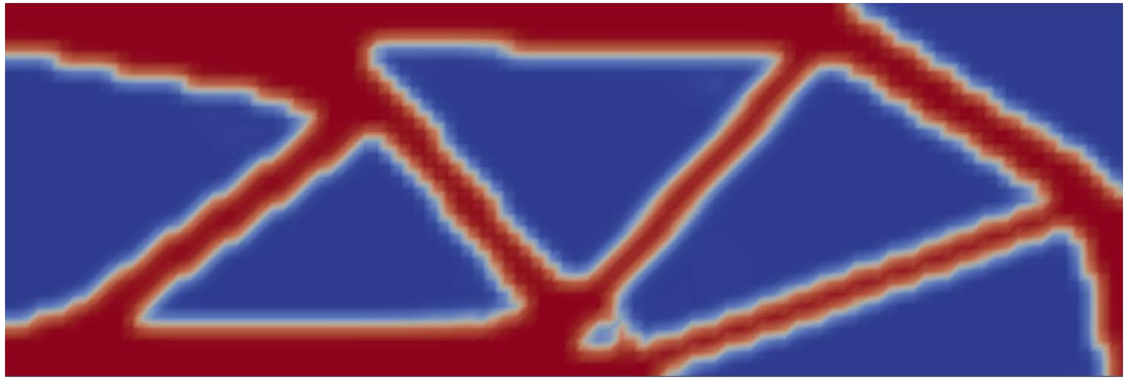
- Compliance minimization with geometrically nonlinear analysis
- Optimized design changes with load magnitude



$F = 10N$

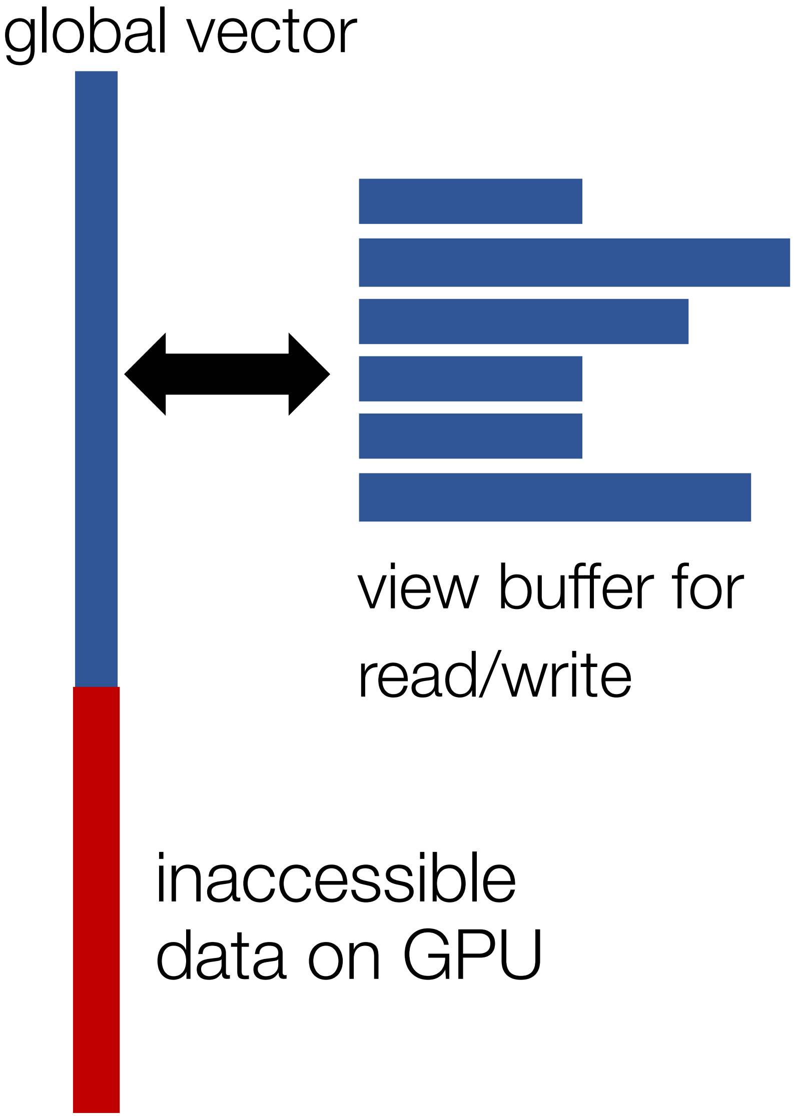


$F = 50N$



$F = 100N$

A path to BYOV in OpenMDAO/Mphys?



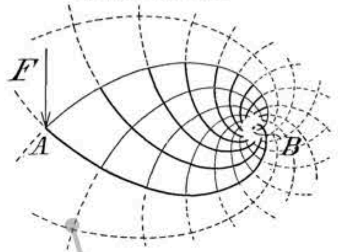
- Current approach to vector views provides component-wise slices of the residual/solution/design
- Problem: Not all data will be on the CPU or should be copied from component
- Vector class encapsulates two behaviors
  - Global operations – uses inaccessible data implicitly
    - norm, dot-product, axpy
  - Component-wise access and manipulation – explicit access only to buffered data
    - __setitem__, __getitem__
- Provide component-wise vector through views of subset of data
- Less capability for automatic scaling/unit conversions on inaccessible data

Conclusions

- Second derivatives can improve computational efficiency
- Automatic differentiation can be used for multiphysics applications
- Something like BYOV needed for integration of OpenMDAO with GPU/HPC computing

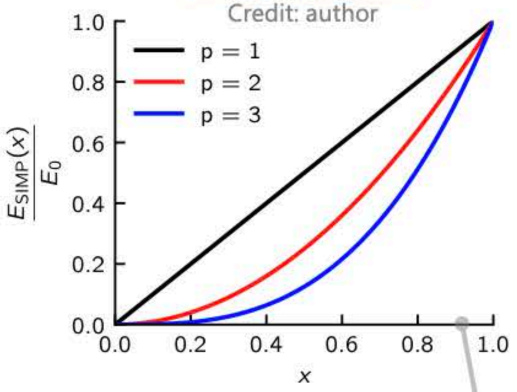
History of topology optimization

optimal frame for singular supported beam
Credit: Michell

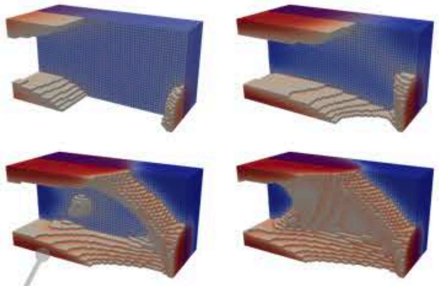


Analytical optimal truss layout
(A. Michell, 1904)

SIMP penalization
Credit: author

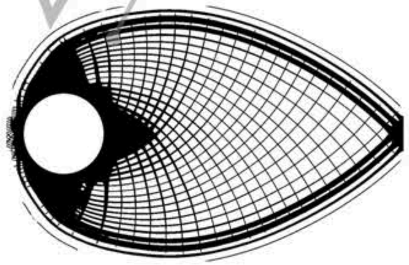


Evolution of density solutions
Credit: author



Density method (SIMP)
(M. Bendsøe, 1989, Rozvany and Zhou, 1991)

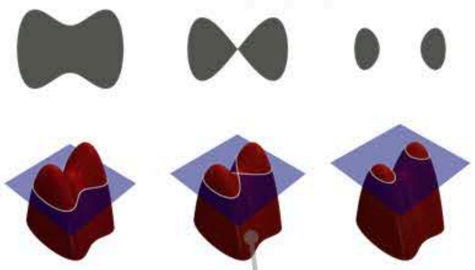
Topology Optimization w/ homogenization method
(M. Bendsøe, 1988)



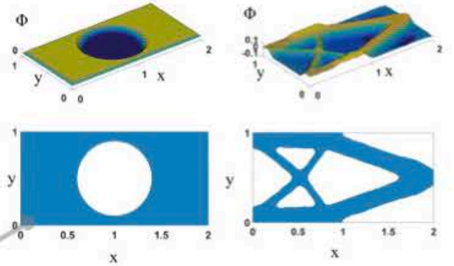
Optimal Michell-type w/ homogenization and projection
Credit: Groen et al.

Topological derivatives
(Eschenauer et al., 1994)

Level set function and isolines
Credit: Nicoguaro



Level set cantilever solution
Credit: Jiang et al.



Level-set method
(Oscher and Sethian, 1998, etc.)

Others (phase field approach, evolutionary approach, discrete approach, etc.)

year