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

\[
\begin{align*}
\min_x & \quad c(x) = f^T u \\
\text{such that} & \quad x \in (0, 1]^n \\
& \quad m^T x \leq m_0 \\
\text{governed by} & \quad K(\rho)u = f \\
& \quad \rho = Fx
\end{align*}
\]
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

\[ K\psi = \frac{\partial Ku}{\partial x} p_x \]

Hessian-vector product

\[ Hp_x = 2\psi^T \frac{\partial Ku}{\partial x} \]
Curvature condition failures for compliance optimization

Compliance minimization

\[
\min_{x_1, x_2} c(x_1, x_2) = f^T u
\]

such that

\[
x_1, x_2 \in [0, 1]
\]

\[
x_1 + x_2 \leq 1
\]

governed by

\[
K(x_1, x_2)u = f
\]

Stolpe-Svanberg 6-bar truss system
Approximate only the positive part of the Hessian

\[ \frac{\partial^2 f(x)}{\partial x^2} = \text{P} - \text{N} \]

Causes curvature condition failures
Maximize stiffness and optimize for frequency

**Test problem 1: compliance minimization under a linear constraint**

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

**Test problem 2: mass minimization under natural frequency constraint**

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

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNOPT</td>
<td>SQP active-set line search method</td>
</tr>
<tr>
<td>IPOPT</td>
<td>Interior point method</td>
</tr>
<tr>
<td>ParOpt</td>
<td>SQP trust region method</td>
</tr>
<tr>
<td>ParOpt w/</td>
<td>SQP trust region with quasi-Newton correction</td>
</tr>
<tr>
<td>correction</td>
<td></td>
</tr>
<tr>
<td>MMA</td>
<td>Method of moving asymptotes</td>
</tr>
</tbody>
</table>

**Design domains and boundary conditions for problem 1**

(a) cantilever beam  (b) cantilever beam w/ orthogonal forces  (c) Michell beam  (d) MBB beam  (e) L-bracket
Curvature condition failures

- Curvature condition fails on average 50% – 90% of the time
- Very few failures with correction
Correction performs better across 150 problems

![Graph showing averaged normalized objectives for different optimization algorithms: ParOpt, ParOpt w/ correction, IPOPT, SNOPT, MMA. The x-axis represents the number of objective function evaluations, ranging from 0 to 1500, and the y-axis represents the averaged normalized objectives, ranging from 1.00 to 1.35. The graph shows that ParOpt w/ correction performs the best, followed by ParOpt, IPOPT, SNOPT, and MMA.]
Performance profile after 100 function evaluations

Fraction of cases (%)

Normalized objective (compliance)

ParOpt
ParOpt w/ correction
SNOPT
IPOPT
MMA
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

```c
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:
  \[ \Phi = \sum_i w_i \Phi_i (\nabla u) \]

- Residual is the derivative of energy:
  \[ \nabla u = Nu_e \]
  \[ R = \sum_i w_i N^T \left[ \frac{\partial \Phi_i}{\partial \nabla u} \right]^T = N^T \]

Computed from the element solution

AD applied here
Why second derivatives?

- The Jacobian is the second derivative of energy:

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

\[ N = N^T \]

- 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

\[ x_i \rightarrow y_j \rightarrow f(y(x)) \]

• Reverse mode AD

\[
\bar{y}_j = \frac{\partial f}{\partial y_j} \quad \Rightarrow \quad \bar{x}_i = \bar{y}_j \frac{\partial y_j}{\partial x_i}
\]

• 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
multi.hforward();
strain.hforward();
energy.hreverse();
strain.hreverse();
multi.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

- **Analytical optimal truss layout** (A. Michell, 1904)
- **Optimal frame for singular supported beam**
  Credit: Michell
- **Optimal Michell-type w/ homogenization and projection**
  Credit: Groen et al.
- **Topology Optimization w/ homogenization method**
  (M. Bendsøe, 1988)
- **Density method (SIMP)**
  (M. Bendsøe, 1989, Rozvany and Zhou, 1991)
- **Topological derivatives**
  (Eschenauer et al., 1994)
- **Level-set method**
  (Osher and Sethian, 1998, etc.)
- **Level set cantilever solution**
  Credit: Jiang et al.
- **Level set function and isolines**
  Credit: Nicoquaro
- **Evolution of density solutions**
  Credit: author
- **SIMP penalization**
  Credit: author

Year