Analysis of the Adjoint Euler Equations as used for Gradient-based Aerodynamic Shape Optimization

Final Presentation

Dylan Jude
Graduate Research Assistant

University of Maryland
AMSC 663/664

May 4, 2017

Abstract

- Adjoint methods are often used in gradient-based optimization because they allow for a significant reduction of computational cost for problems with many design variables.
- The project focuses on the use of adjoint methods for two-dimensional airfoil shape optimization using Computational Fluid Dynamics to solve the steady Euler equations.

Design Problem

We can change the shape of an airfoil by prescribing sinusoidal "bumps" along the surface of the airfoil:

Known as Hicks-Henne "bumps", both the location and the height of the bumps can be changed to alter the airfoil shape. [Hicks and Henne(1977)]

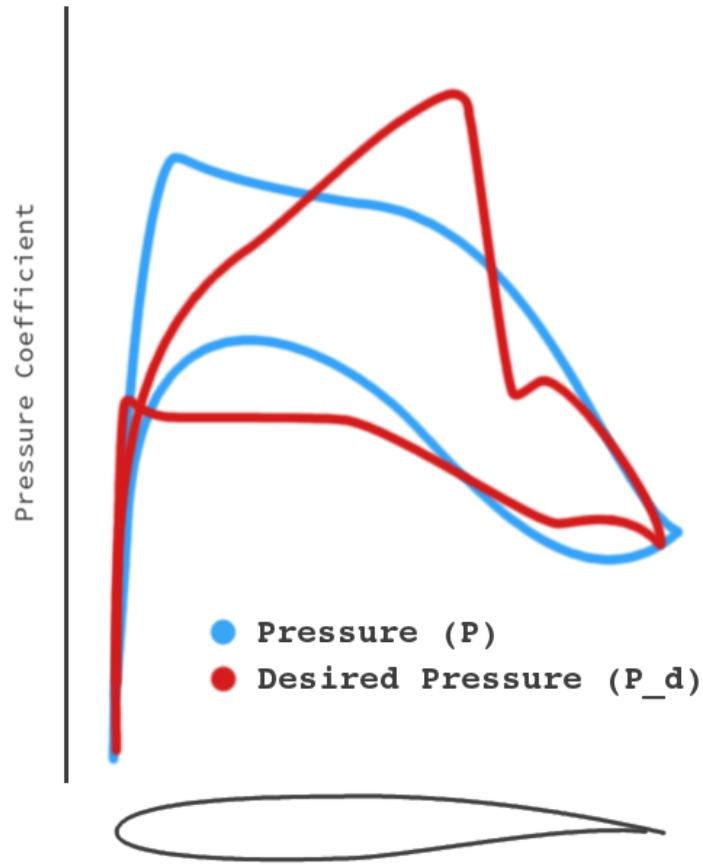
Design Problem

We would want to alter an airfoil to obtain more favorable aerodynamic properties. A simple example would be to approach a desired pressure distribution.

Mathematically, we want to minimize the cost function:

$$I_c(\alpha) = \oint_{airfoil} \frac{1}{2}(P - P_d)^2$$

where α is the set of design variables and $P = P(\alpha)$

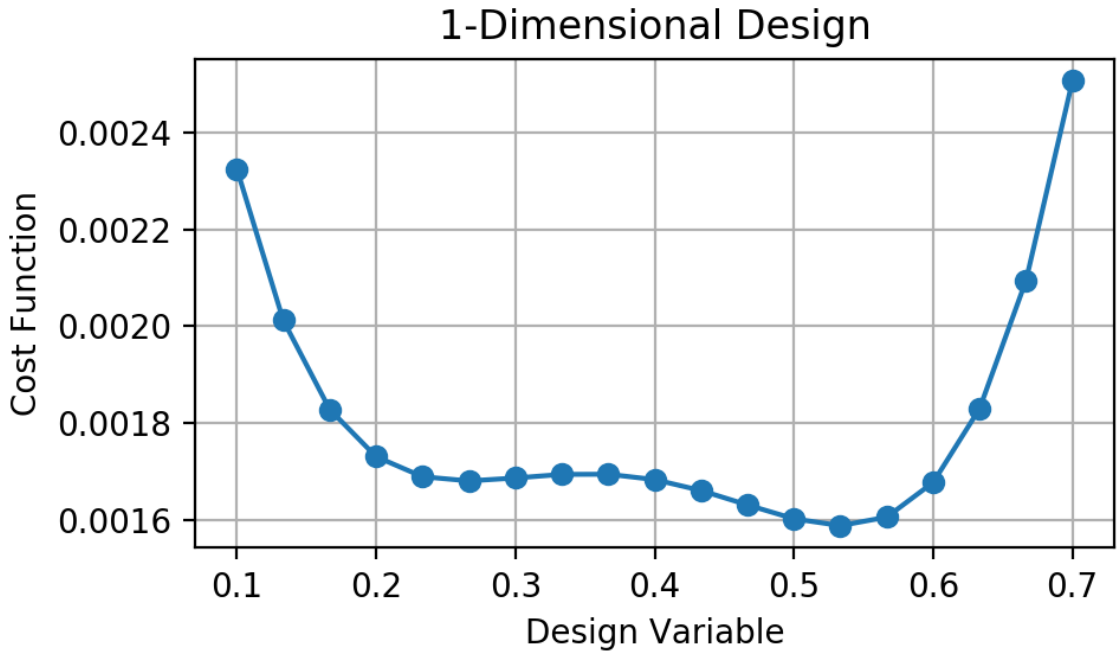


[Nadarajah and Jameson(2002)]

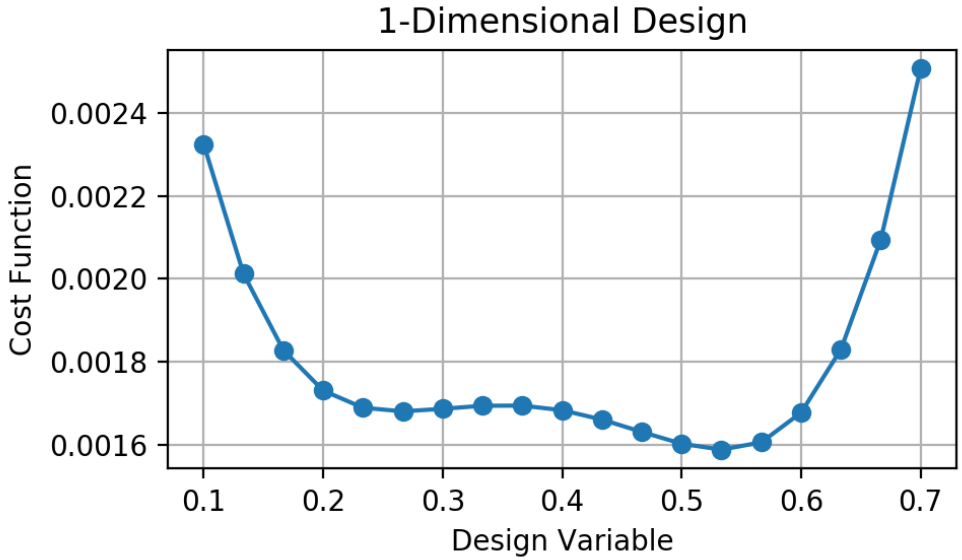
Design Problem

For a single variable representing the location of one bump on the surface of the airfoil

a parametric sweep of the cost function for a given desired pressure distribution shows



Design Problem



- The goal of airfoil optimization is to obtain the design variable (airfoil geometry) where the cost function is a minimum.
- As shown, it is possible to have local minima in the cost function. This can be a challenge for gradient-based optimization methods.
  - This will be discussed in a few slides
- Parametric sweeps of the design space (as shown above) are very expensive for multiple variables.

Computing the Cost Function

- Most design cases will have many variables.
- Computing the value of the cost function is an expensive process because it requires solving the Euler Equations (using Computational Fluid Dynamics or CFD) to find the pressure.
- Knowing the gradient of the cost function is often required for gradient-based optimization methods however finding the gradient is non-trivial and expensive.

Computing the Cost Function

We want the gradient of the cost function with respect to the design variables. Using a brute-force approach:

$$\frac{\partial I_c}{\partial \alpha_1} = \frac{I_c(\alpha_1 + \Delta\alpha_1) - I_c(\alpha_1)}{\Delta\alpha_1}$$

For 2 variables, 3 **expensive** CFD flow calculations are required to find

$$I_c(\alpha_{1,2}), \quad I_c(\alpha_1 + \Delta\alpha_1), \quad I_c(\alpha_2 + \Delta\alpha_2)$$

The adjoint method instead can find N variable sensitivities with the cost of a single CFD flow-computation and an additional adjoint.

Discrete Euler Equations

The Euler equations in coordinate directions ξ:

$$\frac{\partial q}{\partial t} + \frac{\partial f_{c,i}}{\partial \xi_i} = 0 \quad , \quad i = 1, 2 \tag{1}$$

$$q = J^{-1} \begin{bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ e \end{bmatrix}, \quad f_c = J^{-1} \begin{bmatrix} \rho V_1 \\ \rho u_1 V_1 + \xi_{1,1} p \\ \rho u_2 V_1 + \xi_{1,2} p \\ (e+p) V_1 \end{bmatrix} \tag{2}$$

$$V_i = u_1 \xi_{i,1} + u_2 \xi_{i,2} \tag{3}$$

Note: Conventional usage of ρ, u_i, e, p for density, velocity, energy, and pressure

Discrete Euler Residual

Let f denote flux in j-coordinate direction and g denote flux in k-coordinate direction.

$$\frac{\partial q}{\partial t} + \frac{\partial f}{\partial \xi_1} + \frac{\partial g}{\partial \xi_2} = 0$$

Let the Residual of the steady Euler Equation be defined as:

$$R^n = \frac{q^{n+1} - q^n}{\Delta t} = 0 \tag{4}$$

The Residual expanded in both dimensions j, k at time n is

$$R^n_{j,k} = -\left(f_{j+1/2,k} - f_{j-1/2,k}\right) - \left(g_{j,k+1/2} - g_{j,k-1/2}\right) = 0 \tag{5}$$

Approach: Adjoint Equation

For our flow solution q and airfoil geometry $X = X(\alpha_1, ..., \alpha_n)$ our cost function is

$$I_c = I_c(q, X)$$

and a perturbation of the cost function is represented as:

$$\delta I = \frac{\partial I^T}{\partial q}\delta q + \frac{\partial I^T}{\partial X}\delta X$$

A perturbation of the flow residual R is represented as:

$$\delta\left[\frac{\partial \vec{q}}{\partial t} + \frac{\partial \vec{f}_{c,i}}{\partial \xi_i}\right] = \delta R = \left[\frac{\partial R}{\partial q}\right]\delta q + \left[\frac{\partial R}{\partial X}\right]\delta X = 0$$

Using the method of Lagrange multipliers:

$$\delta I = \frac{\partial I^T}{\partial q}\delta q + \frac{\partial I^T}{\partial X}\delta X - \psi^T \left\{ \left[\frac{\partial R}{\partial q}\right] \delta q + \left[\frac{\partial R}{\partial X}\right] \delta X \right\}$$

If the adjoint equation is satisfied:

$$\left[\frac{\partial R}{\partial q}\right]^T \psi = \frac{\partial I}{\partial q} \quad \rightarrow \quad \psi^T \left[\frac{\partial R}{\partial q}\right] = \frac{\partial I^T}{\partial q}$$

then

$$\delta I = \left\{ \frac{\partial I^T}{\partial X} - \psi^T \left[\frac{\partial R}{\partial X}\right] \right\} \delta X$$

In this final equation:

$$\delta I = \left\{ \frac{\partial I^T}{\partial X} - \psi^T \left[\frac{\partial R}{\partial X} \right] \right\} \delta X$$

the cost function is independent of the flow solution. This means we can calculate all sensitivities

$$\frac{\partial I_c}{\partial \alpha_1}, \quad \frac{\partial I_c}{\partial \alpha_2}$$

from "simply" solving the adjoint equation (same cost as Euler equations)

$$\left[\frac{\partial R}{\partial q} \right]^T \psi = \frac{\partial I}{\partial q}$$

Adjoint and Euler Problem

- The 2-D problems covered in this project are relatively simple in terms of memory requirements and computational cost.
- The selected 2D grid dimensions are:
  - 181 (around airfoil) $\times$ 60 (normal to airfoil) for a total of 10860 points, 43440 degrees of freedom.
  - Euler solver with implicit routine converges 8-orders in 1000 iterations in $\sim$ 10 seconds (Intel i7 : 3.2 GHz).
- This problem size is intended to be small. More common 2D problems will have $O(1 \times 10^5)$ degrees of freedom.

Summary of Methodology

- An Euler code, mesh generator, and adjoint code have been written in C++ and wrapped in a Python framework for communication.
- The Euler code has been Auto-Differentiated using TAPENADE [Hascoët and Pascual(2004)]
- The discrete adjoint has been derived for the Euler equations.
- The discrete adjoint equations have been hand-coded and validated against auto-differentiated and brute-force gradients.
- Gradients from the adjoint solutions have been applied to multi-variable airfoil optimization.

Summary of Methodology

- An Euler code, mesh generator, and adjoint code have been written in C++ and wrapped in a Python framework for communication.
- The Euler code has been Auto-Differentiated using TAPENADE [Hascoët and Pascual(2004)]
- The discrete adjoint has been derived for the Euler equations.
- The discrete adjoint equations have been hand-coded and validated against auto-differentiated and brute-force gradients.
- Gradients from the adjoint solutions have been applied to multi-variable airfoil optimization.

Summary of Methodology

- An Euler code, mesh generator, and adjoint code have been written in C++ and wrapped in a Python framework for communication.
- The Euler code has been Auto-Differentiated using TAPENADE [Hascoët and Pascual(2004)]
- The discrete adjoint has been derived for the Euler equations.
- The discrete adjoint equations have been hand-coded and validated against auto-differentiated and brute-force gradients.
- Gradients from the adjoint solutions have been applied to multi-variable airfoil optimization.

Summary of Methodology

- An Euler code, mesh generator, and adjoint code have been written in C++ and wrapped in a Python framework for communication.
- The Euler code has been Auto-Differentiated using TAPENADE [Hascoët and Pascual(2004)]
- The discrete adjoint has been derived for the Euler equations.
- The discrete adjoint equations have been hand-coded and validated against auto-differentiated and brute-force gradients.
- Gradients from the adjoint solutions have been applied to multi-variable airfoil optimization.

Summary of Methodology

- An Euler code, mesh generator, and adjoint code have been written in C++ and wrapped in a Python framework for communication.
- The Euler code has been Auto-Differentiated using TAPENADE [Hascoët and Pascual(2004)]
- The discrete adjoint has been derived for the Euler equations.
- The discrete adjoint equations have been hand-coded and validated against auto-differentiated and brute-force gradients.
- Gradients from the adjoint solutions have been applied to multi-variable airfoil optimization.

Comparing Hand-Coded vs. Auto-Diff Adjoint

- Use of TAPENADE very convenient for auto-differentiation however
  - Unable to auto-differentiate certain, more complicated section of code (ie. Implicit routines of the Euler solver)
  - Creates complex code more difficult to read, optimize, and parallelize with OpenMP
- Hand-coded Adjoint was challenging to implement but gives full control over memory usage and code optimization

	Auto-Diff	By-hand	By-hand + OpenMP
Time (s)	126.3	120.5	15.1
Speedup	1.0	1.05	8.36

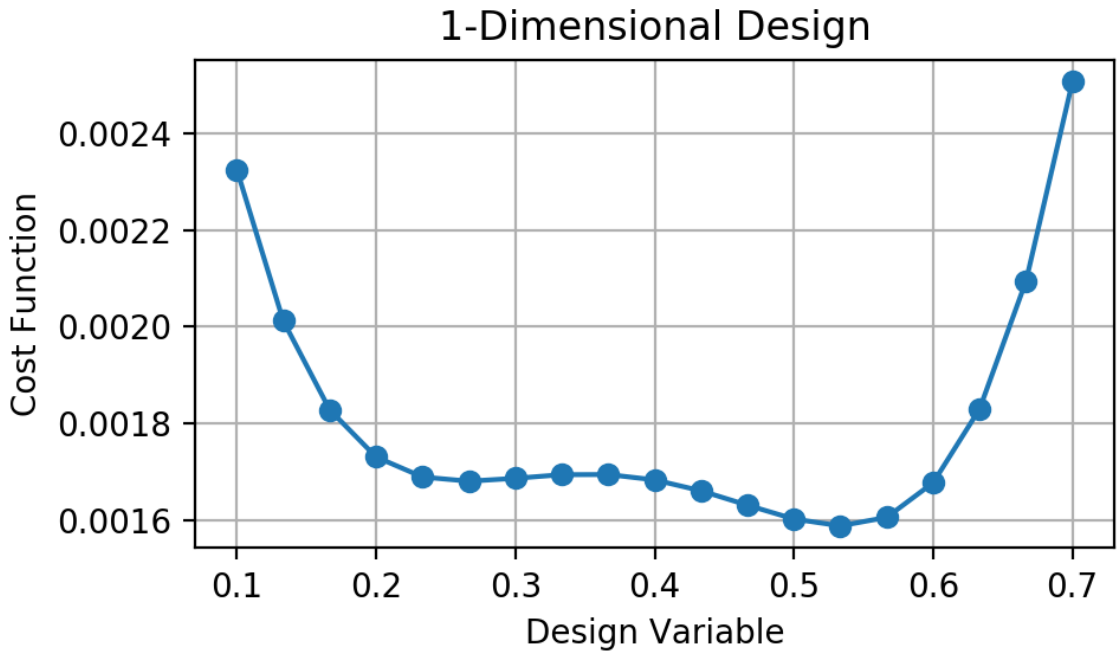
Timing Results for 10000 Adjoint iterations (8-core CPU)

Optimization Algorithm

- The final step for airfoil optimization is to use the gradients from the adjoint solution in a gradient-based optimization algorithm
  - Entire courses are taught on optimization algorithms
  - Many algorithms are readily available in pre-packaged libraries
- The Scientific Python "SciPy" optimization library has been specifically for use with:
  - Conjugate Gradient (CG) method, variant of the Fletcher-Reeves method [Nocedal and Wright(2006)]
  - Sequential Least SQuares Programming (SLSQP) [Kraft(1988)]

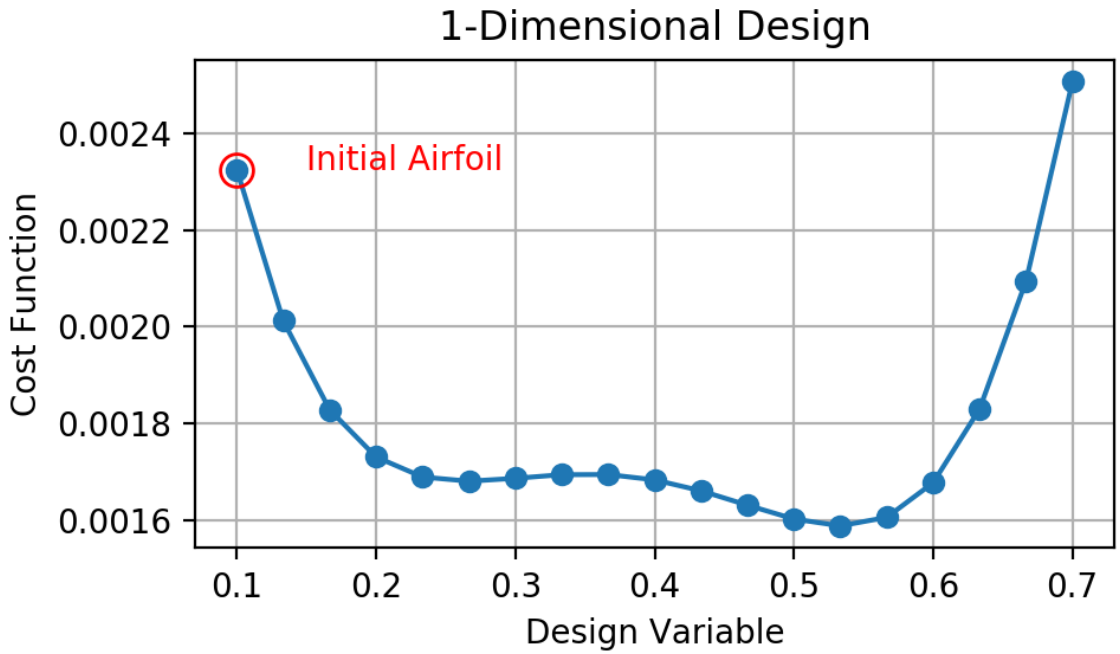
1-Variable Results

Recall the 1-variable case of moving a single bump along the surface of an airfoil

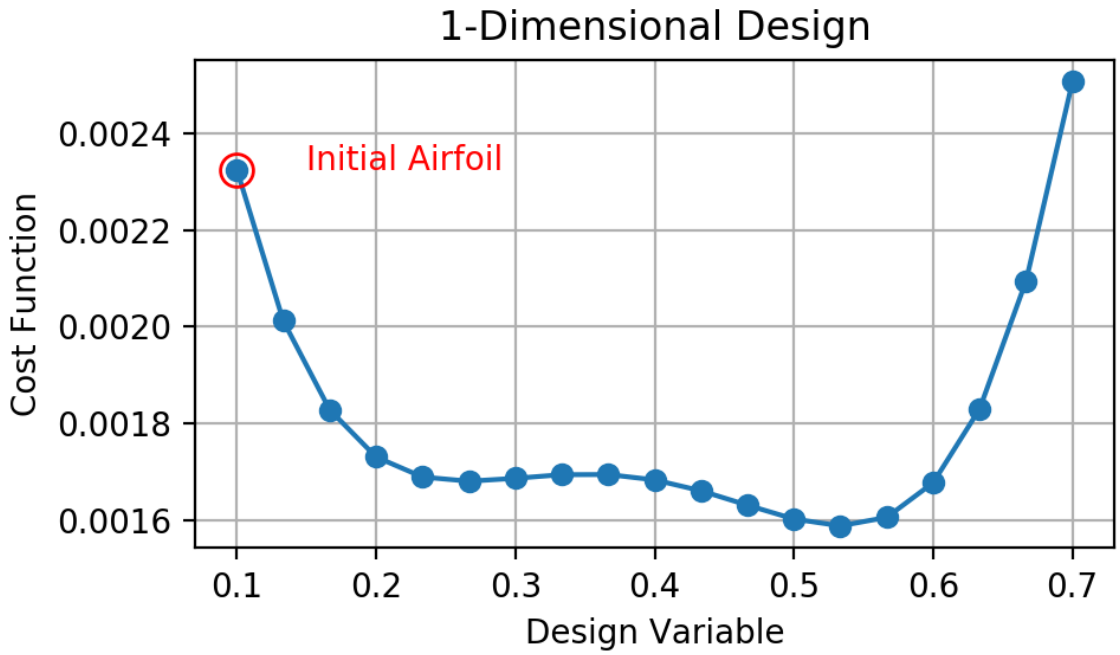


1-Variable Results

Recall the 1-variable case of moving a single bump along the surface of an airfoil



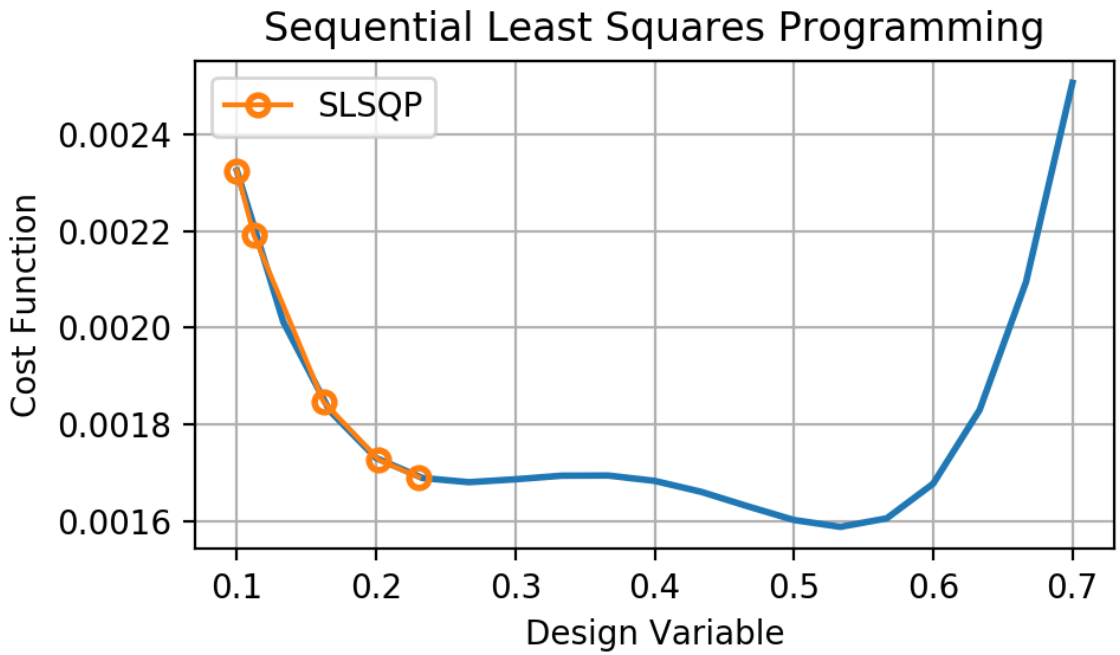
1-Variable Results



	Brute-Force	Auto-Diff Adjoint	By-Hand Adjoint
Gradients	-0.0054687	-0.0054693	-0.0055054

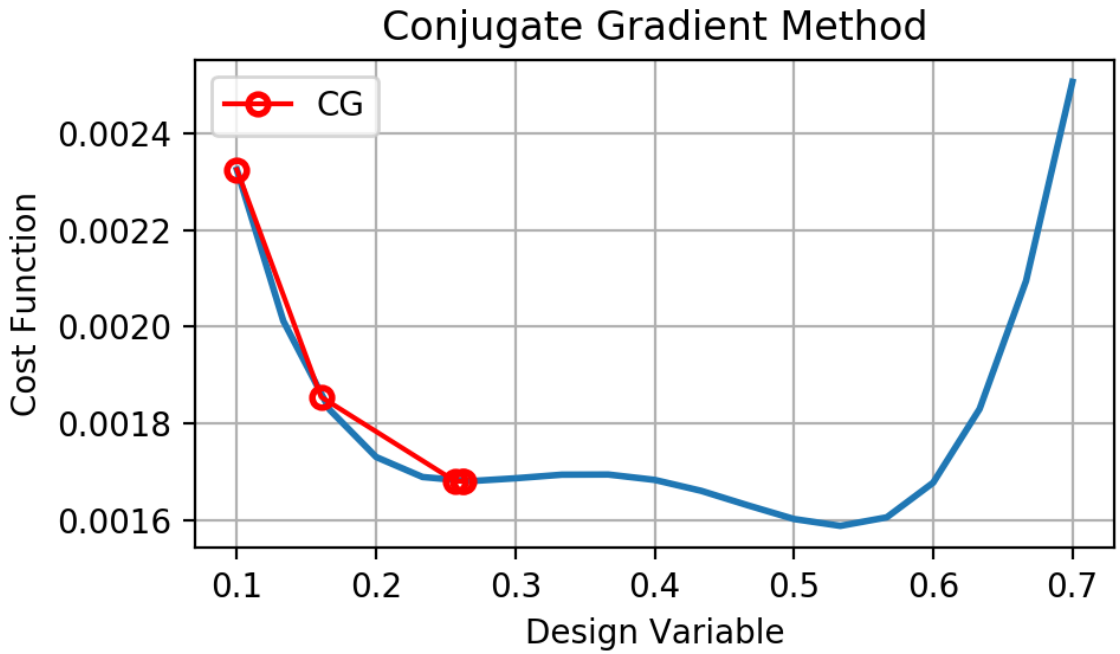
Note: difference in by-hand adjoint from approximation of dissipation flux

Optimization Method: SLSQP



- The Sequential Least SQuares Programming (SLSQP) method for gradient-based optimization is effectively Newton's method for unconstrained problems.
- 4-iterations: 5 Adjoint calls, 5 Euler calls, 209.7 seconds

Optimization Method: Conjugate Gradient

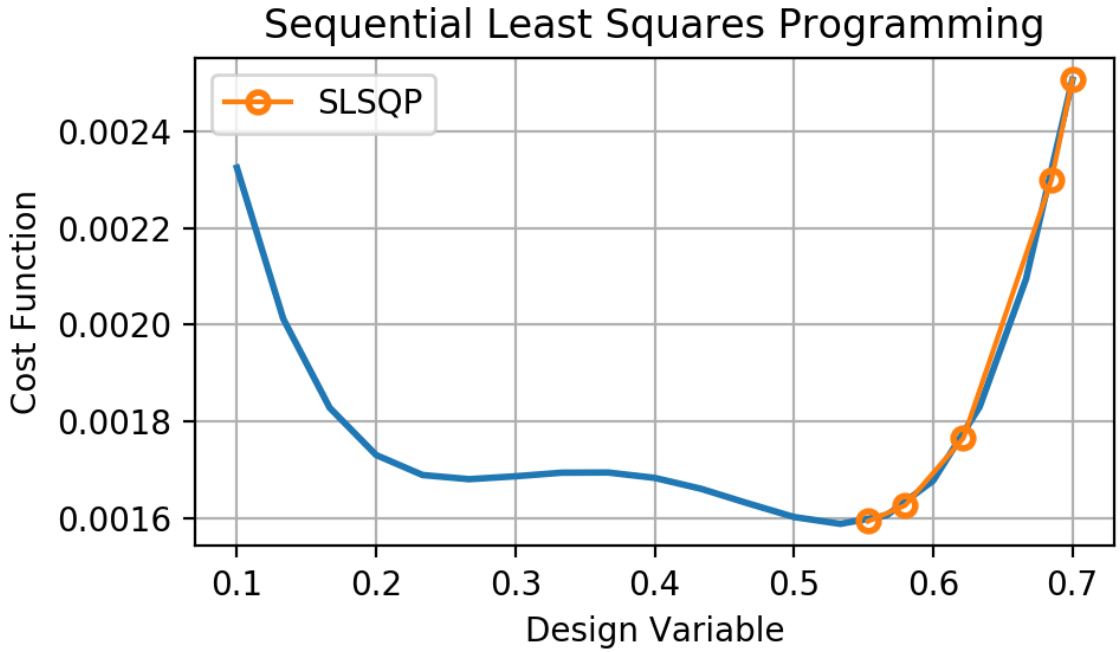


- The Conjugate Gradient (CG) method of non-linear
- Possibly multiple function calls and adjoint calls per iteration.
- 4-iterations: 14 Adjoint calls, 14 Euler calls, 469.7 seconds

Optimization Methods: Comments

- Both CG and SLSQP only found local minima of the single-variable cost function.
- CG converged faster to the solution with 4 “iterations” however each iteration required more function and adjoint calls.
- Because SLSQP required less function and adjoint calls, the 4 iterations ran in less wall-clock time.

Optimization: Local Minima



- By changing the location of the initial guess, a different local minima is reached

6-Variable Results

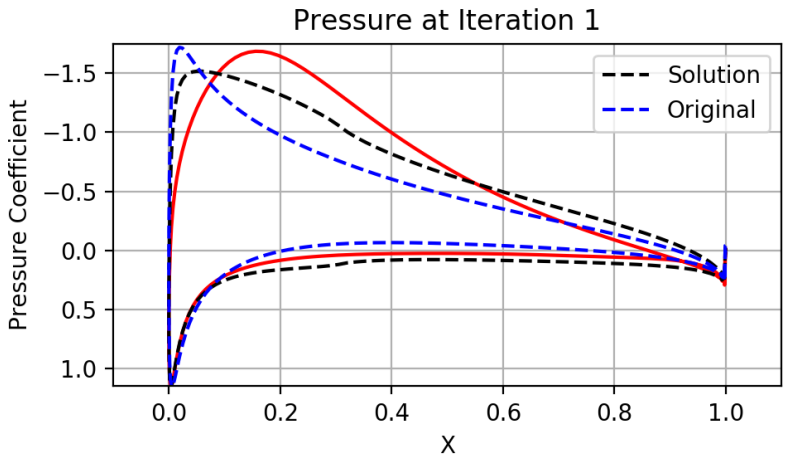
- 3 bumps on the lower surface, 3 on the upper surface
- Location of the bumps fixed at 25%, 50%, and 75% along the airfoil
- Only bump amplitudes are changed $\rightarrow$ 6 total design variables

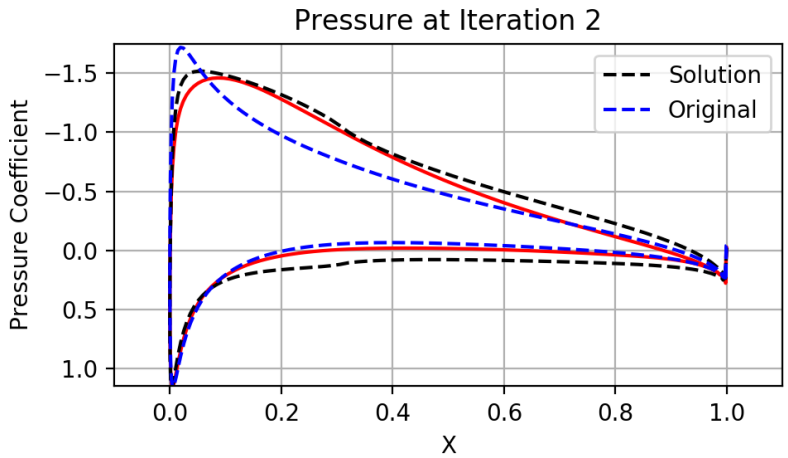
Initial Gradients:

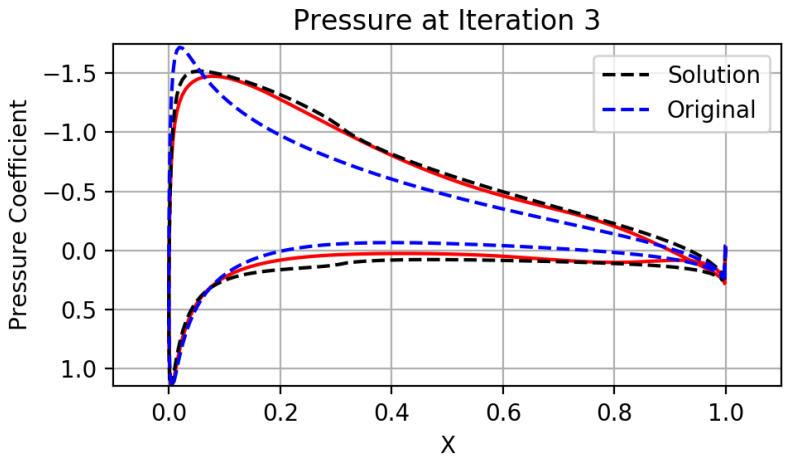
	$\partial I/\partial\alpha_1$	$\partial I/\partial\alpha_2$	$\partial I/\partial\alpha_3$	$\partial I/\partial\alpha_4$	$\partial I/\partial\alpha_5$	$\partial I/\partial\alpha_6$
Brute-Force	-0.8495	-0.4387	-0.1935	-3.588	-0.8853	-0.1000
AD Adjoint	-0.8495	-0.4389	-0.1937	-3.588	-0.8855	-0.1003
Adjoint	-0.8549	-0.4365	-0.1851	-3.635	-0.8902	-0.0946

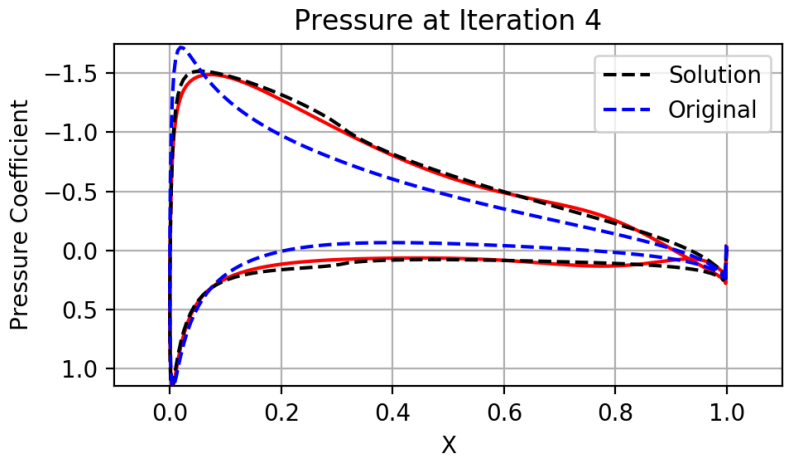
Note: Discrepancy between AD adjoint and by-hand adjoint from approximation of dissipation terms

SLSQP 6-variable Results

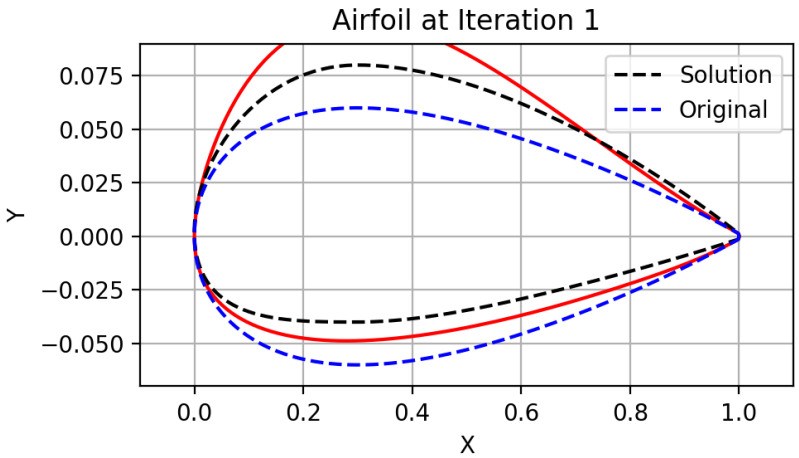


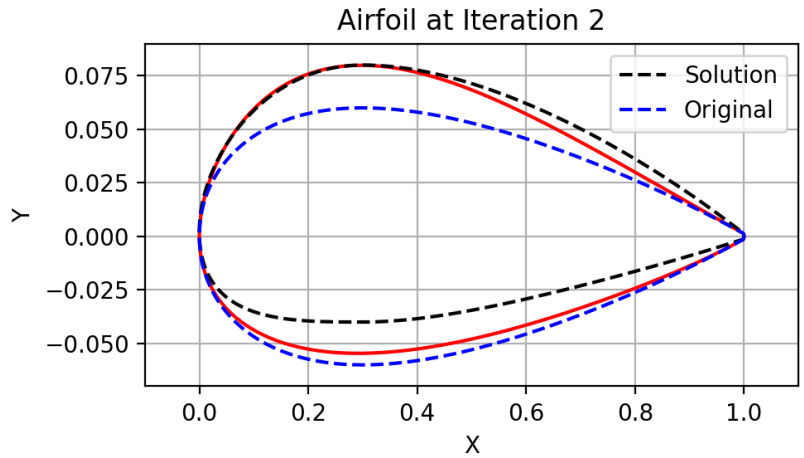


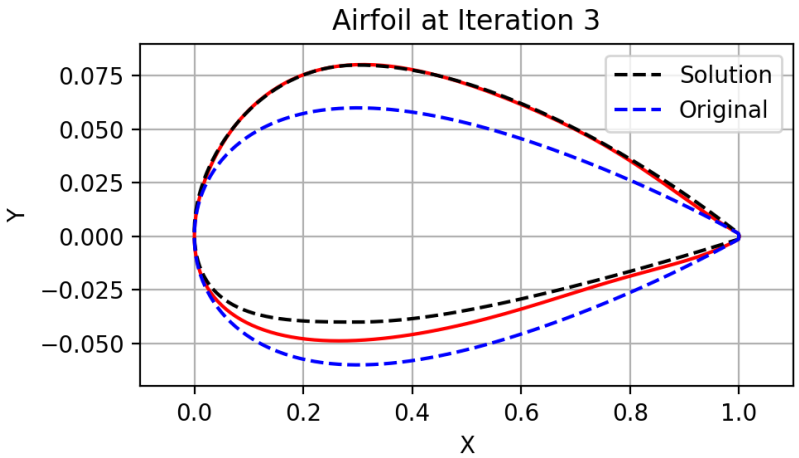


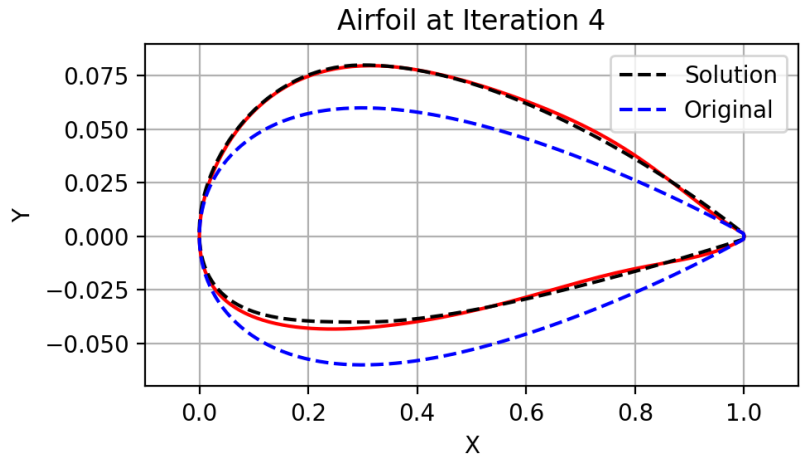


SLSQP 6-variable Results



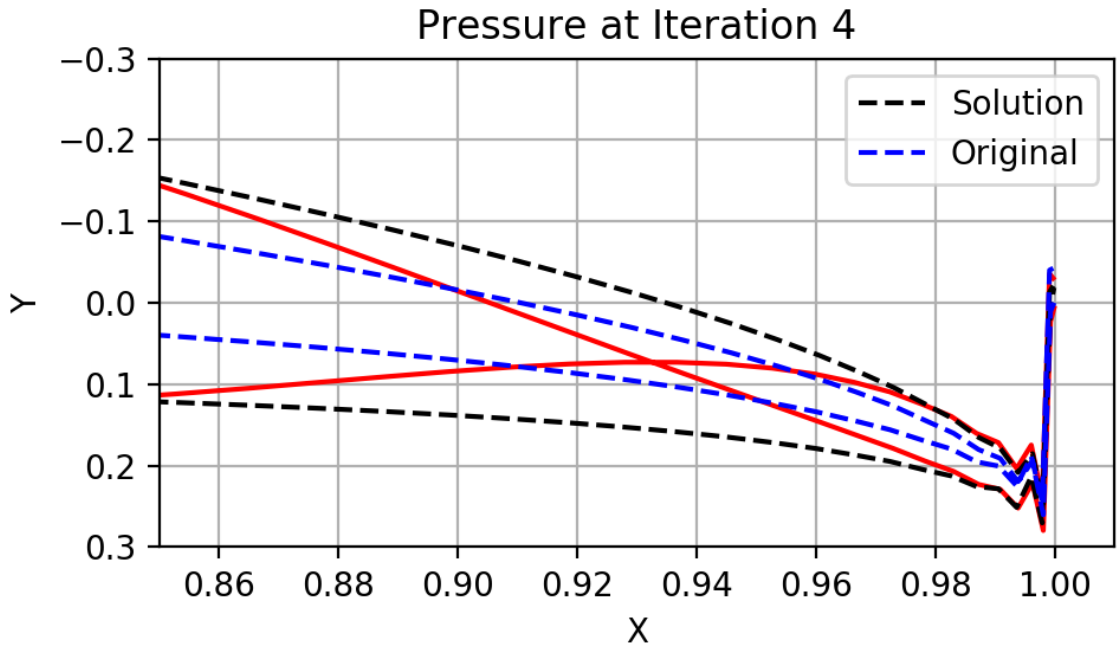






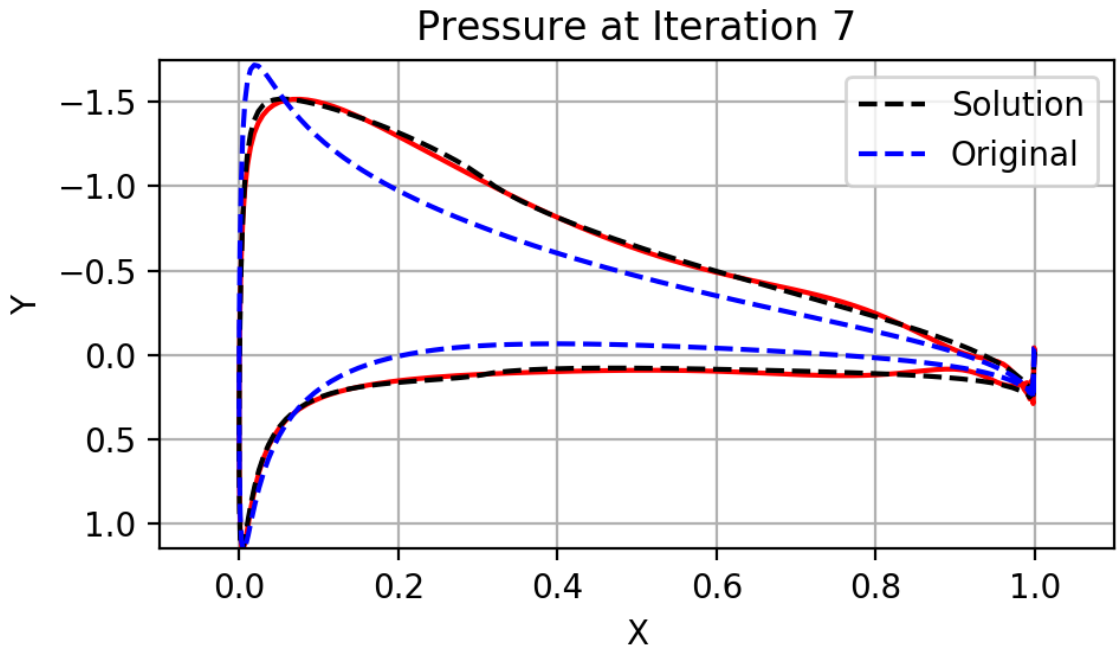
Multi-variable Results: Comments

- Pressure matched relatively well except at trailing edge
- By adding additional bumps near the trailing edge this should help the solution



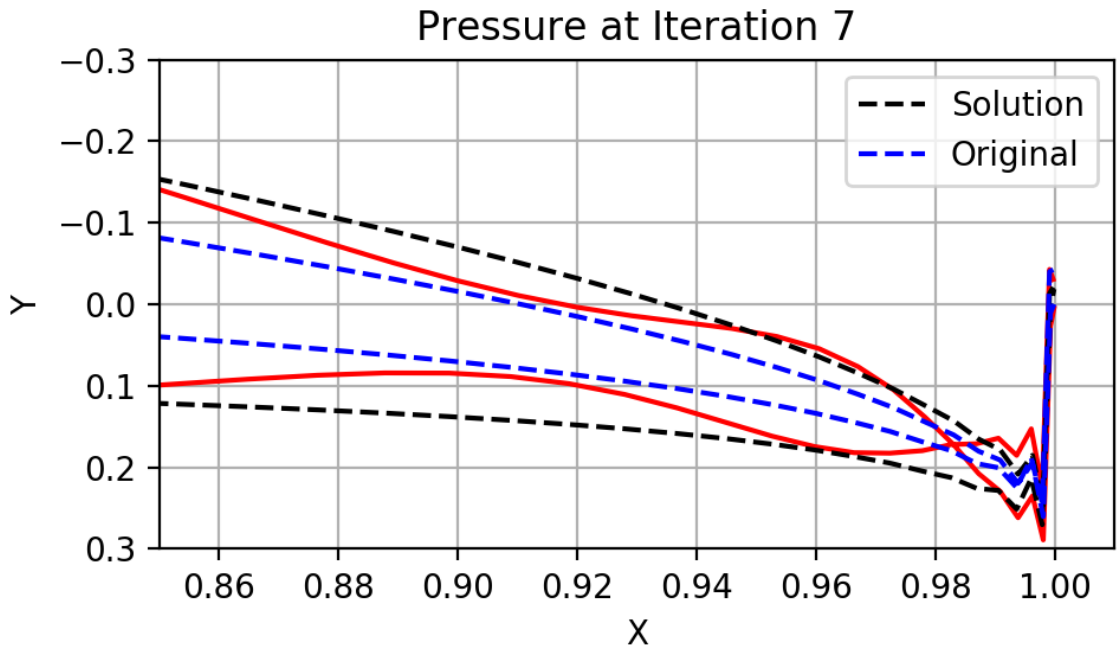
8-Variable Results

One more bump added on both the upper and lower surface at 95% along airfoil helps to reduce the error at the trailing edge. The solution after 7 SLSQP iterations is shown below.



8-Variable Results

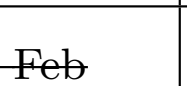
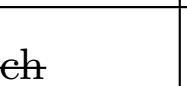
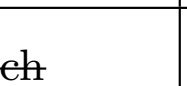
One more bump added on both the upper and lower surface at 95% along airfoil helps to reduce the error at the trailing edge. The solution after 7 SLSQP iterations is shown below.



Summary / Discussion

- Use of adjoint methods significantly reduces the cost of computing gradients of a cost function where the Euler equations must be solved.
- Use of both the auto-differentiated and by-hand Adjoint codes provide accurate gradients of a cost function with respect to a set of variables.
- These gradients can be used with gradient-based optimization methods to approach local minima of the cost function
  - Converging to a local minima is sufficient for aerodynamic shape optimization problems since commonly engineers are interested in incremental changes from an initial, trusted guess.

Milestones

Functioning airfoil perturbation function in combination with mesh generation and 2D Euler Solver.	Late Oct	
Functioning brute-force method for sensitivity of Pressure cost function to airfoil perturbation variables.	Early Nov	
Auto-differentiation of Euler CFD solver.	Late Nov	
Validate auto-diff and brute-force method for simple reverse-design perturbations.	Mid Dec	
Hand-coded explicit discrete adjoint solver.	~~Mid Feb~~ March 6	
~~Implicit routine~~ OpenMP Acceleration for discrete adjoint solver.	~~March~~ Early April	
Validate discrete adjoint solver against auto-diff and brute-force methods.	~~March~~ Mid April	
Test discrete adjoint solver with full reverse-design cases.	~~Mid April~~ Early May	

Deliverables

- Bitbucket GIT Repository of
  - Euler, Grid-Generation, and Airfoil perturbation code in Python Framework
  - Auto-differentiated Adjoint Code
  - Hand-differentiated Adjoint Code
- Equations of hand-derived Adjoint relations for flux and dissipation terms.
- Sample run files for Euler-only, Adjoint-only, and full design case.

References I

[Hicks and Henne(1977)] R. Hicks and P. Henne.
Wing design by numerical optimization.
Aircraft Design and Technology Meeting. American Institute of Aeronautics and Astronautics, Aug 1977.
doi: 10.2514/6.1977-1247.
URL http://dx.doi.org/10.2514/6.1977-1247.

[Nadarajah and Jameson(2002)] Siva Nadarajah and Antony Jameson.
Optimal Control of Unsteady Flows Using a Time Accurate Method.
Multidisciplinary Analysis Optimization Conferences, (June):—-, 2002.
doi: 10.2514/6.2002-5436.
URL http://dx.doi.org/10.2514/6.2002-5436.

[Hascoët and Pascual(2004)] Laurent Hascoët and Valérie Pascual.
TAPENADE 2.1 user's guide.
2004.
URL http://www.inria.fr/rrrt/rt-0300.html.

[Nocedal and Wright(2006)] Jorge Nocedal and Stephen J. Wright.
Numerical optimization.
Springer series in operations research and financial engineering. Springer, New York, 2006.

References II

[Kraft(1988)] Dieter Kraft.
A software package for sequential quadratic programming.
Technical Report DFVLR-FB 88-28. Institut fur Dynamik der Flugsysteme, Oberpfaffenhofen, July 1988.

[Nadarajah(2003)] S. Nadarajah.
The Discrete Adjoint Approach to Aerodynamic Shape Optimization.
Stanford University PhD Dissertation, 2003.

Appendex

Additional Slides

Comments: Dissipation

- The main reason for the mismatch between the auto-differentiation result and by-hand result is the dissipation in the flux routine
- If I remove the dissipation from the auto-differentiated code, I get exactly the same sensitivity

Looking at just the j-direction:

$$R_j = -\left(f_{j+\frac{1}{2}} + f_{j-\frac{1}{2}}\right) - \left(h_{j+\frac{1}{2}} + h_{j-\frac{1}{2}}\right)$$

Where the scalar dissipation term $h_{j+\frac{1}{2}}$ is:

$$h_{j+\frac{1}{2}} = \epsilon\sigma\left(q_{j+1} - q_j\right)$$

Dissipation Approximations

$$h_{j+\frac{1}{2}} = \epsilon\sigma\left(q_{j+1} - q_j\right)$$

ϵ is a constant, approximately 0.25 [Nadarajah(2003)]. σ is the spectral radius scaled by the face area:

$$\sigma = ||V|| + c||S_{j+\frac{1}{2}}||$$

and c is the speed of sound. Both V and c are functions of the flow q however according to Nadarajah [Nadarajah(2003)], σ does not vary significantly and can be considered constant in deriving the adjoint dissipation terms. While this may be the case for the variation of the residual with respect to q, my results show this is not the case for the variation of the residual with respect to X.