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

Dylan Jude
Graduate Research Assistant

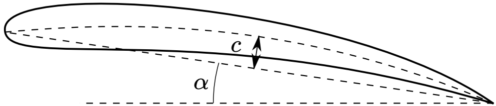
University of Maryland
AMSC 663/664

December 8, 2016

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 proposed project focuses on the use of adjoint methods for two-dimensional airfoil shape optimization using Computational Fluid Dynamics to solve the steady Euler equations.

Background Refresher



Airfoil Example Problem

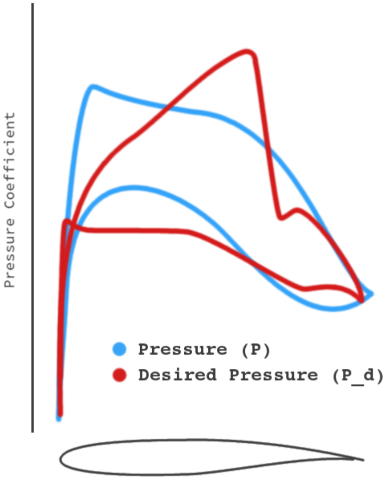
Given n design variables $\alpha_1, \alpha_2, \alpha_3 ... \alpha_n$ we can achieve a change in airfoil shape:

Background Refresher

We want to minimize a cost function I_c in the design process

Mathematically:

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



[Nadarajah and Jameson(2002)]

Background Refresher

We want the sensitivity of the cost function 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 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 in with the cost of a single CFD flow-computation.

Background Refresher

We want the sensitivity of the cost function 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 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 in with the cost of a single CFD flow-computation.

Background Refresher

We want the sensitivity of the cost function 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 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 in with the cost of a single CFD flow-computation.

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 Jan	
Implicit routine for discrete adjoint solver.	Early Feb	
Validate discrete adjoint solver against auto-diff and brute-force methods.	Late Feb	
Test discrete adjoint solver with full reverse-design cases.	Mid Mar	

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 Jan	
Implicit routine for discrete adjoint solver.	Early Feb	
Validate discrete adjoint solver against auto-diff and brute-force methods.	Late Feb	
Test discrete adjoint solver with full reverse-design cases.	Mid Mar	

Milestone: Late October

Mesh Generation:

```
# ----------------------------------------
# Airfoil Surface
#
ktot = 64
half = 93
airfoil = naca.naca4('0012', half, False, True)

# ----------------------------------------
# Mesh Generation
#
mg   = libflow.MeshGen(airfoil, ktot, 5.0)
mg.poisson(500)
xy   = mg.get_mesh()
```

Source Terms to Mesh-Generation Equations

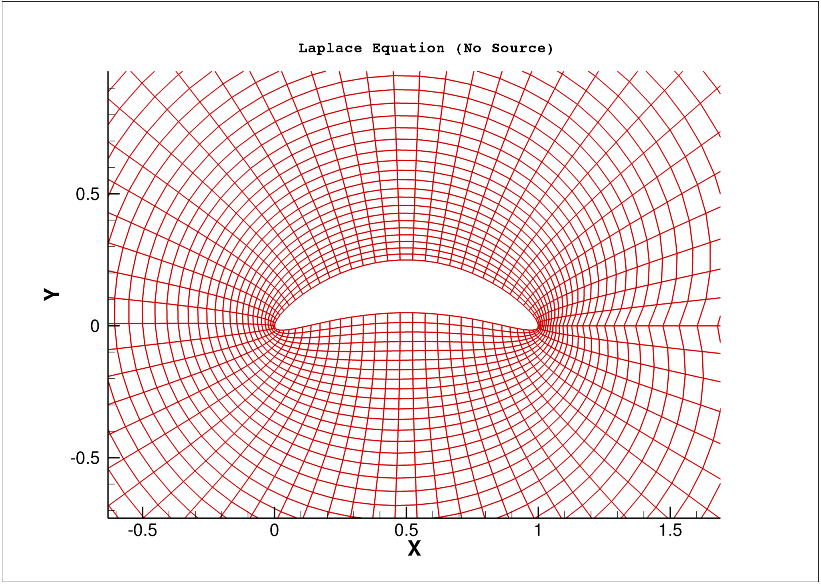
2-Dimensional mesh generation is traditionally done by solving the Poisson equation:

$$\xi_{xx}+\xi_{yy}=P$$
$$\eta_{xx}+\eta_{yy}=Q$$

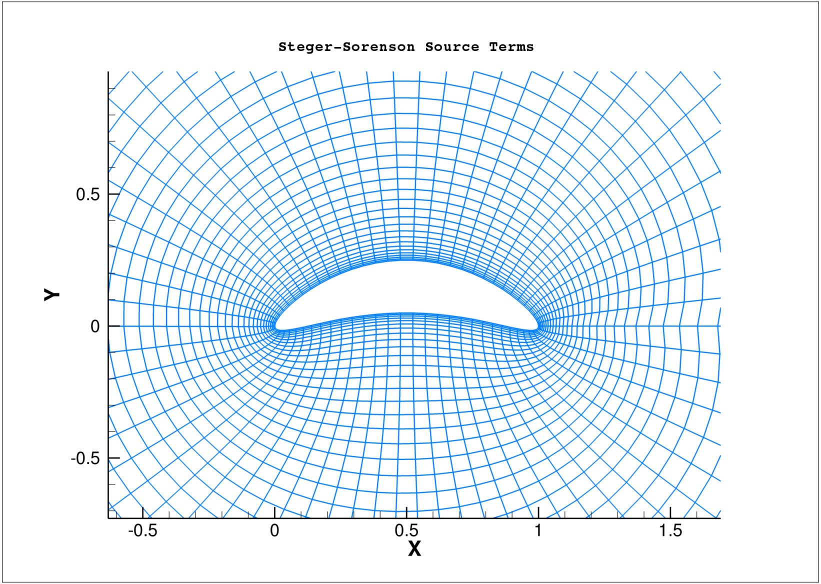
Where ξ and η are coordinates of a mapped, equispaced grid (beyond the scope of this project).

Previous solver was without P and Q (Laplace equation). Steger and Sorenson [Steger and Sorenson(1979)] suggest source terms for P and Q to improve the grid quality near deformed surfaces.

Comparing Mesh-Generator Source Terms



Comparing Mesh-Generator Source Terms



Milestone: Late October

Airfoil Perturbation: [Hicks and Henne(1977)]

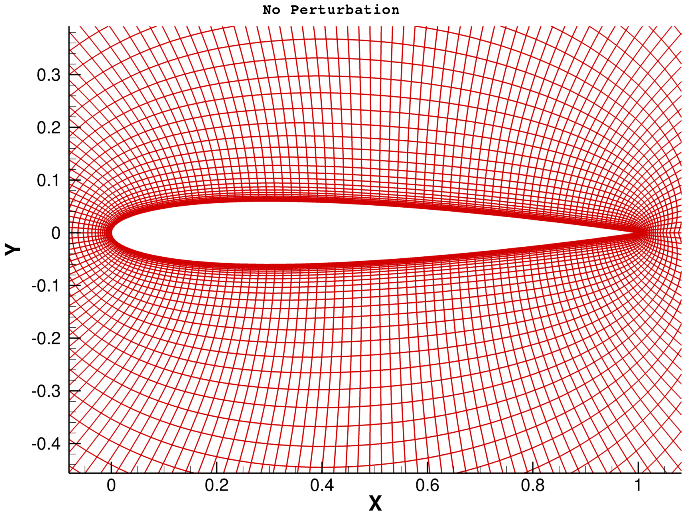
$$b(x) = a\left[sin\left(\pi x^{\frac{log(0.5)}{log(t_1)}}\right)\right]^{t_2}, \quad \text{for } 0 \leq x \leq 1$$

t_1 locates the maximum of the “bump” in $0 \leq x \leq 1$
t_2 controles the width of the “bump”

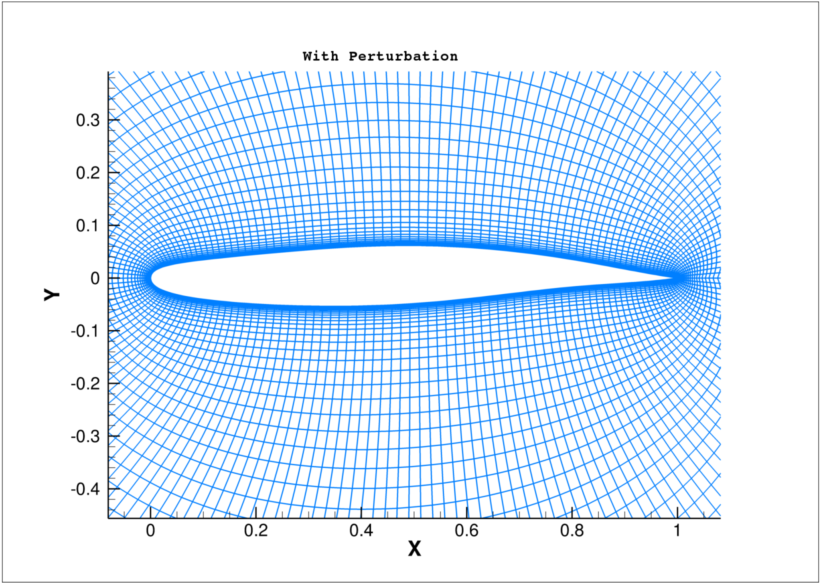
```
#----------------------------------------
# Hicks Henne Perturbation
#
design_vars = np.array([[ 0.25,  0.50 , 0.75 ],
                        [ 0.25,  0.50 , 0.75 ],
                        [ 0.01, -0.005, 0.01 ],
                        [-0.02,  0.01 , 0.005]])

airfoil     = perturb(airfoil,design_vars)
```

Airfoil Perturbations



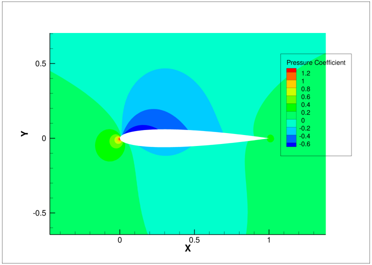
Airfoil Perturbations



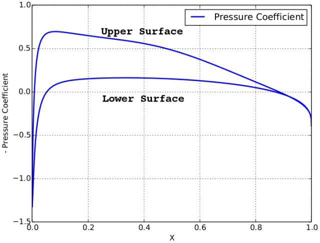
Milestone: Late October

Euler Solver:

```
# ------------------------------------------------
# Start CFD
inputs = euler_utils.read_inputs("input.yaml")
euler  = libflow.Euler(grid, yaml.dump(inputs))
euler.take_steps(1000)
pressure  = euler.pressure()
```



Pressure coefficient contours around the airfoil



Pressure coefficient distribution over the airfoil

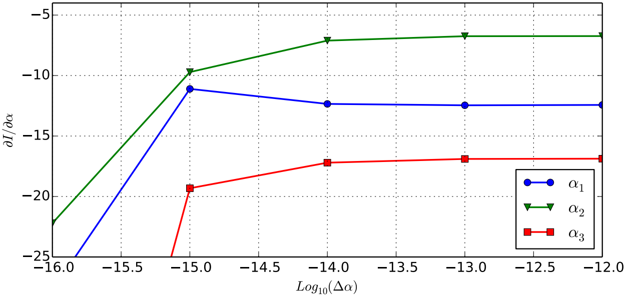
Milestones

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 Jan	
Implicit routine for discrete adjoint solver.	Early Feb	
Validate discrete adjoint solver against auto-diff and brute-force methods.	Late Feb	
Test discrete adjoint solver with full reverse-design cases.	Mid Mar	

Milestone: Early November

Functioning brute-force method for sensitivity of Pressure cost function to 3 airfoil perturbation variables: $\alpha_1, \alpha_2, \alpha_3$

$$\frac{\partial I_c}{\partial \alpha_i} = \frac{I_c(\alpha_i + \Delta\alpha_i) - I_c(\alpha_i)}{\Delta\alpha_i}, \quad i = 1, 2, 3$$



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 Jan	
Implicit routine for discrete adjoint solver.	Early Feb	
Validate discrete adjoint solver against auto-diff and brute-force methods.	Late Feb	
Test discrete adjoint solver with full reverse-design cases.	Mid Mar	

Milestone: Late November

In auto-differentiation of Euler CFD solver, define the "dot" and "bar" operators:

$$\dot{x} := \frac{\partial x}{\partial \alpha} \qquad \bar{x} := \left(\frac{\partial x}{\partial I}\right)^T \quad \text{for } x \text{ in } \{\alpha, X, Q, I\}$$

Foreward	$\dot{\alpha}$	$\rightarrow$	$\dot{X}$	$\rightarrow$	$\dot{Q}$	$\rightarrow$	$\dot{I}$
Reverse	$\bar{\alpha}$	$\leftarrow$	$\bar{X}$	$\leftarrow$	$\bar{Q}$	$\leftarrow$	$\bar{I}$
	design vars		grid		flow soln		cost func

Using the *Tapenade* suite of auto-differentiation software, can auto differentiate in forward or reverse mode.

Milestone: Late November

In auto-differentiation of Euler CFD solver, define the "dot" and "bar" operators:

$$\dot{x} := \frac{\partial x}{\partial \alpha} \qquad \bar{x} := \left(\frac{\partial x}{\partial I}\right)^T \quad \text{for } x \text{ in } \{\alpha, X, Q, I\}$$

Foreward	$\dot{\alpha}$	$\rightarrow$	$\dot{X}$	$\rightarrow$	$\dot{Q}$	$\rightarrow$	$\dot{I}$
Reverse	$\bar{\alpha}$	$\leftarrow$	$\bar{X}$	$\leftarrow$	$\bar{Q}$	$\leftarrow$	$\bar{I}$
	design vars		grid		flow soln		cost func

Using the *Tapenade* suite of auto-differentiation software, can auto differentiate in forward or reverse mode.

Milestone: Late November

In auto-differentiation of Euler CFD solver, define the "dot" and "bar" operators:

$$\dot{x} := \frac{\partial x}{\partial \alpha} \qquad \bar{x} := \left(\frac{\partial x}{\partial I}\right)^T \quad \text{for } x \text{ in } \{\alpha, X, Q, I\}$$

Foreward	$\dot{\alpha}$	$\rightarrow$	$\dot{X}$	$\rightarrow$	$\dot{Q}$	$\rightarrow$	$\dot{I}$
Reverse	$\bar{\alpha}$	$\leftarrow$	$\bar{X}$	$\leftarrow$	$\bar{Q}$	$\leftarrow$	$\bar{I}$
	design vars		grid		flow soln		cost func

Using the *Tapenade* suite of auto-differentiation software, can auto differentiate in forward or reverse mode.

Tapenade Auto-Differentiation

Create a pre-compilation step in the makefile for reverse-differentiation:

```
pressure_cost_b.c : cost.c
 ${TPN} -reverse                                    \
        -inputlanguage  c                           \
        -outputlanguage c                           \
        -I              ../include                  \
        -head           "pressure_cost(I)/(q)"      \
        -adjfuncname    "_b"                        \
        -o pressure_cost                            \
        cost.c
```

resulting code is often not pretty but readable

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 Jan	
Implicit routine for discrete adjoint solver.	Early Feb	
Validate discrete adjoint solver against auto-diff and brute-force methods.	Late Feb	
Test discrete adjoint solver with full reverse-design cases.	Mid Mar	

Auto-differentiation Results

To simplify the design problem, let's temporarily use a different cost function to look at airfoil lift. This allows us to:

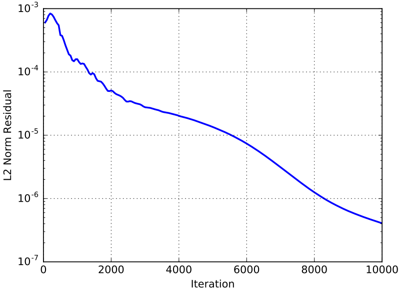
- use a test case for comparison with inviscid thin-airfoil theory
- use a single design variable α = angle-of-attack

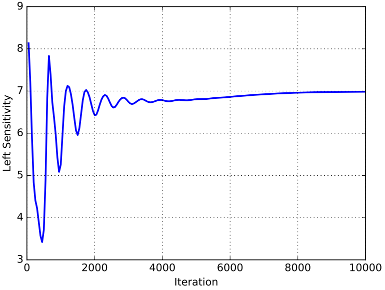
Slightly change our cost function from before

$$I_c(\alpha) = \oint_{airfoil} (P - P_d)^2 \quad \rightarrow \quad I_c(\alpha) = \oint_{airfoil} (-P \cdot d\vec{n})$$

$$\frac{\partial I}{\partial \alpha} = \left(\frac{\partial C_L}{\partial \alpha}\right)_{\text{thin-airfoil theory}} \approx 2\pi \quad \text{for small } \alpha$$

Auto-differentiation Results





Method	$\Delta\alpha$	$\frac{\partial I}{\partial \alpha}$
Brute Force	0.1°	6.9893
Adjoint	-	6.9839
Theory	-	6.2832

Auto-differentiation Results

Case conditions:

Airfoil Thickness	12%
Mach Number	0.5
Angle of attack	2°
Grid Dimensions	187×64

A few comments on these results:

- The 2π result from thin-airfoil theory is for an infinitely thin airfoil in **incompressible flow**.
- Thick airfoils should have $\frac{\partial I}{\partial \alpha} < 2\pi$
- But with increased Mach number $\rightarrow$ $\frac{\partial I}{\partial \alpha} > 2\pi$
- First-order spacial accuracy on relatively coarse mesh

Looking Forward

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 Jan	
Implicit routine for discrete adjoint solver.	Early Feb	
Validate discrete adjoint solver against auto-diff and brute-force methods.	Late Feb	
Test discrete adjoint solver with full reverse-design cases.	Mid Mar	

Thank you!

References I

[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.

[Steger and Sorenson(1979)] J.L. Steger and R.L. Sorenson.
Automatic mesh-point clustering near a boundary in grid generation with elliptic partial differential equations.
Journal of Computational Physics, 33(3):405 – 410, 1979.
ISSN 0021-9991.
doi: http://dx.doi.org/10.1016/0021-9991(79)90165-7.
URL http://www.sciencedirect.com/science/article/pii/0021999179901657.

[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.

Appendix: Hicks Henne Function

With 6 bumps, 12 random variables: 3 t_1, 3 a for each the top and bottom of the airfoil, $t2 = 1.0$