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

Second-Semester Midterm Report

Dylan Jude
Graduate Research Assistant

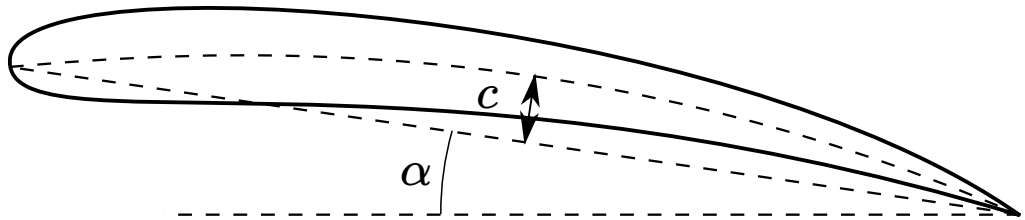
University of Maryland
AMSC 663/664

March 9, 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 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

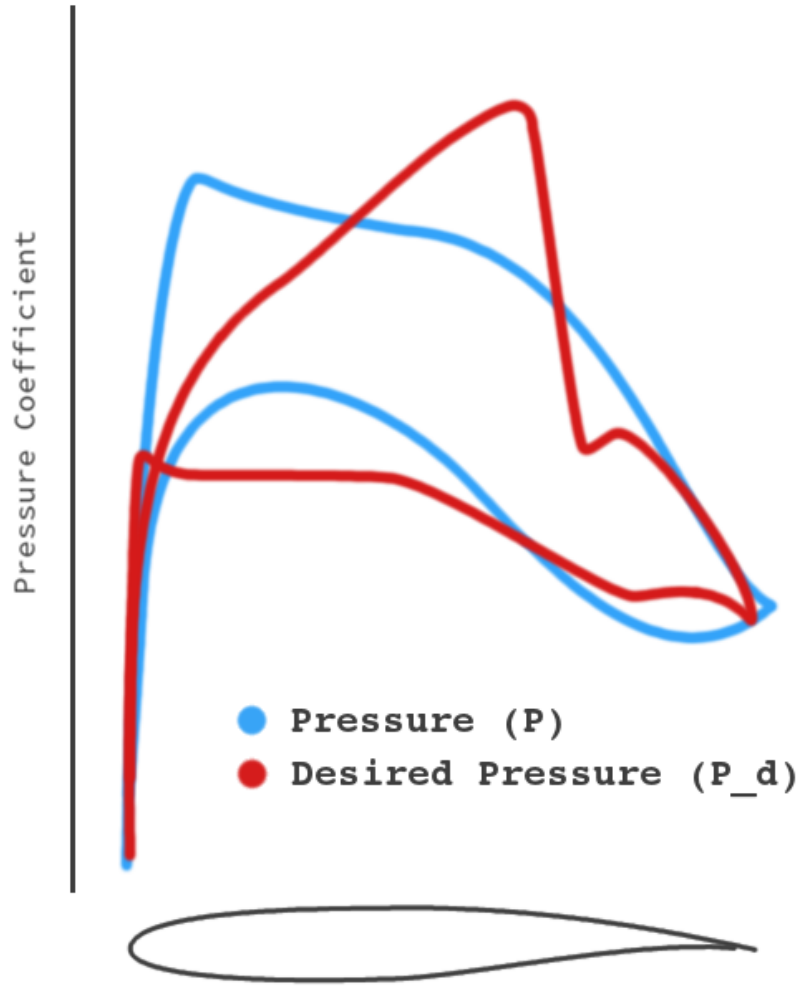
Given n design variables $\alpha_1, \alpha_2, \alpha_3 ... \alpha_n$ we can achieve a change in airfoil shape:

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.

Milestones

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 for discrete adjoint solver.	Early March	
Validate discrete adjoint solver against auto-diff and brute-force methods.	Late March	
Test discrete adjoint solver with full reverse-design cases.	Mid April	

Cost Function Variation

Recall our cost function comparing current and desired pressures:

$$I(\alpha) = I(q, X) = \oint_{airfoil} (P - P_d)^2$$

The variation of a cost function I can be broken down into the sensitivity to the flow variables q and the grid coordinates X:

$$\partial I = \left[\frac{\partial I}{\partial q}\right] \delta q + \left[\frac{\partial I}{\partial X}\right] \delta X = 0$$

Adjoint Formulation

Combine the variation of the cost function:

$$\partial I = \left[\frac{\partial I}{\partial q}\right] \delta q + \left[\frac{\partial I}{\partial X}\right] \delta X$$

with the variation of the discrete Euler residual (detailed in following slides):

$$\partial R = \left[\frac{\partial R}{\partial q}\right] \delta q + \left[\frac{\partial R}{\partial X}\right] \delta X = 0$$

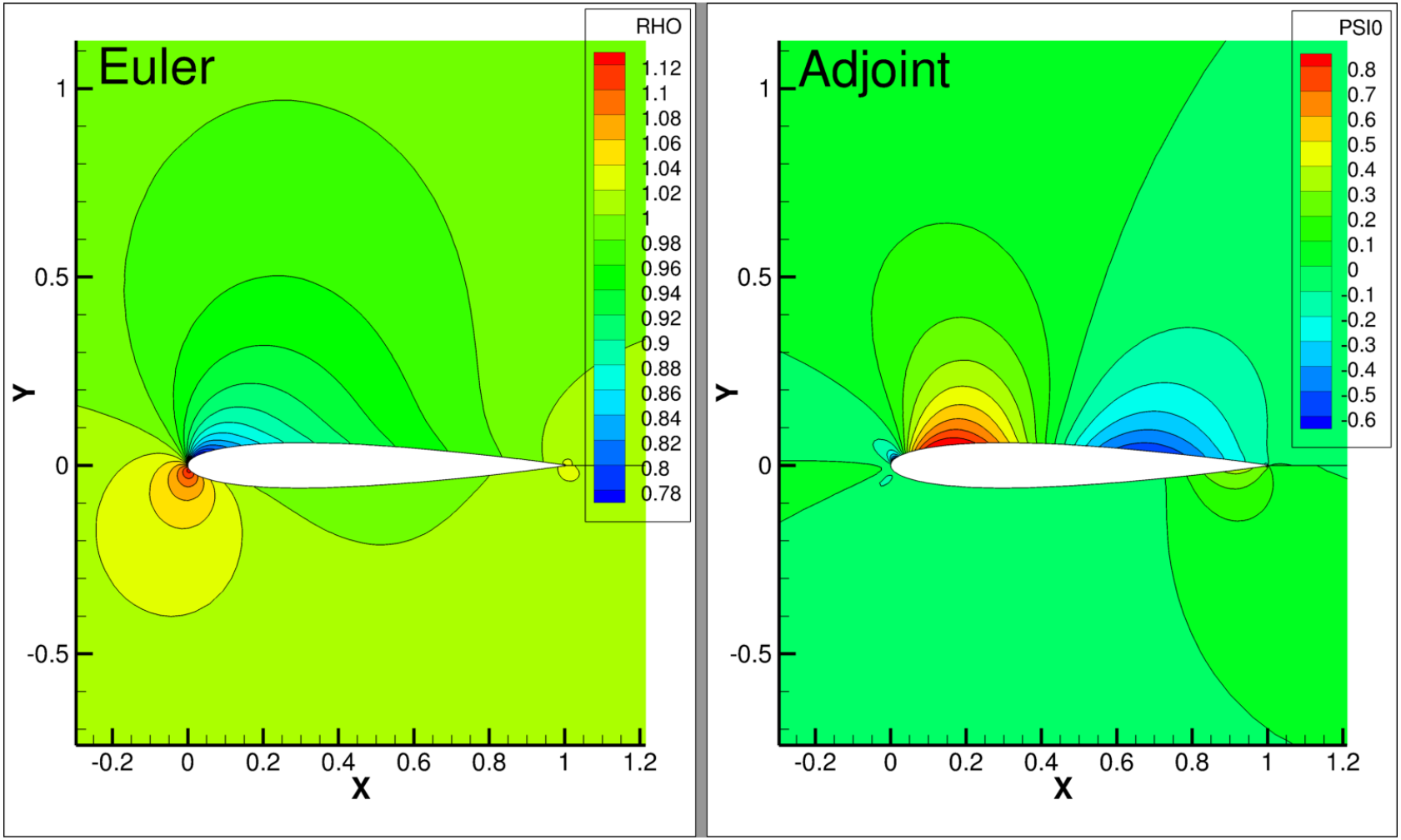
and adding a Lagrange multiplier ψ:

$$\delta I = \delta I_q + \delta I_X - \psi(\delta R_q + \delta R_X)$$

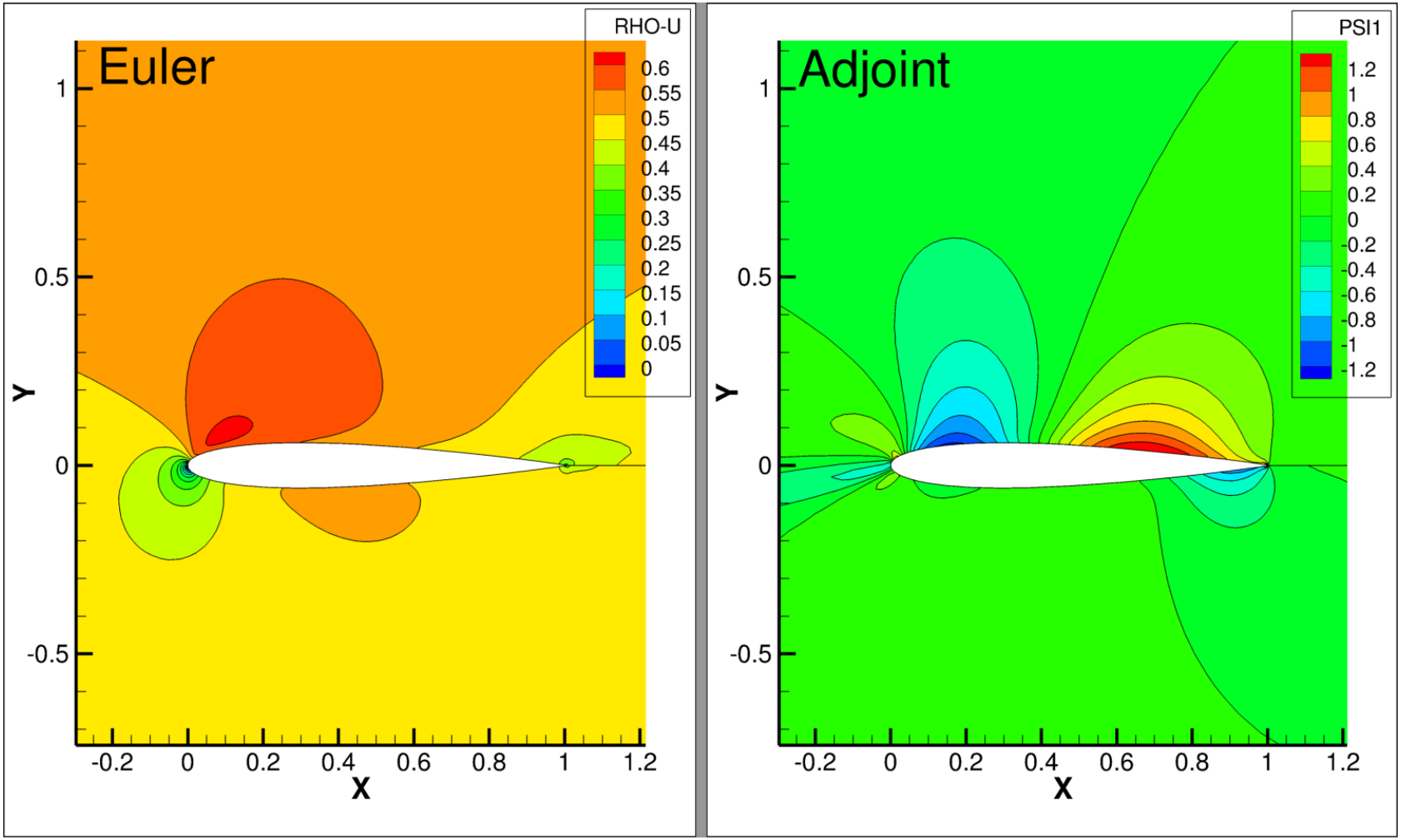
we want to eliminate dependince on q, so that

$$\boxed{\delta I_q - \psi(\delta R_q) = 0} \tag{1}$$

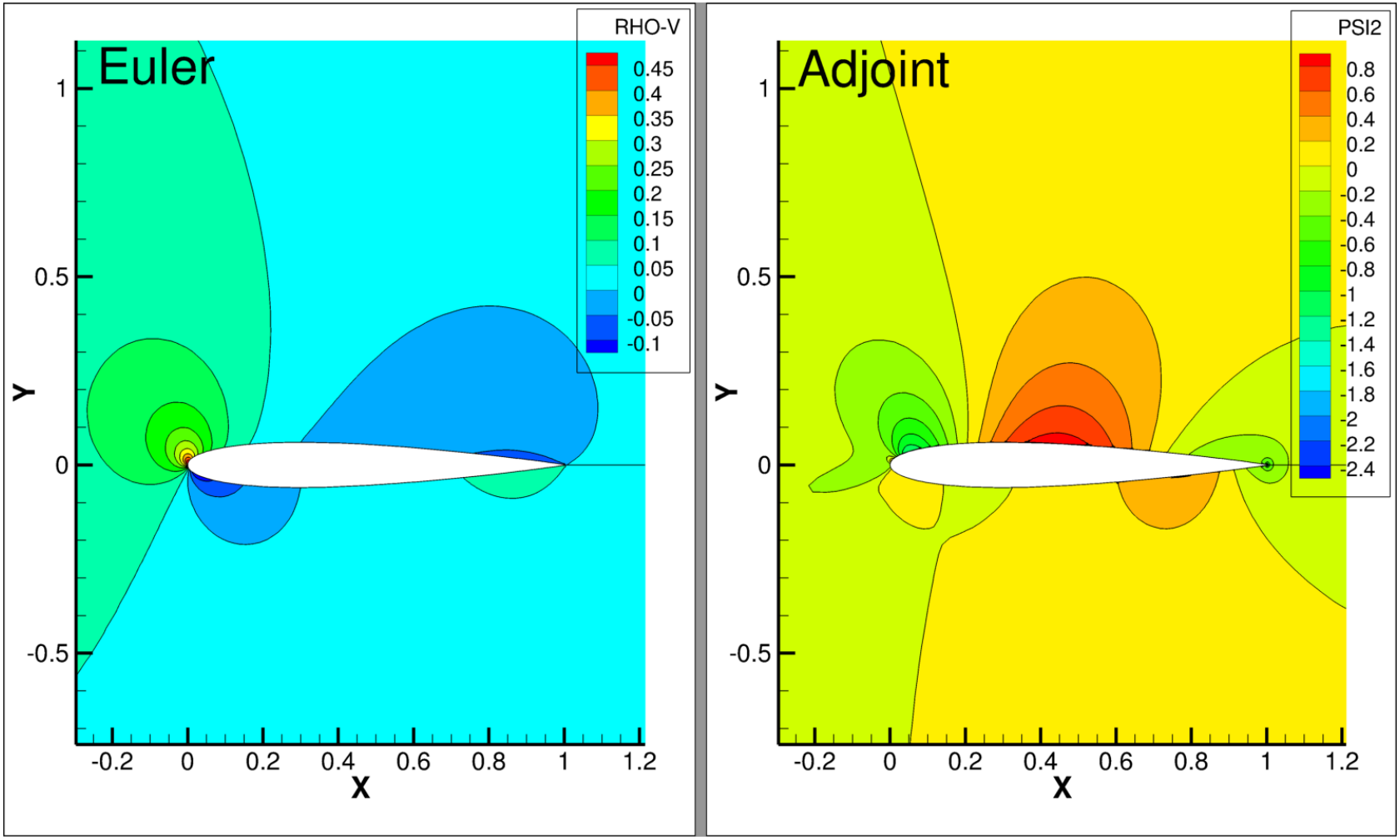
Plotted Adjoint Solution



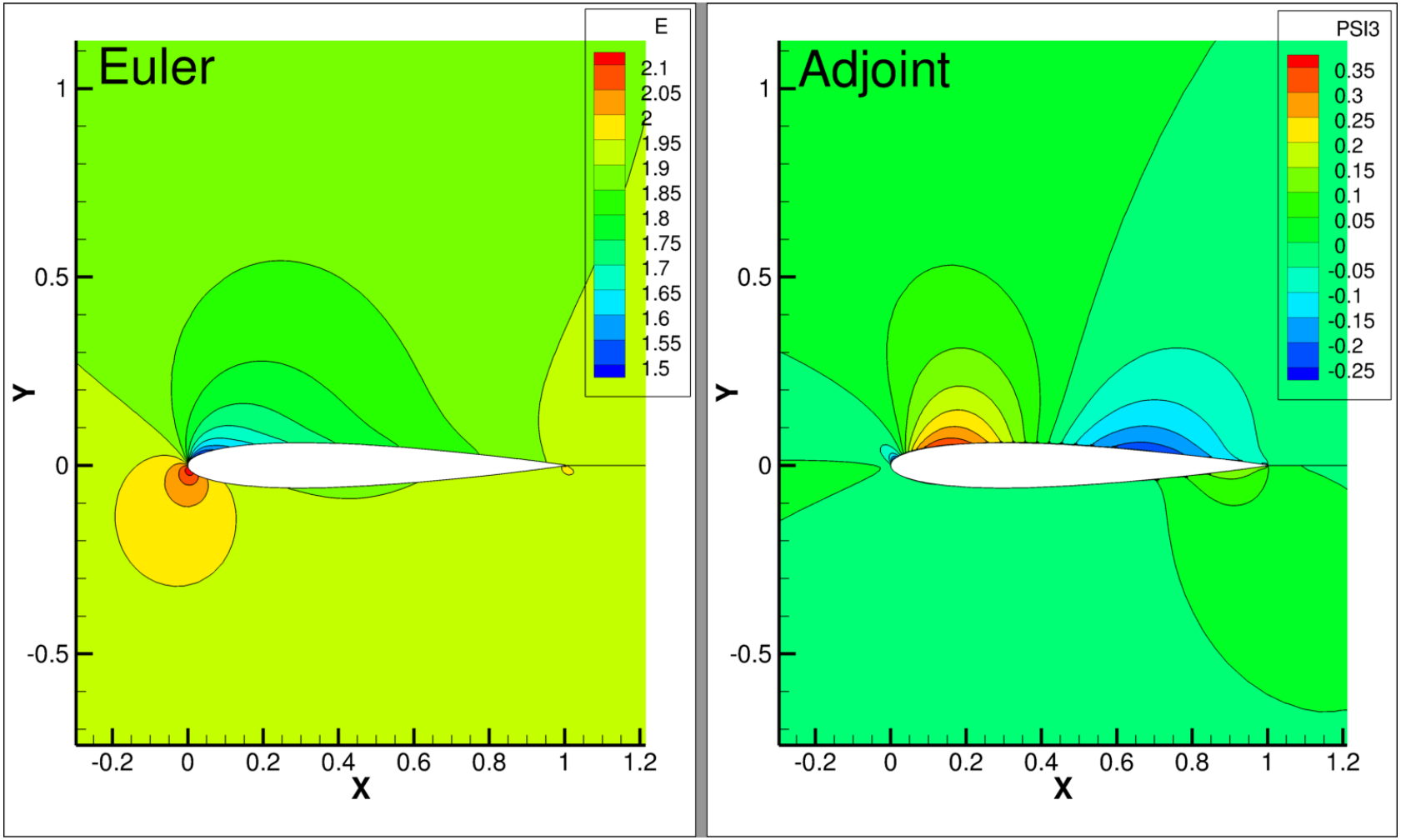
Plotted Adjoint Solution



Plotted Adjoint Solution



Plotted Adjoint Solution



Discrete Euler Equations

Recall the Euler equations in coordinate directions ξ:

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

$$q = J^{-1} \begin{bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ e \end{bmatrix}, \quad f_c = J^{-1} \begin{bmatrix} \rho V_1 \\ \rho u_1 V_1 + \xi_{1,1} p \\ \rho u_2 V_1 + \xi_{1,2} p \\ (e+p) V_1 \end{bmatrix} \tag{3}$$

$$V_i = u_1 \xi_{i,1} + u_2 \xi_{i,2} \tag{4}$$

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:

$$R^n = \frac{q^{n+1} - q^n}{\Delta t} = 0 \tag{5}$$

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

$$R^n_{j,k} = -\left(f_{j+1/2,k} - f_{j-1/2,k}\right) - \left(g_{j,k+1/2} - g_{j,k-1/2}\right) \tag{6}$$

In addition there are artificial dissipation terms that are not shown here (presented at the end of this presentation, time permitting).

Discrete Adjoint Formulation

$$\delta I_q - \psi(\delta R_q) = 0$$

This becomes the new PDE that we want to solve. The term $\psi(\delta R_q)$ is what was auto-differentiated, since R can be viewed as a computer function of variables q.

By hand, however, it is not trivial:

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

Discrete Adjoint Formulation

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

$$\begin{aligned}\partial R^n_{j,k} = &-\left(\frac{1}{2}(\partial f_{j,k} + \partial f_{j+1,k}) - \frac{1}{2}(\partial f_{j-1,k} + \partial f_{j,k})\right) \\ &-\left(\frac{1}{2}(\partial g_{j,k} + \partial g_{j,k+1}) - \frac{1}{2}(\partial g_{j,k-1} + \partial g_{j,k})\right)\end{aligned}$$

$$\begin{aligned}\partial R^n_{j,k} = &-\frac{1}{2}\left[\left(\frac{\partial f}{\partial q}\delta q\right)_{j+1,k} + \left(\frac{\partial f}{\partial q}\delta q\right)_{j,k}\right] + \frac{1}{2}\left[\left(\frac{\partial f}{\partial q}\delta q\right)_{j,k} + \left(\frac{\partial f}{\partial q}\delta q\right)_{j-1,k}\right] \\ &-\frac{1}{2}\left[\left(\frac{\partial g}{\partial q}\delta q\right)_{j,k} + \left(\frac{\partial g}{\partial q}\delta q\right)_{j,k+1}\right] + \frac{1}{2}\left[\left(\frac{\partial g}{\partial q}\delta q\right)_{j,k-1} + \left(\frac{\partial g}{\partial q}\delta q\right)_{j,k}\right]\end{aligned}$$

Discrete Adjoint Formulation

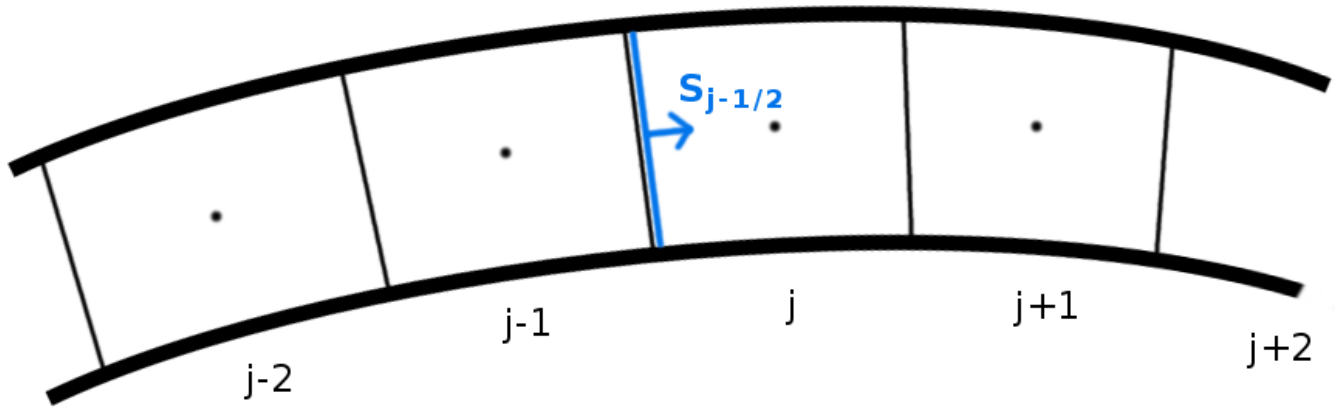
We can use the definition of the flux Jacobian A and B:

$$\left(\frac{\partial f}{\partial q}\delta q\right)_{j,k} = (A\delta q)_{j,k} \quad , \quad \left(\frac{\partial g}{\partial q}\delta q\right)_{j,k} = (B\delta q)_{j,k}$$

so that the equations simplify to

$$\begin{aligned}\partial R^n_{j,k} = &-\frac{1}{2}\left[(A\delta q)_{j+1,k} + (A\delta q)_{j,k}\right] + \frac{1}{2}\left[(A\delta q)_{j,k} + (A\delta q)_{j-1,k}\right] \\ &-\frac{1}{2}\left[(B\delta q)_{j,k} + (B\delta q)_{j,k+1}\right] + \frac{1}{2}\left[(B\delta q)_{j,k-1} + (B\delta q)_{j,k}\right]\end{aligned}$$

Mistake Made: Flux Jacobian



- q is the mass, momentum, and energy at the cell center
- f (and A)requires information from the cell face
- f usually uses interpolated q values to the face

Initially I used $A_{j-1/2}$ with interpolated q values however found this leads to **incorrect** terms.

Jacobian Discretization

$$A = \frac{\partial f}{\partial q} = \begin{bmatrix} 0 & S_x & S_y & 0 \\ S_x\phi - uV & V - a_3 S_x u & S_y u - a_2 S_x v & a_2 S_x \\ S_y\phi - vV & S_x v - a_2 S_y v & V - a_3 S_y v & a_2 S_y \\ V(\phi - a_1) & S_x a_1 - a_2 uV & S_y a_1 - a_2 vV & \gamma V \end{bmatrix}$$

where a_1, a_2, a_3 are constants and

$$V = S_x u + S_y v \quad , \quad \phi = \frac{1}{2}(\gamma - 1)(u^2 + v^2)$$

Since we have f at the cell interface $(j + \frac{1}{2})$ but q at the cell center, I defined u, v, E at interfaces using interpolation

$$u_{j+\frac{1}{2}} = \frac{1}{2}(u_j + u_{j+1}) \quad \leftarrow \quad \text{WRONG!}$$

Jacobian Discretization

Instead, use the cell-interface geometry in both the "left" and "right" Jacobian and average them over the face:

$$A_{j+\frac{1}{2}} = \left[\frac{1}{2}(A_j + A_{j+1})\right]_{S@j+\frac{1}{2}}$$

Note: In a continuous adjoint implementation, this would not matter since we can discretize the adjoint equation however we like. To best match our discretized Euler equations, however, we must follow this methodology.

By-Hand Code

```
#include "adjoint.hpp"
#include "tmpfluxb.h"

// #define DO_SIMPLE
#define EPS 0.25

void adj_flux(double* q_l,   double* q_r,
              double* psi_l, double* psi_r,
              double* f_l,   double* f_r,
              Dim *dim, double S[2], double A[4][4], int mini, int maxi){

  double Sx, Sy, phi, u, v, V, E;
  double a1;
  double a2 = GAMMA-1;
  double a3 = GAMMA-2;
  Sx        = S[0];
  Sy        = S[1];
  double dpsi0, dpsi1, dpsi2, dpsi3;
  dpsi0     = 0.5*(psi_l[0]*(1-mini) - psi_r[0]*(1-maxi));
  dpsi1     = 0.5*(psi_l[1]*(1-mini) - psi_r[1]*(1-maxi));
  dpsi2     = 0.5*(psi_l[2]*(1-mini) - psi_r[2]*(1-maxi));
  dpsi3     = 0.5*(psi_l[3]*(1-mini) - psi_r[3]*(1-maxi));

  // ============================================================
  // LEFT
  //
  u         = q_l[1]/q_l[0];
  v         = q_l[2]/q_l[0];
  E         = q_l[3]/q_l[0];

  V         = Sx*u + Sy*v;
  phi       = 0.5*a2*(u*u + v*v);
  a1        = GAMMA*E - phi;
  // build the Jacobian
  A[0][0]   = 0.0;
  A[0][1]   = Sx;
  A[0][2]   = Sy;
  A[0][3]   = 0.0;
  A[1][0]   = Sx*phi - u*V;
  A[1][1]   = V-a3*Sx*u;
  A[1][2]   = Sy*u-a2*Sx*v;
  A[1][3]   = a2*Sx;
  A[2][0]   = Sy*phi-v*V;
  A[2][1]   = Sx*v-a2*Sy*u;
  A[2][2]   = V-a3*Sy*v;
  A[2][3]   = a2*Sy;
  A[3][0]   = V*(phi-a1);
  A[3][1]   = Sx*a1-a2*u*V;
  A[3][2]   = Sy*a1-a2*v*V;
  A[3][3]   = GAMMA*V;

  // We want  [ A^T ]*[dpsi] - dissipation
  f_l[0]  = A[0][0]*dpsi0 + A[1][0]*dpsi1 + A[2][0]*dpsi2 + A[3][0]*dpsi3;
  f_l[1]  = A[0][1]*dpsi0 + A[1][1]*dpsi1 + A[2][1]*dpsi2 + A[3][1]*dpsi3;
  f_l[2]  = A[0][2]*dpsi0 + A[1][2]*dpsi1 + A[2][2]*dpsi2 + A[3][2]*dpsi3;
  f_l[3]  = A[0][3]*dpsi0 + A[1][3]*dpsi1 + A[2][3]*dpsi2 + A[3][3]*dpsi3;

  // ============================================================
  // RIGHT
  //
  u         = q_r[1]/q_r[0];
  v         = q_r[2]/q_r[0];
  E         = q_r[3]/q_r[0];

  V         = Sx*u + Sy*v;
  phi       = 0.5*a2*(u*u + v*v);
  a1        = GAMMA*E - phi;
  // build the Jacobian
  A[0][0]   = 0.0;
  A[0][1]   = Sx;
  A[0][2]   = Sy;
  A[0][3]   = 0.0;
  A[1][0]   = Sx*phi - u*V;
  A[1][1]   = V-a3*Sx*u;
  A[1][2]   = Sy*u-a2*Sx*v;
  A[1][3]   = a2*Sx;
  A[2][0]   = Sy*phi-v*V;
  A[2][1]   = Sx*v-a2*Sy*u;
  A[2][2]   = V-a3*Sy*v;
  A[2][3]   = a2*Sy;
  A[3][0]   = V*(phi-a1);
  A[3][1]   = Sx*a1-a2*u*V;
  A[3][2]   = Sy*a1-a2*v*V;
  A[3][3]   = GAMMA*V;

  f_r[0]  = A[0][0]*dpsi0 + A[1][0]*dpsi1 + A[2][0]*dpsi2 + A[3][0]*dpsi3;
  f_r[1]  = A[0][1]*dpsi0 + A[1][1]*dpsi1 + A[2][1]*dpsi2 + A[3][1]*dpsi3;
  f_r[2]  = A[0][2]*dpsi0 + A[1][2]*dpsi1 + A[2][2]*dpsi2 + A[3][2]*dpsi3;
  f_r[3]  = A[0][3]*dpsi0 + A[1][3]*dpsi1 + A[2][3]*dpsi2 + A[3][3]*dpsi3;

}
```

- 90 lines, with comments
- Actual code for Adjoint Flux
- Code executed at every point in the mesh
- Distinctly see two “sections”: one for each Jacobian Multiplication

Auto-Differentiated Code

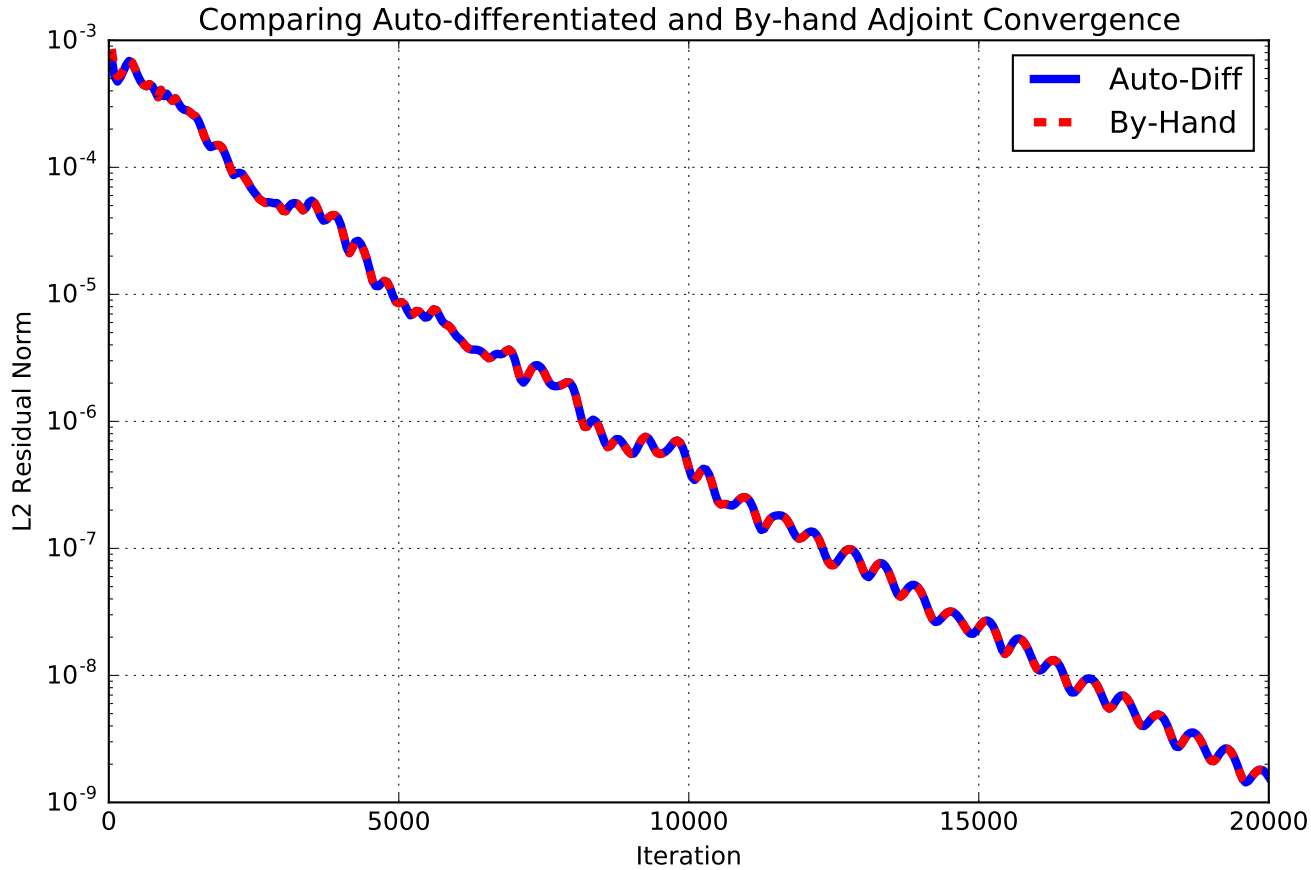
- 195 lines
- Same routine as previous slide but auto-differentiated
- This is from a “cleaned up” routine
  - Replace **if** statements with boolean algebra
  - Minimize square roots and other branching code
- No longer any coherent structure

Mistake Made: Boundary Conditions

- Typically boundary conditions are applied to "ghost" points across the boundaries of a mesh.
- In the discrete adjoint formulation, however, boundary conditions are instead computed above the wall at real field points.
- The adjoint variable ψ is then simply extrapolated to the ghost cells

Initially I tried to apply the adjoint boundary conditions to the ghost cells as is done in the Euler equations.

Results: Convergence



- We expect these to be exactly the same
- Euler Explicit time marching restricts timestep size to be very small

Analysis

Our original equation of interest was:

$$\delta I = \delta I_q + \delta I_X - \psi(\delta R_q + \delta R_X)$$

We solved the Adjoint relation

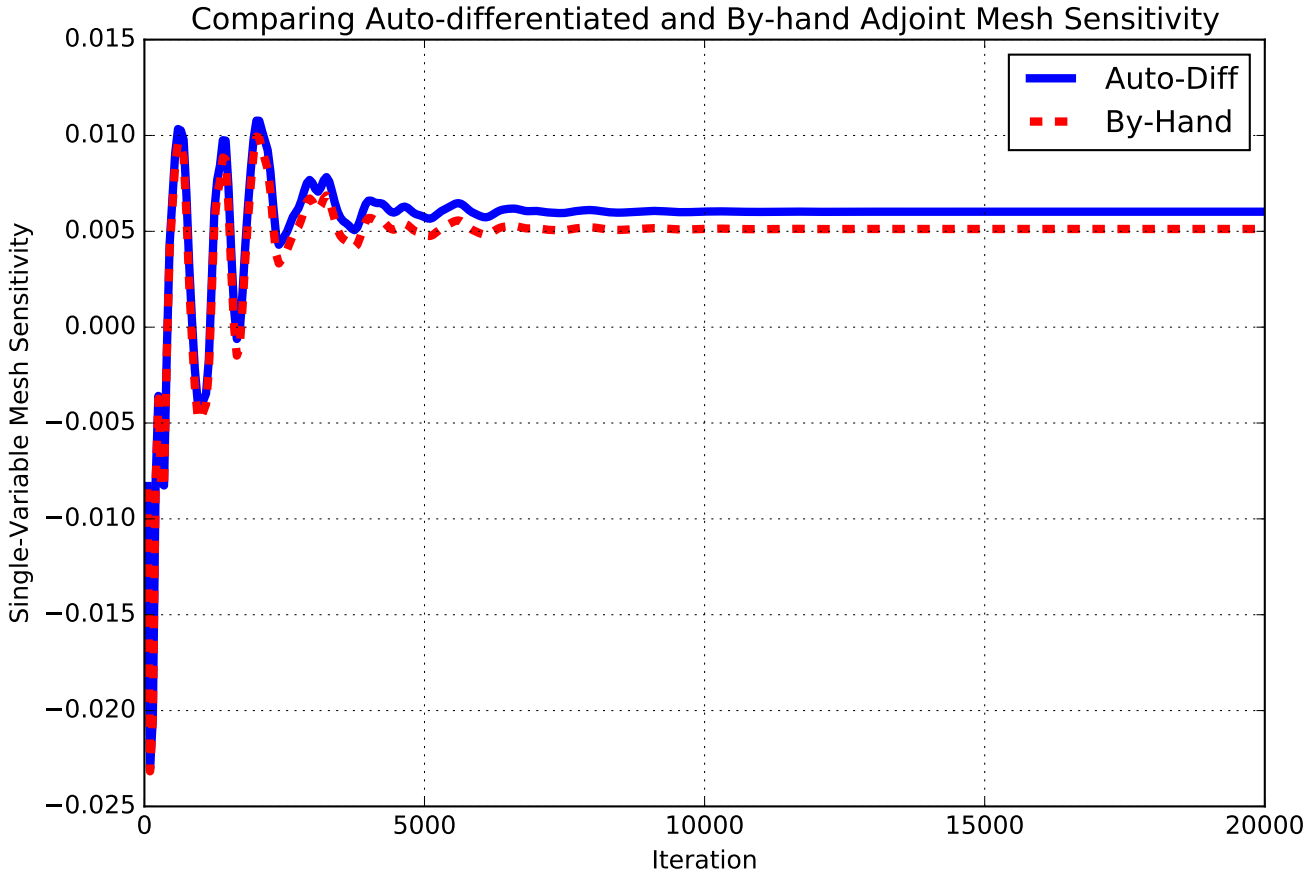
$$\delta I_q - \psi(\delta R_q) = 0$$

which leaves

$$\delta I = \delta I_X - \psi(\delta R_X)$$

There is one final differentiation of R but with respect to the grid coordinates.

Grid Coordinate Sensitivity



- We expect these to be exactly the same (they are not)
- I have been having issues with getting this differentiation to match exactly with the auto-differentiated result.

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.

Next Steps and Comments

- Fix errors in $X-$differentiation of the residual
- Compare design variable sensitivities to auto-differentiated and brute-force methods.
- Implement implicit routines to speed up convergence
  - Tens of thousands of iterations not ideal (even if ~ 1 min runtime)
  - Implicit routines do not have to exactly match auto-differentiated version, since they only affect convergence rate and not accuracy.
- Though auto-diff and by-hand results are the same mathematically, the by-hand approach allows for further potential speedup through parallelization (OpenMP, MPI, CUDA).

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.

[Nadarajah and Jameson(2000)] Siva Nadarajah and Antony Jameson.
A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization.
Aerospace Sciences Meetings, (c):—-, 2000.
doi: 10.2514/6.2000-667.
URL http://dx.doi.org/10.2514/6.2000-667.

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