Adjoint code development and optimization using automatic differentiation (AD)

Praveen. C

Computational and Theoretical Fluid Dynamics Division
National Aerospace Laboratories
Bangalore - 560 037

CTFD Seminar, March 31, 2006

Outline

Outline

Mathematical formulation

- Constrained minimization problem

$$\min_{\alpha} J(\alpha, u) \quad \text{subject to} \quad R(\alpha, u) = 0$$

- Find $\delta\alpha$ such that $\delta J < 0$

$$\begin{aligned} \delta J &= \frac{\partial J}{\partial \alpha}\delta\alpha + \frac{\partial J}{\partial u}\delta u \\ &= \frac{\partial J}{\partial \alpha}\delta\alpha + \frac{\partial J}{\partial u}\frac{\partial u}{\partial \alpha}\delta\alpha \\ &= G\delta\alpha \end{aligned}$$

- Steepest descent

$$\delta\alpha = -\epsilon G^{\top}$$

$$\delta J = -\epsilon G G^{\top} = -\epsilon \|G\|^2 < 0$$

Mathematical formulation

- Constrained minimization problem

$$\min_{\alpha} J(\alpha, u) \quad \text{subject to} \quad R(\alpha, u) = 0$$

- Find $\delta\alpha$ such that $\delta J < 0$

$$\begin{aligned} \delta J &= \frac{\partial J}{\partial \alpha}\delta\alpha + \frac{\partial J}{\partial u}\delta u \\ &= \frac{\partial J}{\partial \alpha}\delta\alpha + \frac{\partial J}{\partial u}\frac{\partial u}{\partial \alpha}\delta\alpha \\ &= G\delta\alpha \end{aligned}$$

- Steepest descent

$$\delta\alpha = -\epsilon G^{\top}$$

$$\delta J = -\epsilon G G^{\top} = -\epsilon \|G\|^2 < 0$$

Sensitivity approach

- Linearized state equation

$$\frac{\partial R}{\partial \alpha}\delta\alpha + \frac{\partial R}{\partial u}\delta u = 0$$

or

$$\boxed{\frac{\partial R}{\partial u}\frac{\partial u}{\partial \alpha} = -\frac{\partial R}{\partial \alpha}}$$

- Solve sensitivity equation iteratively

$$\frac{\partial}{\partial t}\frac{\partial u}{\partial \alpha} + \frac{\partial R}{\partial u}\frac{\partial u}{\partial \alpha} = -\frac{\partial R}{\partial \alpha}$$

- Same eigenvalues as original system
- Costly if number of α are large

Adjoint approach

- We have

$$\delta J = \frac{\partial J}{\partial \alpha}\delta\alpha + \frac{\partial J}{\partial u}\delta u \quad \text{and} \quad \frac{\partial R}{\partial \alpha}\delta\alpha + \frac{\partial R}{\partial u}\delta u = 0$$

- Introduce a new unknown v

$$\begin{aligned}\delta J &= \frac{\partial J}{\partial \alpha}\delta\alpha + \frac{\partial J}{\partial u}\delta u + v^\top\left(\frac{\partial R}{\partial \alpha}\delta\alpha + \frac{\partial R}{\partial u}\delta u\right) \\ &= \left(\frac{\partial J}{\partial \alpha} + v^\top\frac{\partial R}{\partial \alpha}\right)\delta\alpha + \left(\frac{\partial J}{\partial u} + v^\top\frac{\partial R}{\partial u}\right)\delta u\end{aligned}$$

- Adjoint equation

$$\boxed{\left(\frac{\partial R}{\partial u}\right)^\top v = -\left(\frac{\partial J}{\partial u}\right)^\top}$$

- Iterative solution

$$\frac{\partial v}{\partial t} + \left(\frac{\partial R}{\partial u}\right)^\top v = -\left(\frac{\partial J}{\partial u}\right)^\top$$

Continuous vs Discrete

- Continuous approach:
  - Start with governing PDE
  - Derive adjoint PDE and boundary conditions
  - Discretize adjoint PDE and solve
  - Must be re-derived whenever cost function changes
  - Not all cost functions admissible
- Discrete approach:
  - Start with discrete approximation like FVM
  - Derive discrete adjoint equations
  - Solve discrete adjoint equations
  - True sensitivity of discrete solution

Outline

Techniques for computing gradients

- Hand differentiation
- Finite difference method
- Complex variable method
- Automatic differentiation
  - Computer code to compute some function
  - Chain rule of differentiation
  - Generates a code to compute derivatives
  - ADIFOR, ADOLC, ODYSEE, TAMC, TAF, TAPENADE

Differentiation: Example

- A simple example

$$f = (xy + \sin x + 4)(3y^2 + 6)$$

- Computer code, $f = t_{10}$

$$\begin{aligned}
t_1 &= x \\
t_2 &= y \\
t_3 &= t_1 t_2 \\
t_4 &= \sin t_1 \\
t_5 &= t_3 + t_4 \\
t_6 &= t_5 + 4 \\
t_7 &= t_2^2 \\
t_8 &= 3t_7 \\
t_9 &= t_8 + 6 \\
t_{10} &= t_6 t_9
\end{aligned}$$

Automatic Differentiation: F77 code

```
      subroutine costfunc(x, y, f)
      t1 = x
      t2 = y
      t3 = t1*t2
      t4 = sin(t1)
      t5 = t3 + t4
      t6 = t5 + 4
      t7 = t2**2
      t8 = 3.0*t7
      t9 = t8 + 6.0
      t10 = t6*t9
      f  = t10
      end
```

Differentiation: Direct mode

- Some definitions

$$I_k := \{i : i < k \text{ and } t_k \text{ depends explicitly on } t_i\}$$

$$\dot{t}_i := \frac{\partial t_i}{\partial \alpha}, \quad t_{i,k} := \frac{\partial t_i}{\partial t_k}$$

- Apply chain rule of differentiation

$$\dot{t}_k = \frac{\partial t_k}{\partial \alpha} = \sum_{i \in I_k} \frac{\partial t_i}{\partial \alpha} \frac{\partial t_k}{\partial t_i} = \sum_{i \in I_k} \dot{t}_i t_{k,i} \quad k = 1, 2, \ldots, 9, 10$$

- $\alpha = x$, or y

Differentiation: Direct mode

- Apply chain rule of differentiation

$$
\begin{aligned}
t_1 &= x \\
t_2 &= y \\
t_3 &= t_1 t_2 \\
t_4 &= \sin(t_1) \\
t_5 &= t_3 + t_4 \\
t_6 &= t_5 + 4 \\
t_7 &= t_2^2 \\
t_8 &= 3t_7 \\
t_9 &= t_8 + 6 \\
t_{10} &= t_6 t_9
\end{aligned}
\qquad
\begin{aligned}
\dot{t}_1 &= \dot{x} \\
\dot{t}_2 &= \dot{y} \\
\dot{t}_3 &= \dot{t}_1 t_2 + t_1 \dot{t}_2 \\
\dot{t}_4 &= \cos(t_1)\dot{t}_1 \\
\dot{t}_5 &= \dot{t}_3 + \dot{t}_4 \\
\dot{t}_6 &= \dot{t}_5 \\
\dot{t}_7 &= 2t_2 \dot{t}_2 \\
\dot{t}_8 &= 3\dot{t}_7 \\
\dot{t}_9 &= \dot{t}_8 \\
\dot{t}_{10} &= \dot{t}_6 t_9 + t_6 \dot{t}_9
\end{aligned}
$$

- $\dot{x} = 1, \dot{y} = 0, \dot{t}_{10} = \frac{\partial f}{\partial x}$ and $\dot{x} = 0, \dot{y} = 1, \dot{t}_{10} = \frac{\partial f}{\partial y}$
- `tapenade -d -vars "x y" -outvars f costfunc.f`

Automatic Differentiation: Direct mode

```
SUBROUTINE COSTFUNC_D(x, xd, y, yd, f, fd)
t1d = xd
t1 = x
t2d = yd
t2 = y
t3d = t1d*t2 + t1*t2d
t3 = t1*t2
t4d = t1d*COS(t1)
t4 = SIN(t1)
t5d = t3d + t4d
t5 = t3 + t4
t6d = t5d
t6 = t5 + 4
t7d = 2*t2*t2d
t7 = t2**2
t8d = 3.0*t7d
t8 = 3.0*t7
t9d = t8d
t9 = t8 + 6.0
t10d = t6d*t9 + t6*t9d
t10 = t6*t9
fd = t10d
f = t10
END
```

Differentiation: Reverse mode

- Some definitions

$$J_k := \{i : i > k \text{ and } t_i \text{ depends explicitly on } t_k\}$$

$$\bar{t}_i := \frac{\partial f}{\partial t_i} = \frac{\partial t_{10}}{\partial t_i}, \quad t_{i,k} := \frac{\partial t_i}{\partial t_k}$$

- Apply chain rule of differentiation in reverse

$$\bar{t}_k = \frac{\partial t_{10}}{\partial t_k} = \sum_{i \in J_k} \frac{\partial t_{10}}{\partial t_i} \frac{\partial t_i}{\partial t_k} = \sum_{i \in J_k} \bar{t}_i t_{i,k} \quad k = 10, 9, \ldots, 2, 1$$

Differentiation: Reverse mode

- Apply chain rule of differentiation in reverse

$$
\begin{aligned}
t_1 &= x \\
t_2 &= y \\
t_3 &= t_1 t_2 \\
t_4 &= \sin(t_1) \\
t_5 &= t_3 + t_4 \\
t_6 &= t_5 + 4 \\
t_7 &= t_2^2 \\
t_8 &= 3t_7 \\
t_9 &= t_8 + 6 \\
t_{10} &= t_6 t_9
\end{aligned}
\qquad
\begin{aligned}
\bar{t}_{10} &= 1 \\
\bar{t}_9 &= \bar{t}_{10} t_{10,9} &&= t_6 \\
\bar{t}_8 &= \bar{t}_9 t_{9,8} &&= t_6 \\
\bar{t}_7 &= \bar{t}_8 t_{8,7} &&= 3t_6 \\
\bar{t}_6 &= \bar{t}_{10} t_{10,6} &&= t_9 \\
\bar{t}_5 &= \bar{t}_6 t_{6,5} &&= t_9 \\
\bar{t}_4 &= \bar{t}_5 t_{5,4} &&= t_9 \\
\bar{t}_3 &= \bar{t}_5 t_{5,3} &&= t_9 \\
\bar{t}_2 &= \bar{t}_7 t_{7,2} + \bar{t}_3 t_{3,2} &&= 6t_2 t_6 + t_1 t_9 \\
\bar{t}_1 &= \bar{t}_4 t_{4,1} + \bar{t}_3 t_{3,1} &&= t_9 \cos(t_1) + t_9 t_2
\end{aligned}
$$

- $\bar{t}_1 = \frac{\partial f}{\partial x}, \bar{t}_2 = \frac{\partial f}{\partial y}$
- `tapenade -b -vars "x y" -outvars f costfunc.f`

Automatic Differentiation: Reverse mode

```
SUBROUTINE COSTFUNC_B(x, xb, y, yb, f, fb)
t1 = x
t2 = y
t3 = t1*t2
t4 = SIN(t1)
t5 = t3 + t4
t6 = t5 + 4
t7 = t2**2
t8 = 3.0*t7
t9 = t8 + 6.0
t10b = fb
t6b = t9*t10b
t9b = t6*t10b
t8b = t9b
t7b = 3.0*t8b
t5b = t6b
t3b = t5b
t2b = t1*t3b + 2*t2*t7b
t4b = t5b
t1b = t2*t3b + COS(t1)*t4b
yb = t2b
xb = t1b
fb = 0.0
END
```

Direct versus reverse AD

- Direct mode: cost is proportional to number of independent variables
- Reverse mode: cost does not depend on the number of independent variables
- Reverse mode preferred when number of independent variables is large
- Scalar output f
  - Direct mode gives $\nabla f \cdot s$ for given vector s
  - Reverse mode gives ∇f
- Vector output $R \in \mathbb{R}^n$
  - Direct mode gives $\nabla R \cdot s$ for given vector s
  - Reverse mode gives $(\nabla R)^\top \cdot s$

Issues in reverse AD

- Intermediate variables required in reverse order
- Some variables may be over-written
- Iterative solvers
  - Only fi nal solution required
  - AD differentiates the iterative loop
  - Intermediate solutions stored in stack
  - Huge memory requirements
  - Not practical for large problems
- Piecemeal differentiation approach (Courty et al., Giles et al.):
  - Modular flow solver
  - Adjoint solver written manually
  - AD for differentiating the modules

Outline

Discrete 1-D adjoint equations

- Finite volume residual for i'th cell

$$R_i := F_{i+1/2} - F_{i-1/2} = 0$$

- Numerical flux function

$$F = F(X, Y), \quad F_{i+1/2} = F(U_i, U_{i+1})$$

- Perturbation equation

$$\begin{aligned} \delta R_i = & \quad \frac{\partial}{\partial X} F_{i+1/2} \delta U_i + \frac{\partial}{\partial Y} F_{i+1/2} \delta U_{i+1} \\ & - \frac{\partial}{\partial X} F_{i-1/2} \delta U_{i-1} - \frac{\partial}{\partial Y} F_{i-1/2} \delta U_i + \frac{\partial R_i}{\partial \alpha} \delta \alpha \quad = 0 \end{aligned}$$

- Introduce adjoint variable V_i for i'th cell

$$\delta J = \frac{\partial J}{\partial \alpha} \delta \alpha + \sum_i \frac{\partial J}{\partial U_i} \delta U_i + \sum_i V_i^\top \delta R_i$$

Discrete adjoint equations

- Collecting terms containing δU_i

$$\delta J = \sum_i \left[\frac{\partial J}{\partial U_i} + V_{i-1}^\top \frac{\partial}{\partial Y} F_{i-1/2} + V_i^\top \left(\frac{\partial}{\partial X} F_{i+1/2} - \frac{\partial}{\partial Y} F_{i-1/2} \right) - V_{i+1}^\top \frac{\partial}{\partial X} F_{i+1/2} \right] \delta U_i + \ldots$$

- Adjoint equation for i'th cell

$$\left(\frac{\partial J}{\partial U_i} \right)^\top + \left(\frac{\partial}{\partial Y} F_{i-1/2} \right)^\top V_{i-1} + \left(\frac{\partial}{\partial X} F_{i+1/2} - \frac{\partial}{\partial Y} F_{i-1/2} \right)^\top V_i - \left(\frac{\partial}{\partial X} F_{i+1/2} \right)^\top V_{i+1} = 0$$

Example flow solver

```
While u is not converged
  res = 0.0
  fluxinflow(u(1), res(1))
  do i=1,N-1
    fluxinterior(u(i), u(i+1), res(i), res(i+1))
  enddo
  fluxoutflow(u(N), res(N))
  do i=1,N
    u(i) = u(i) - (dt/dx)*res(i)
  enddo
endwhile

cost=0.0
do i=1,N
  costfunc(u(i), cost)
enddo
```

Example adjoint solver

```
costb=1.0
do i=1,N
  costfunc_b(u(i), ub1(i), cost, costb)
enddo

v=0.0
While v is not converged
  ub2 = 0.0
  fluxinflow_b(u(1), ub2(1), res(1), v(1))
  do i=1,N-1
    fluxinterior_b(u(i), ub2(i), u(i+1), ub2(i+1),
                   res(i), v(i), res(i+1), v(i+1))
  enddo
  fluxoutflow_b(u(N), ub2(N), res(N), v(N))
  do i=1,N
    v(i) = v(i) – (dt/dx)*(ub1(i) + ub2(i))
  enddo
endwhile
```

Quasi 1-D fbw

- Quasi 1-D flow in a duct

$$\frac{\partial}{\partial t}(hU) + \frac{\partial}{\partial x}(hf) = \frac{\mathrm{d}h}{\mathrm{d}x}P, \quad x \in (a,b) \quad t > 0$$

$$U = \begin{bmatrix} \rho \\ \rho u \\ E \end{bmatrix}, \quad f = \begin{bmatrix} \rho u \\ p + \rho u^2 \\ (E+p)u \end{bmatrix}, \quad P = \begin{bmatrix} 0 \\ p \\ 0 \end{bmatrix}$$

$h(x) =$ cross-section height of duct

- Inverse design: find shape h to get pressure distribution p^*
- Optimization problem: find the shape h which minimizes

$$J = \int_a^b (p - p^*)^2 \mathrm{d}x$$

Quasi 1-D fbw

- Finite volume scheme

$$h_i \frac{\mathrm{d}U_i}{\mathrm{d}t} + \frac{h_{i+1/2}F_{i+1/2} - h_{i-1/2}F_{i-1/2}}{\Delta x} = \frac{(h_{i+1/2} - h_{i-1/2})}{\Delta x} P_i$$

- Discrete cost function

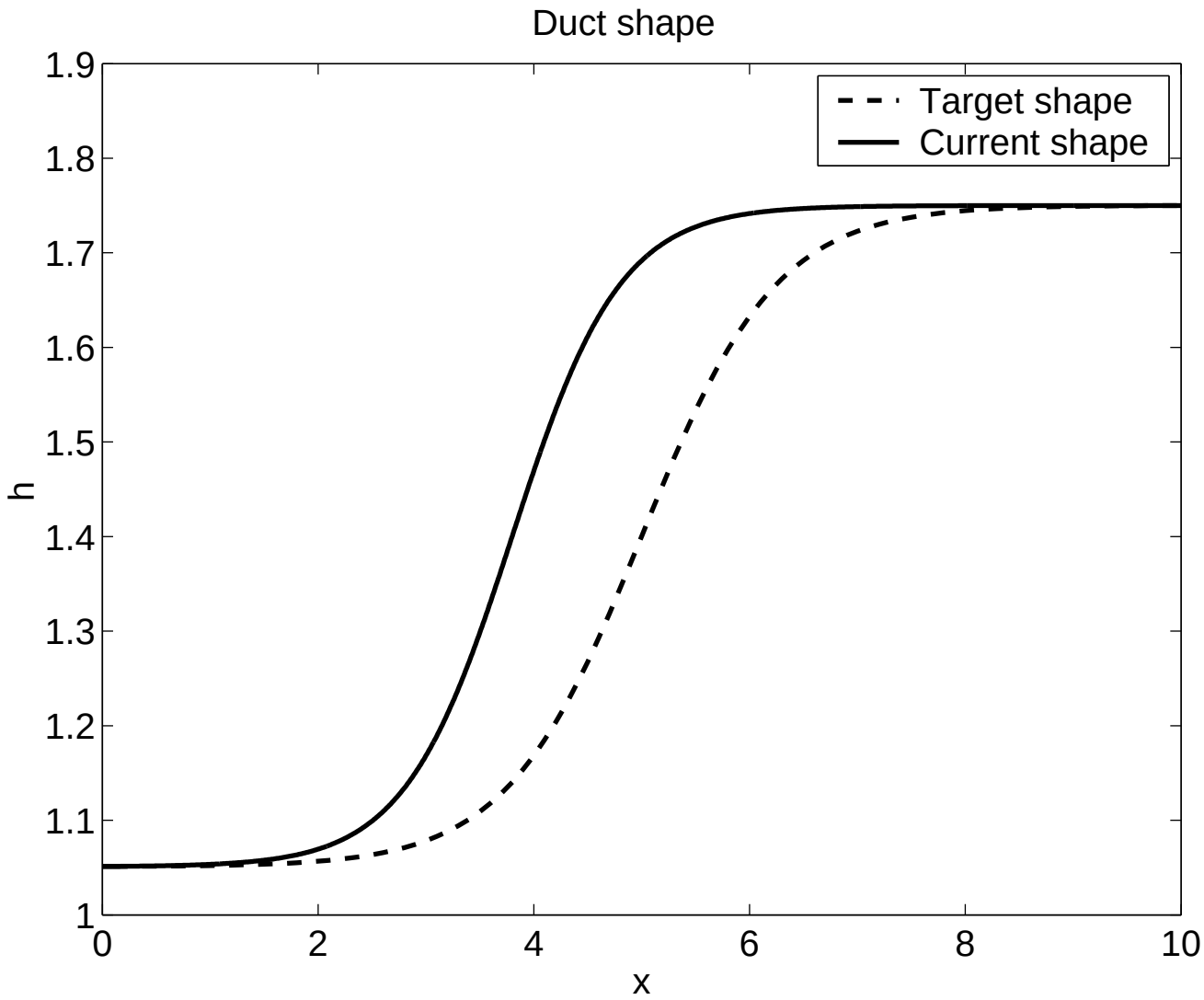
$$J = \sum_{i=1}^{N} (p_i - p_i^*)^2$$

- Control variables

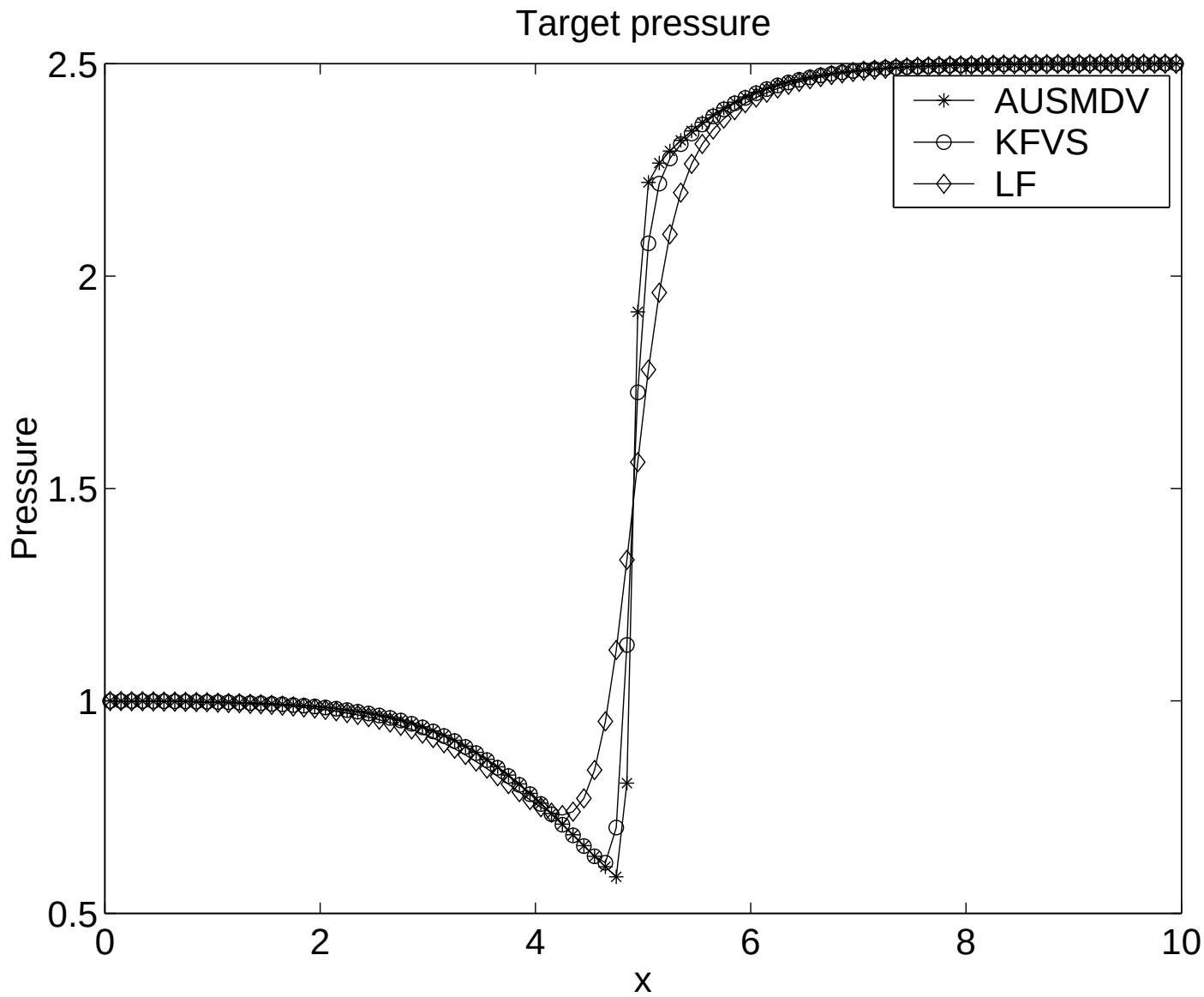
$$h_{1/2}, h_{1+1/2}, \ldots, h_{i+1/2}, \ldots, h_{N+1/2}$$

- $N = 100$

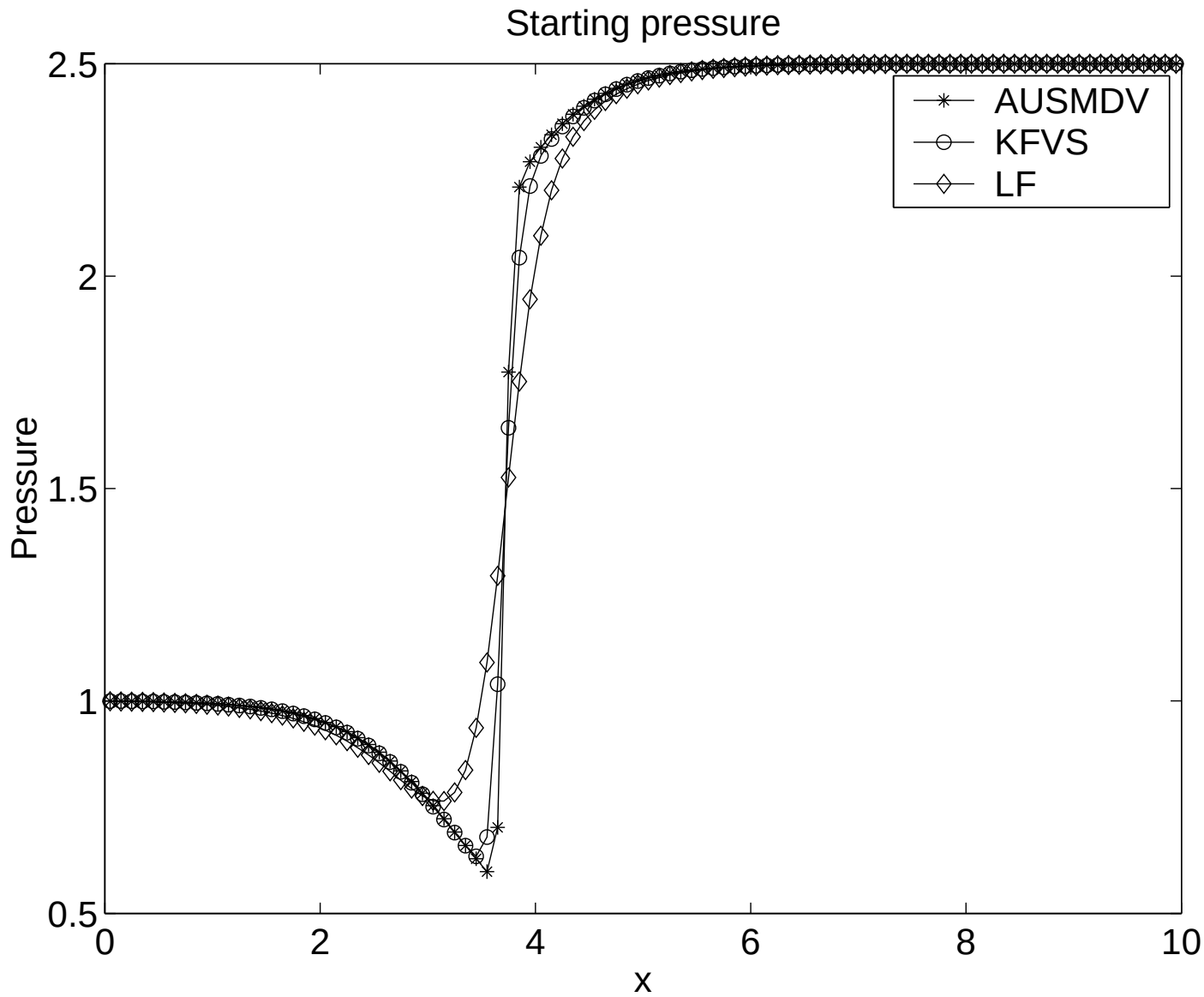
Duct shape


Duct shape
Target shape
Current shape
h
x
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0
2
4
6
8
10

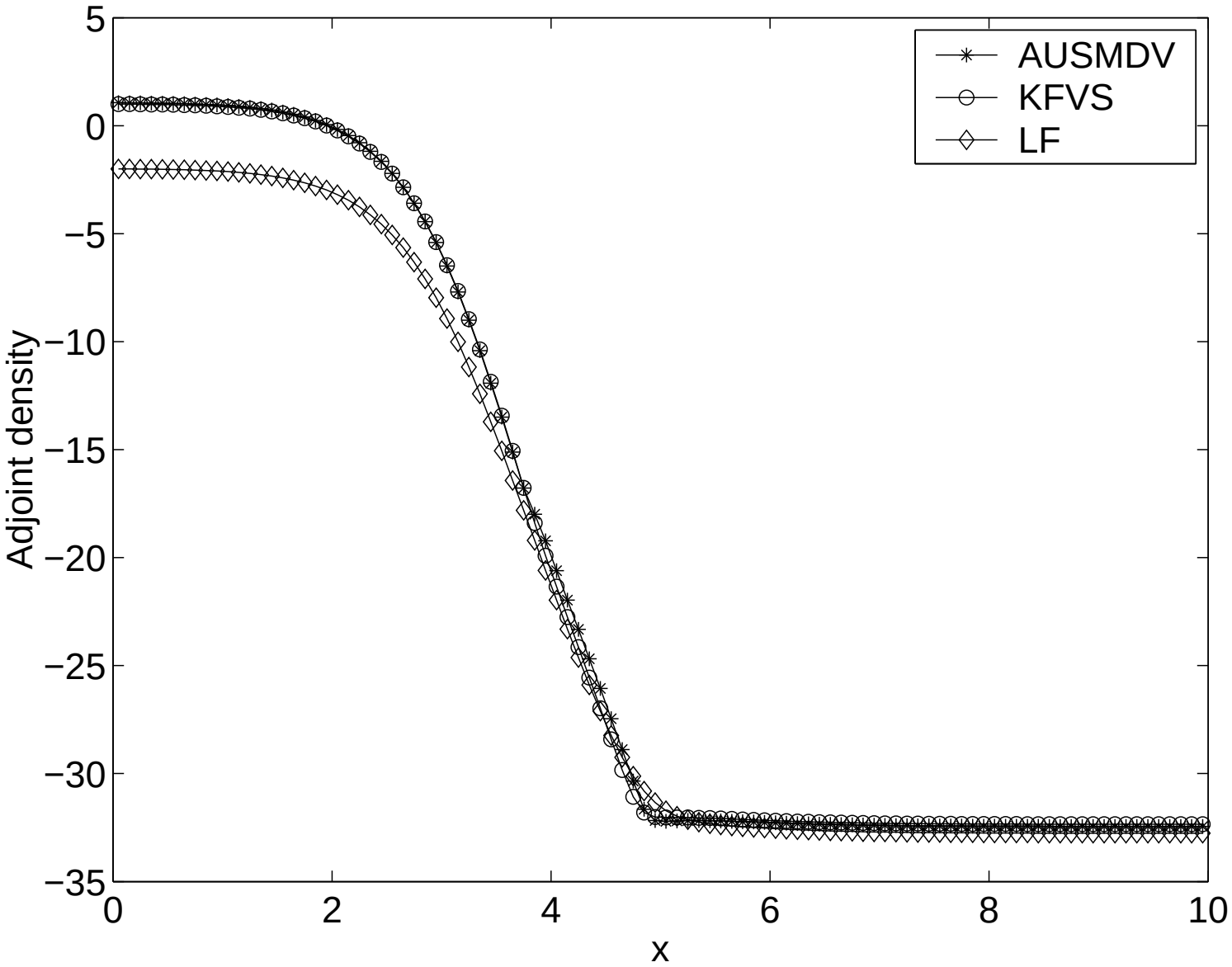
Target pressure distribution p^*



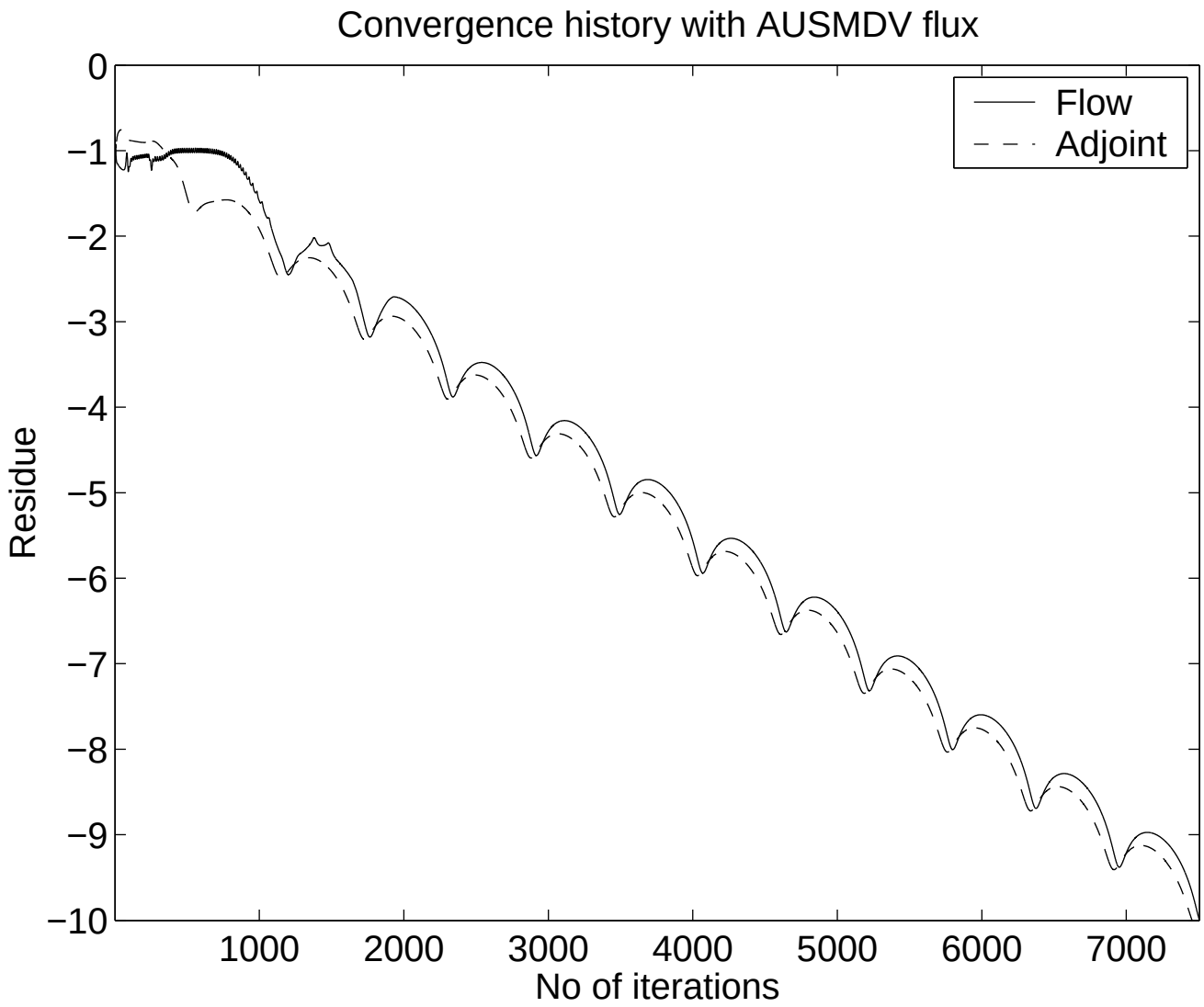
Current pressure distribution



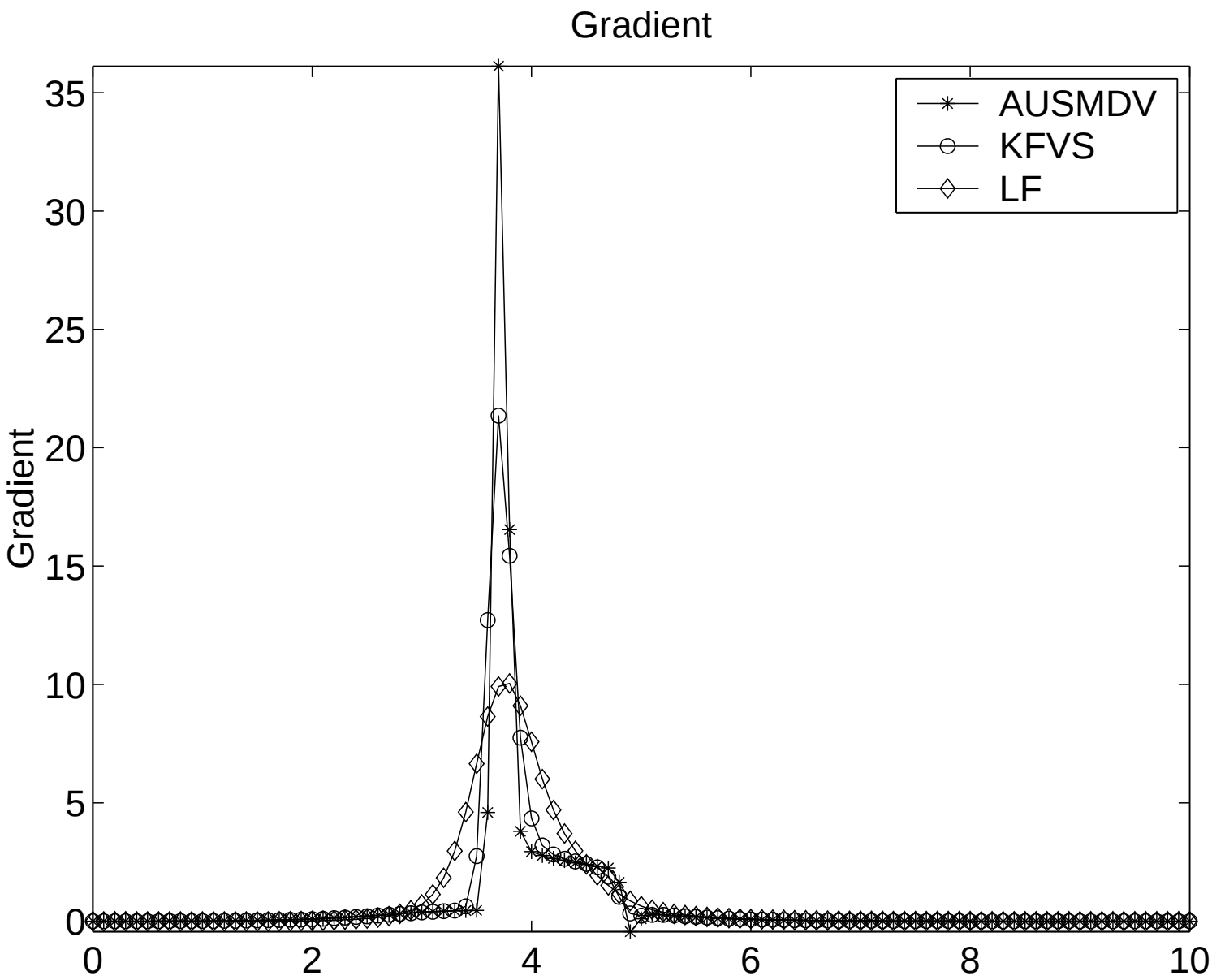
Adjoint density



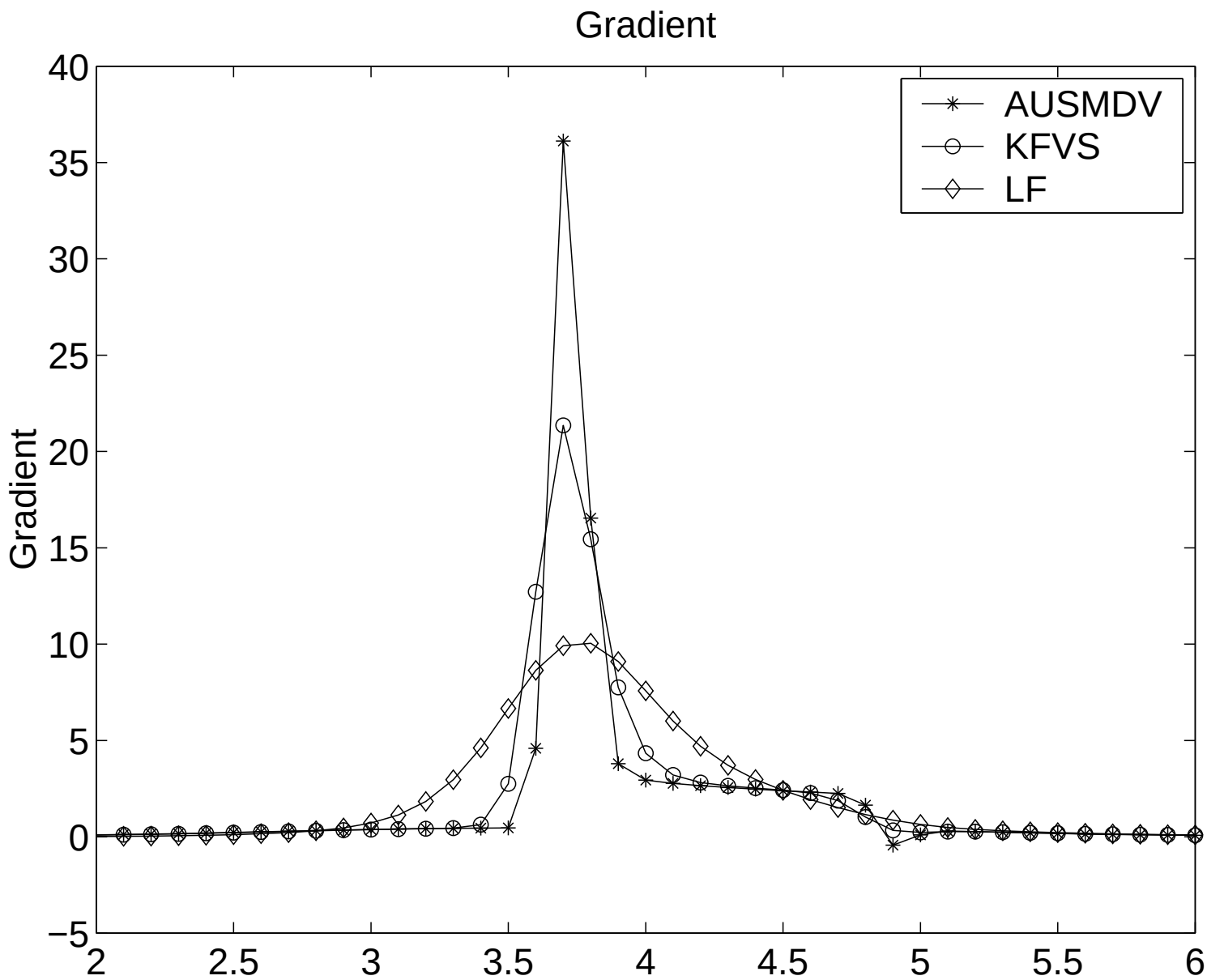
Convergence history: Explicit Euler



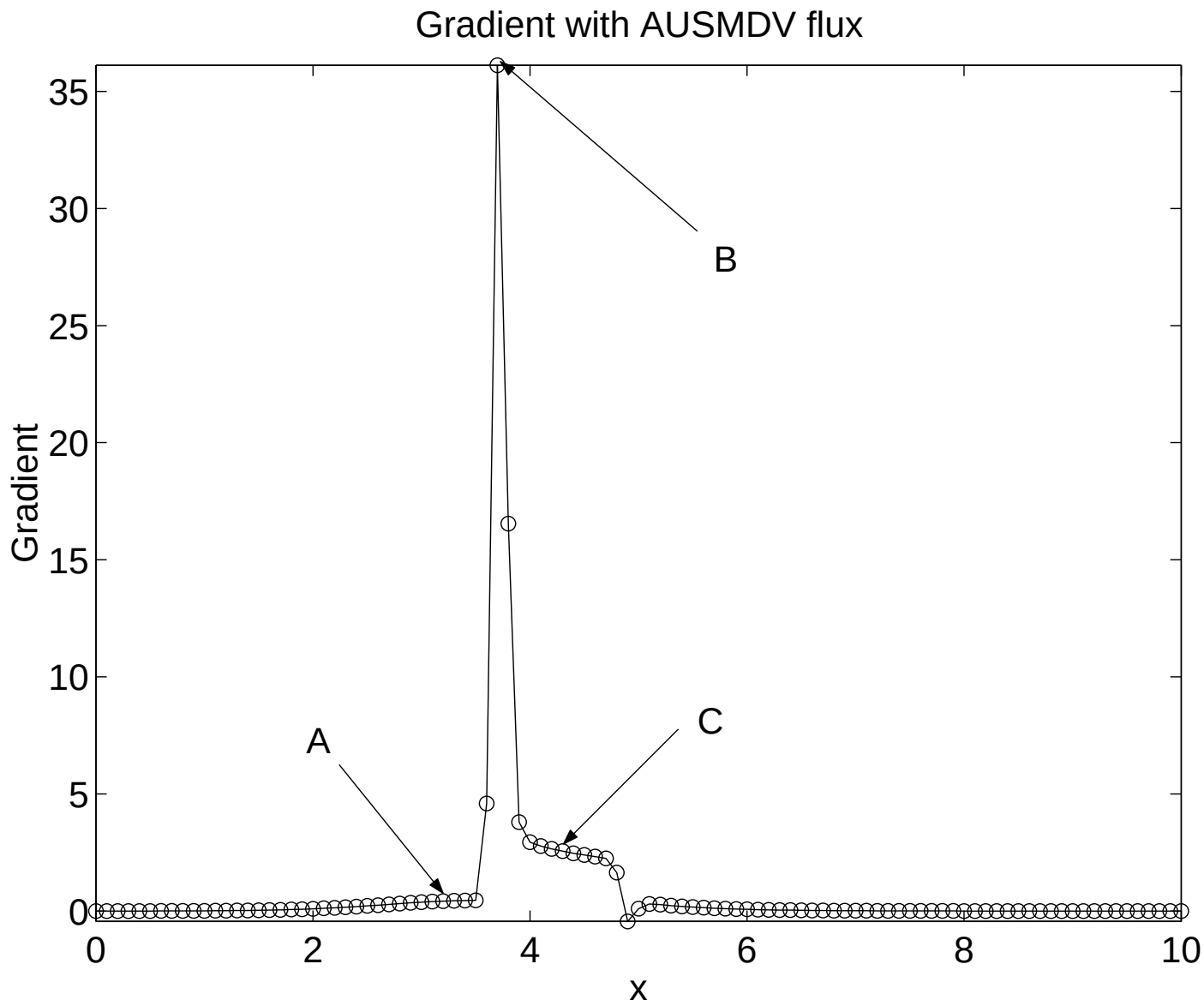
Shape gradient



Shape gradient



Validation of Shape gradient



Validation of shape gradient

$$\frac{\partial J}{\partial h} \approx \frac{J(h + \Delta h) - J(h - \Delta h)}{2\Delta h}$$

Δh	A	B	C
0.01	0.4191069499	35.18452823	2.545316345
0.001	0.4231223499	36.10982621	2.556461900
0.0001	0.4231624999	36.11933154	2.556573499
0.00001	0.4231599998	36.11942125	2.556575000
0.000001	0.4231999994	36.11942305	2.556550001
0.0000001	0.4229999817	36.11942329	2.556499971
AD	0.4231628330	36.11941951	2.556574450

Outline

Gradient smoothing

- Non-smooth gradients especially in the presence of shocks
- Smooth using an elliptic equation

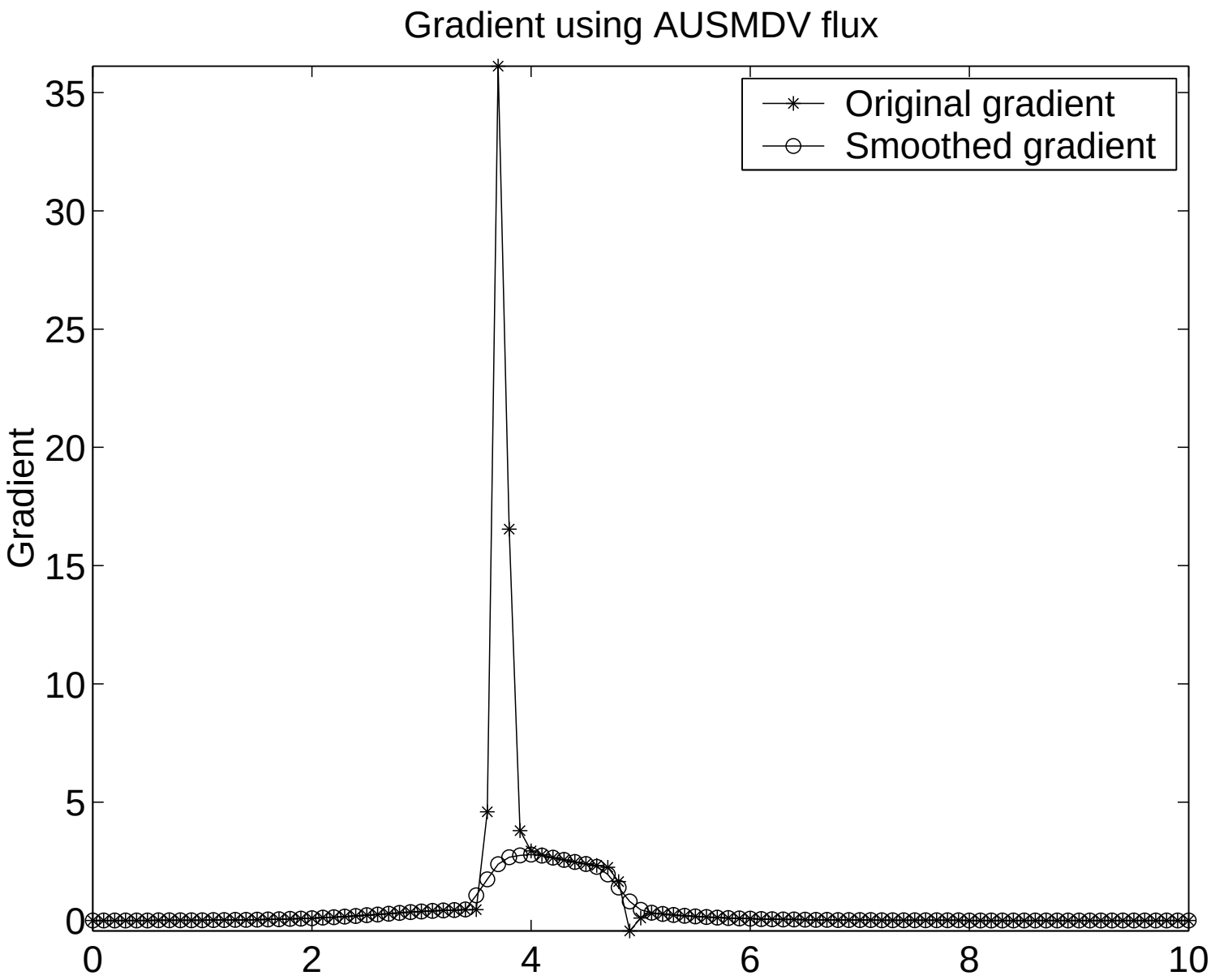
$$\left(1 - \epsilon_i \frac{\mathrm{d}^2}{\mathrm{d}x^2}\right) \bar{g}_i = g_i$$

$$\epsilon_i = \{|g_{i+1} - g_i| + |g_i - g_{i-1}|\} L_i$$

$$L_i = \frac{|g_{i+1} - 2g_i + g_{i-1}|}{\max(|g_{i+1} - g_i| + |g_i - g_{i-1}|, \mathrm{TOL})}$$

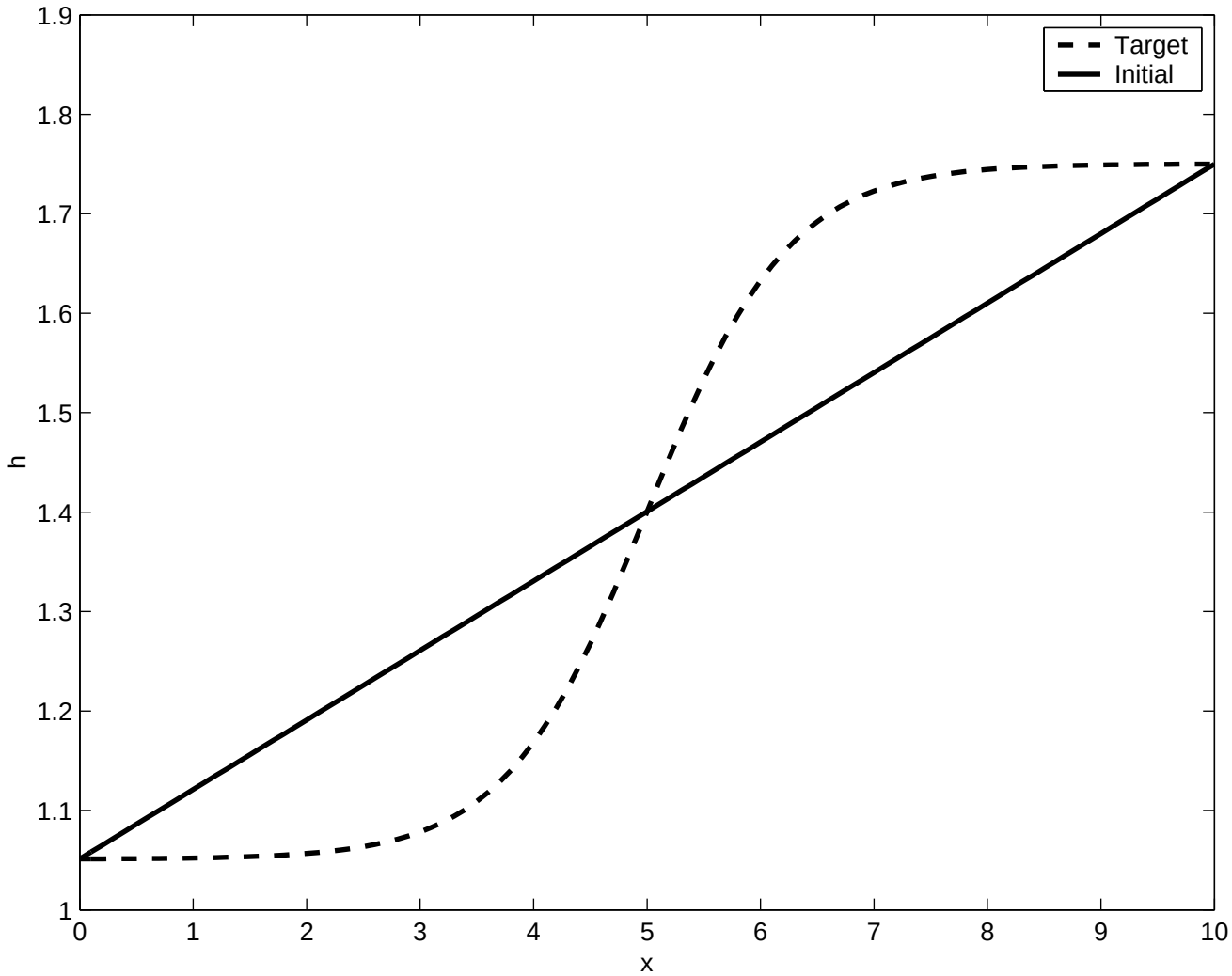
- Finite difference with Jacobi iterations

Gradient smoothing

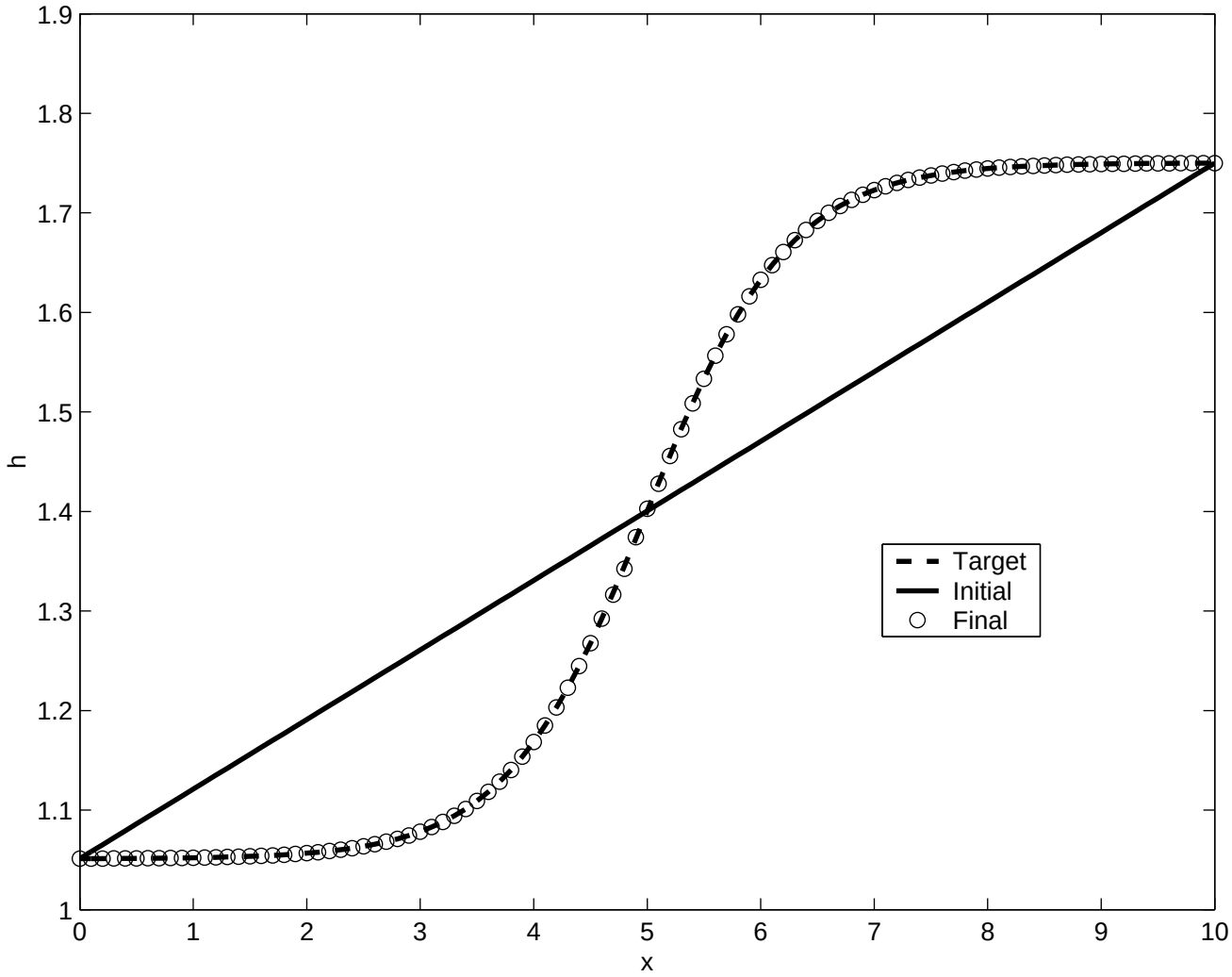


Outline

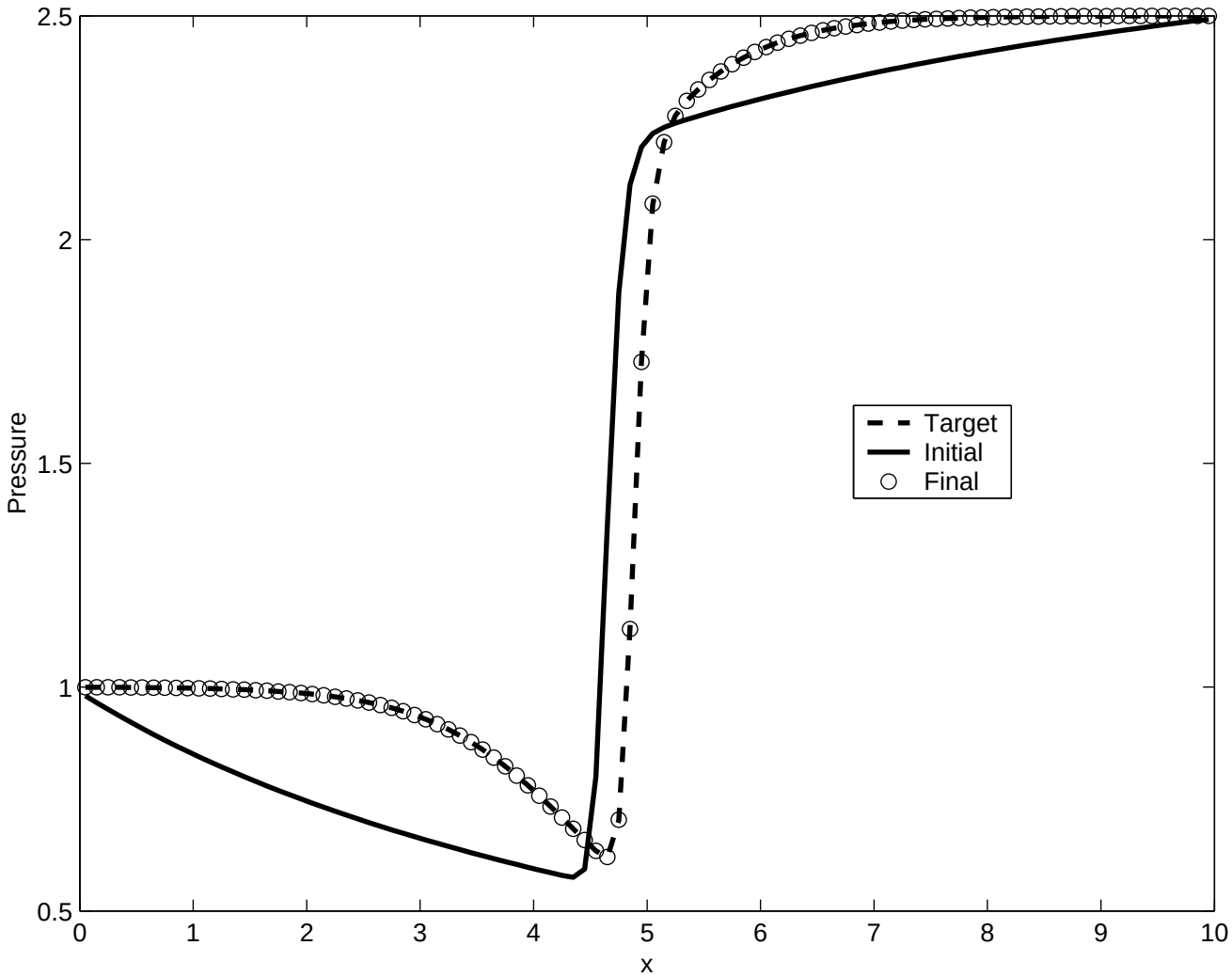
Quasi 1-D optimization: Shape



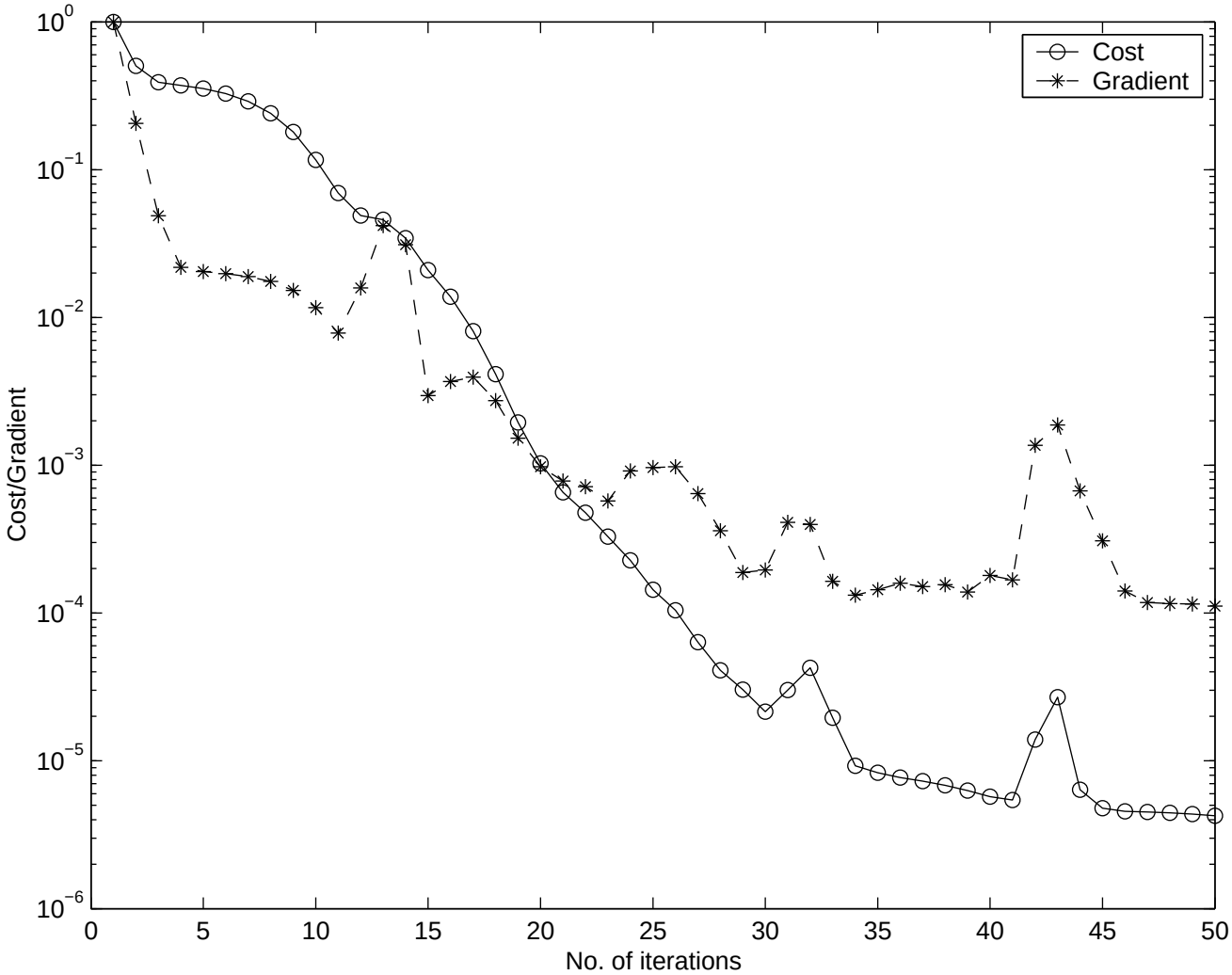
Quasi 1-D optimization: Final shape



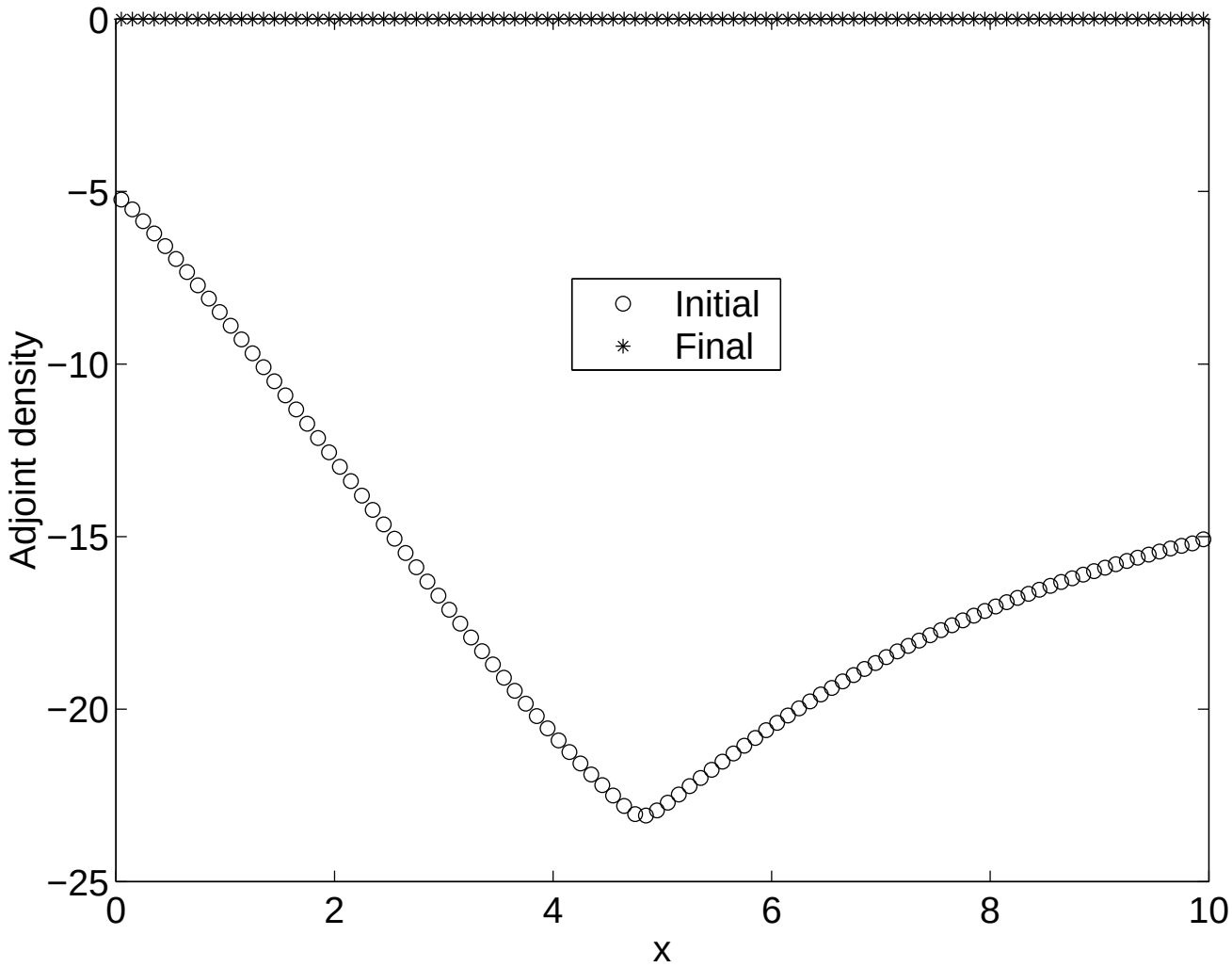
Quasi 1-D optimization: Pressure



Quasi 1-D optimization: Convergence



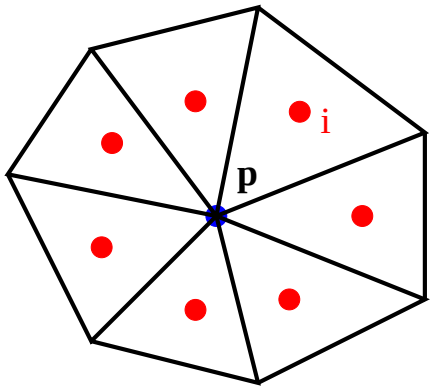
Quasi 1-D optimization: Adjoint density

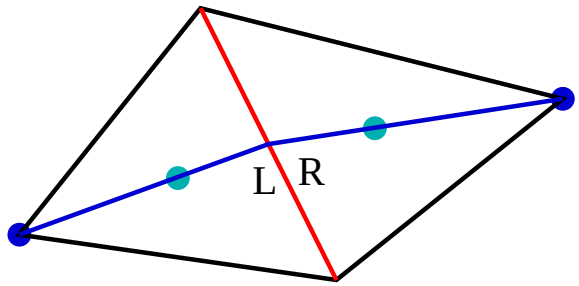


Outline

2-D fi nite volume fbw solver

- Unstructured triangular grid
- Vertex-centroid scheme (Frink, Jameson)
- Vertex values obtained using inverse-distance averaging with correction for linear consistency





$$u_p = \frac{\sum_i \frac{w_i}{r_i} u_i}{\sum_i \frac{w_i}{r_i}}$$

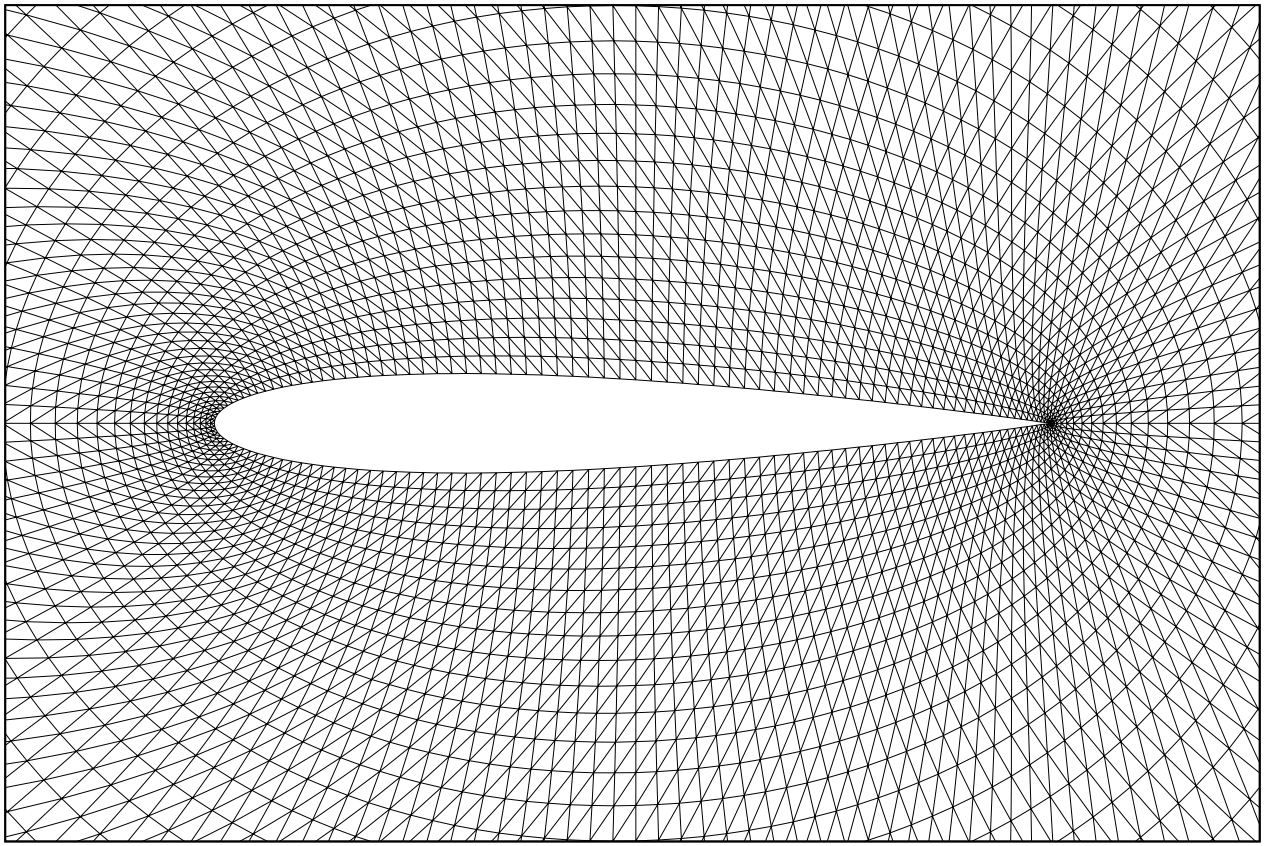
- Convergence acceleration using matrix-free LUSGS with spectral radius approximation

2-D fi nite volume adjoint solver

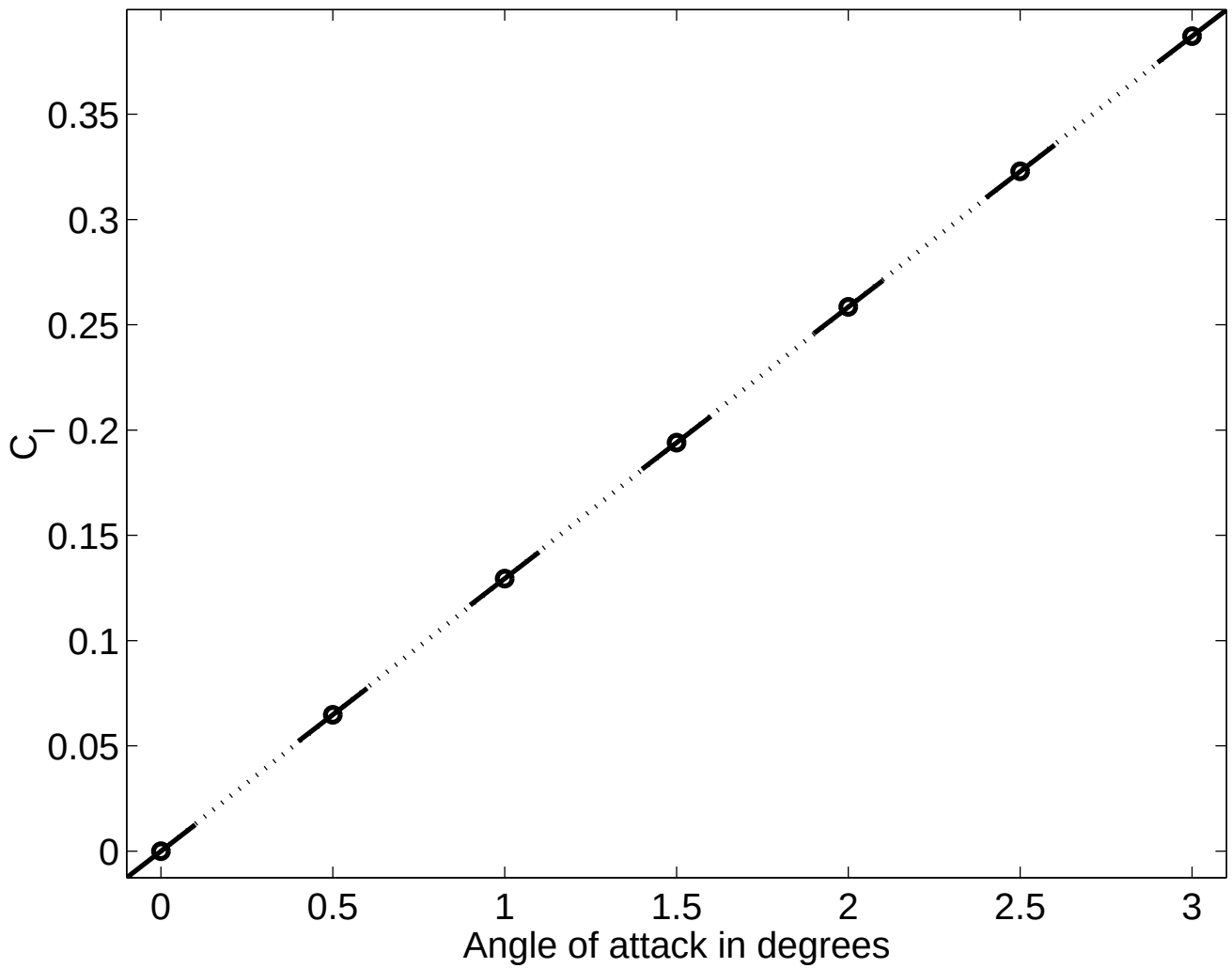
- Vertex averaging including weights are differentiated
- For transonic flow limiter is also differentiated
- Adjoint solver requires 38% more memory compared to flow solver
- Time per adjoint iteration = Twice the time per flow solver iteration
- Convergence acceleration using LUSGS with spectral-radius approximation
- Automated using `Makefile`
- Adjoint validation for $J(\alpha) = C_l$

Grid for NACA-0012: 160×40

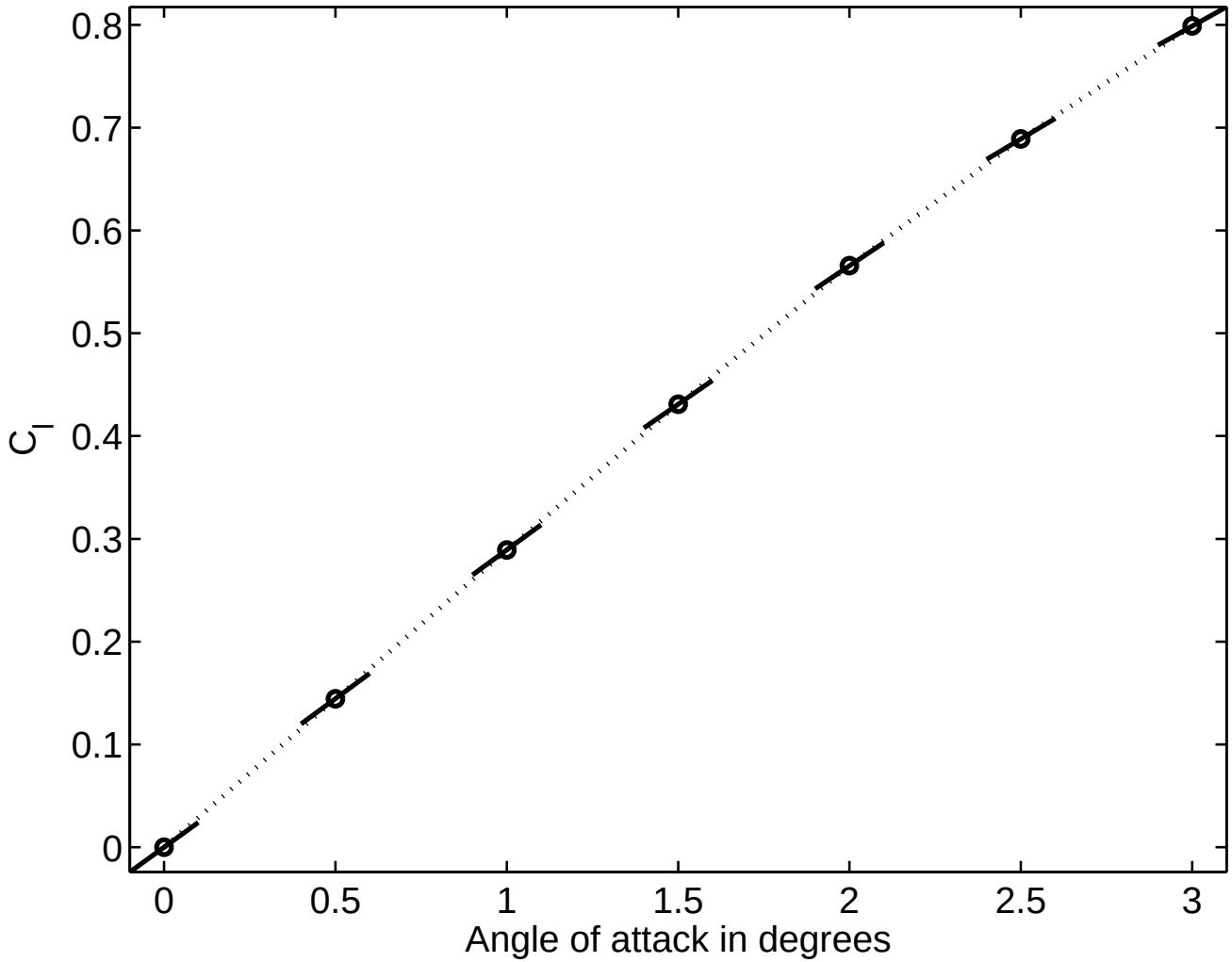
Number of vertices = 6560
Number of triangles = 12800



Validation of adjoints: Lift-curve slope, subsonic fbw



Validation of adjoints: Lift-curve slope, transonic flow

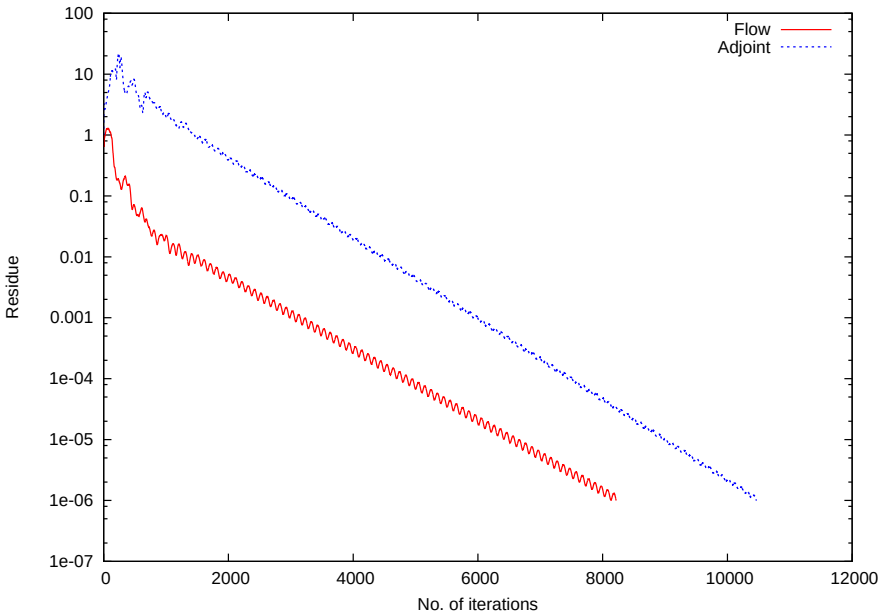


Lift-curve slope: C_{l_α}

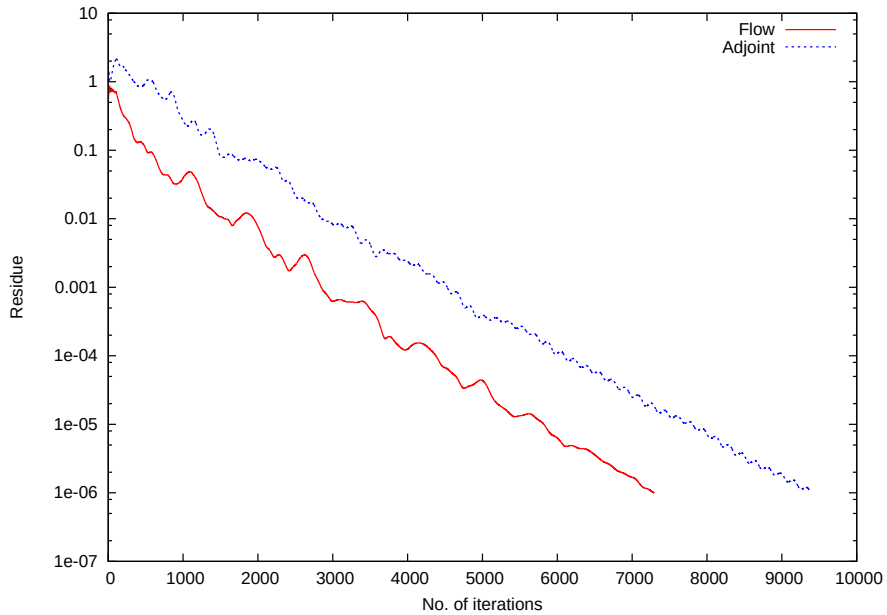
AOA	$M_\infty = 0.4$		$M_\infty = 0.8$	
	C_{l_α}	$C_{l_\alpha}/C^{\text{th}}_{l_\alpha}$	C_{l_α}	$C_{l_\alpha}/C^{\text{th}}_{l_\alpha}$
0.0	7.2095	1.0516	13.7506	1.3130
0.5	7.2069	1.0512	14.0556	1.3422
1.0	7.2000	1.0502	13.9544	1.3325
1.5	7.1901	1.0487	13.2116	1.2616
2.0	7.1789	1.0471	12.8314	1.2253
2.5	7.1675	1.0455	11.4077	1.0893
3.0	7.1562	1.0438	10.7135	1.0230

Convergence history: $\alpha = 3^o$

$M_\infty = 0.4$

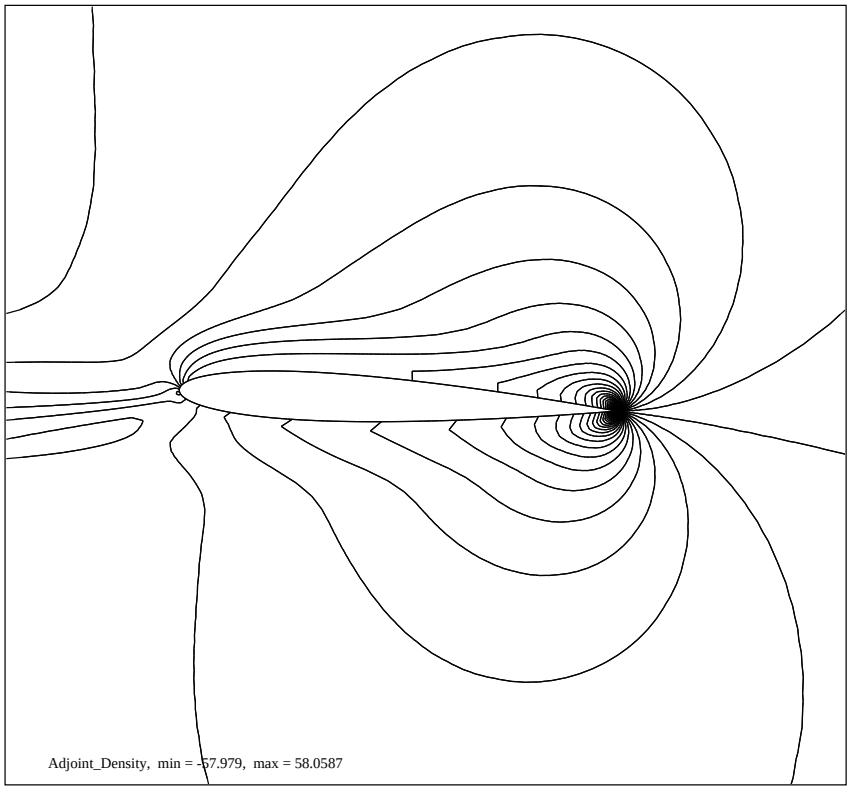


$M_\infty = 0.8$

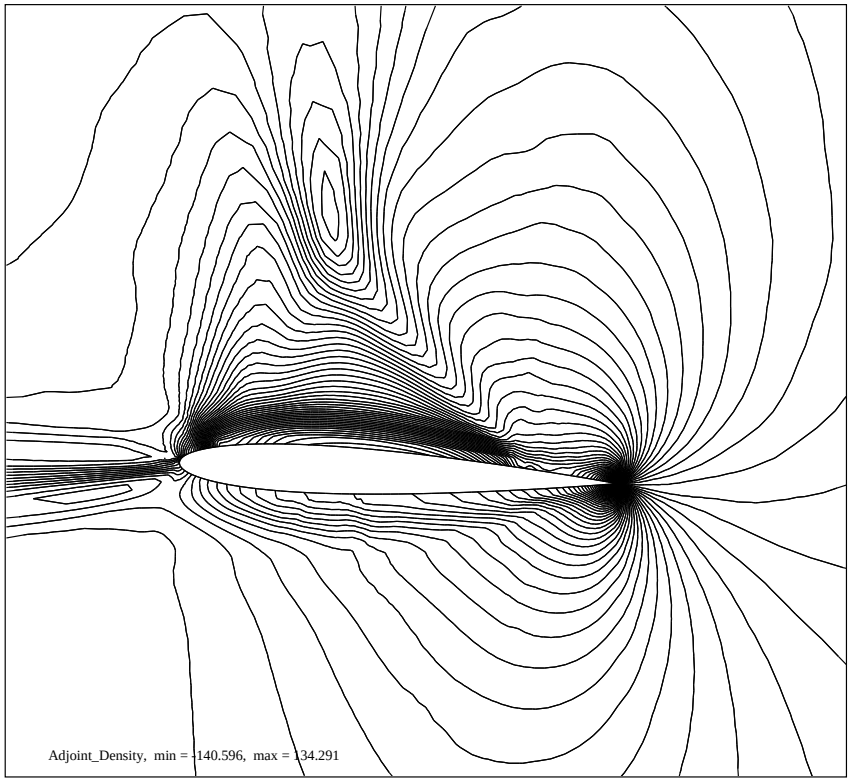


Adjoint density for lift coeffi cient: $\alpha = 3^o$

$M_\infty = 0.4$



$M_\infty = 0.8$



Outline

Future work

- How to deform the grid ?
- How to account for grid perturbation ?
- Smoothing of gradients in 2-D
- Application to 2-D optimization problems
- Extension to viscous flows
- Grid adaptation