Adjoint approach to optimization

Praveen. C
praveen@math.tifrbng.res.in

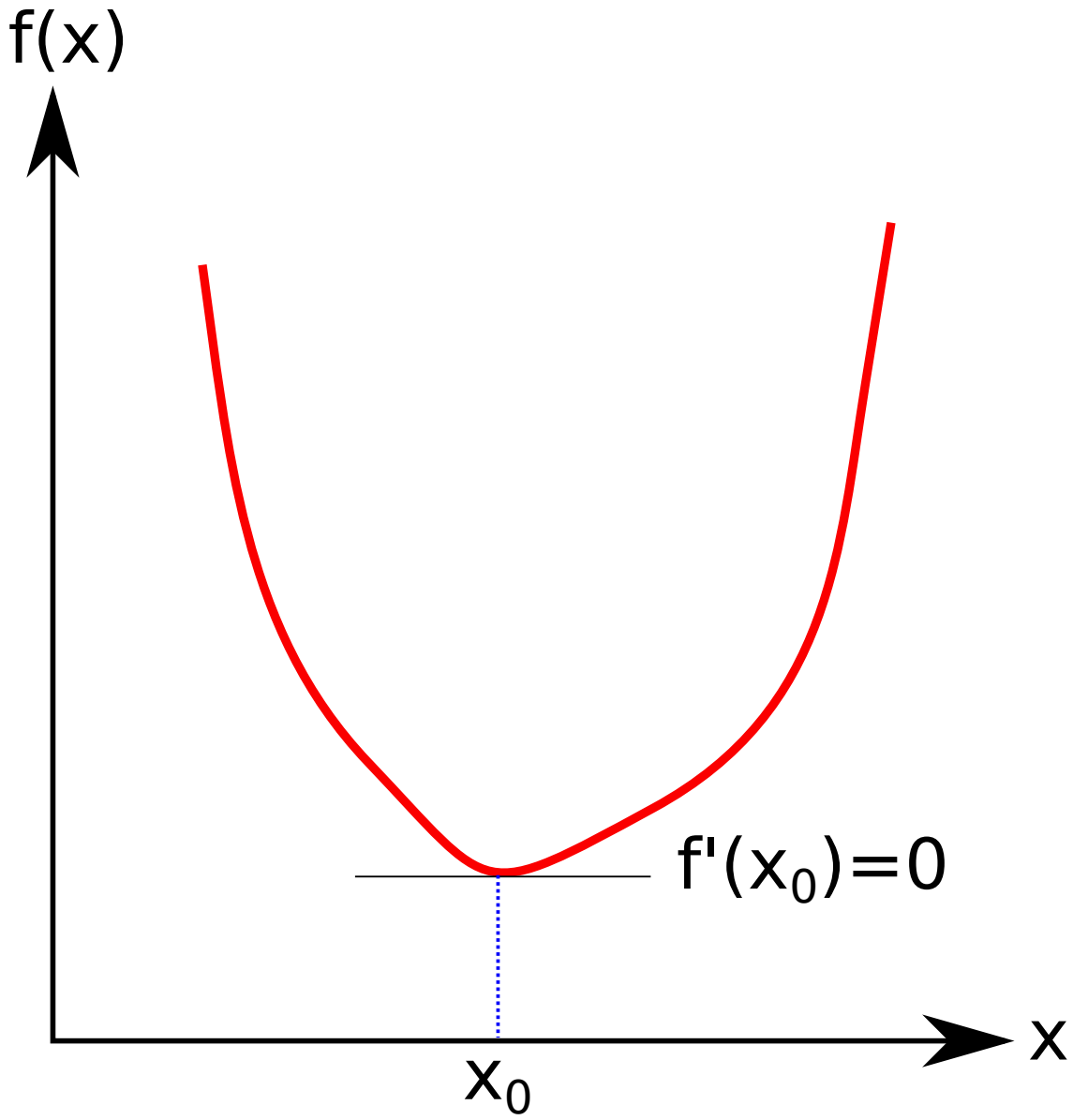
Tata Institute of Fundamental Research
Center for Applicable Mathematics
Bangalore 560065
http://math.tifrbng.res.in

Health, Safety and Environment Group
BARC
6-7 October, 2010

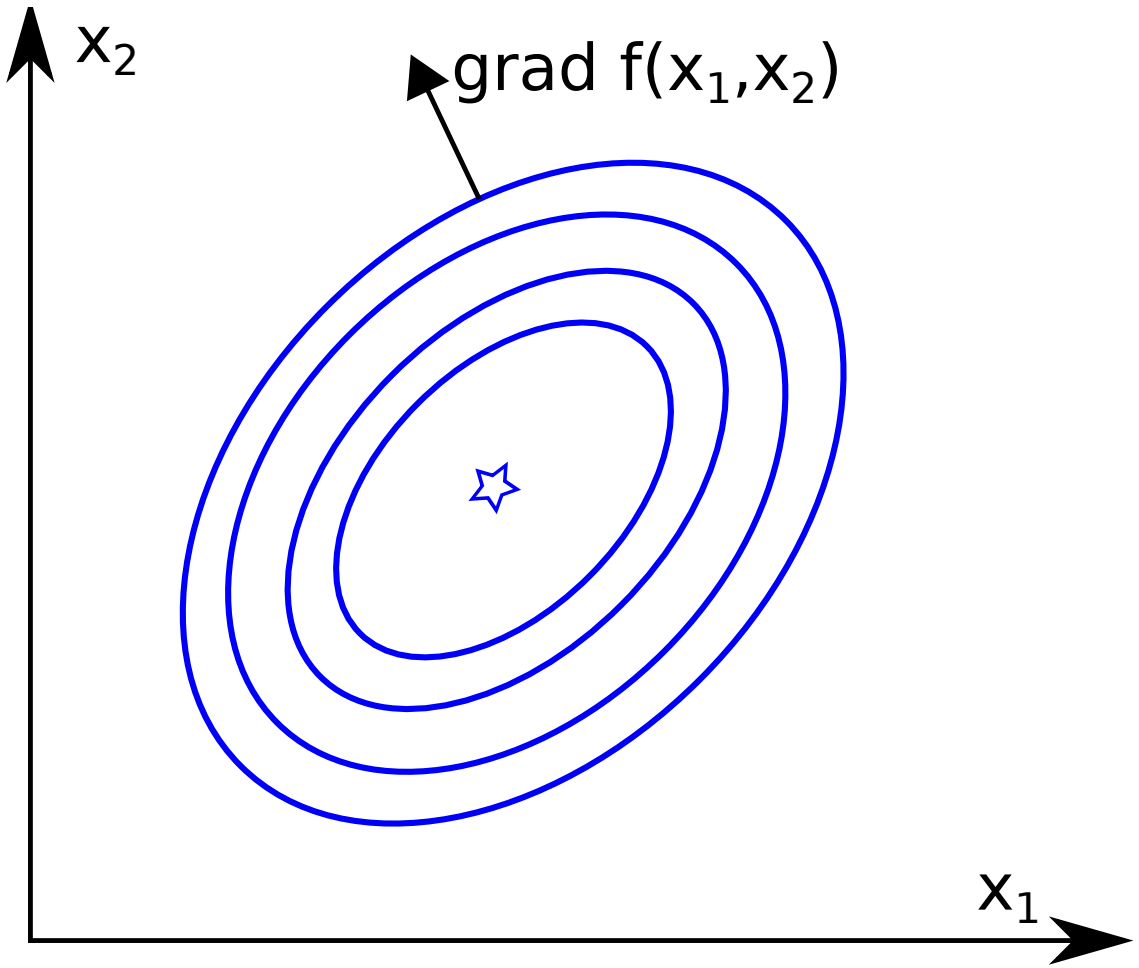
Outline

- Minimum of a function
- Constrained minimization
- Finite difference approach
- Adjoint approach
- Automatic differentiation
- Example

Minimum of a function



Steepest descent method



$$x^{n+1} = x^n - s^n \nabla f(x^n)$$

Objectives and controls

- Objective function
$$J(\alpha) = J(\alpha, u)$$
mathematical representation of system performance
- Control variables α
  - Parametric controls $\alpha \in \mathbb{R}^n$
  - Infinite dimensional controls $\alpha : X \to Y$
  - Shape $\alpha \in$ set of admissible shapes
- State variable u: solution of an ODE or PDE

$$R(\alpha, u) = 0$$

$$\Downarrow$$

$$u = u(\alpha)$$

Gradient-based minimization: blackbox/FD approach

$$\min_{\beta \in \mathbb{R}^N} \mathfrak{I}(\beta, Q(\beta))$$

- Initialize β^0, $n = 0$
- For $n = 0, \ldots, N_{iter}$
  - Solve $R(\beta^n, Q^n) = 0$
  - For $j = 1, \ldots, N$
    - $\beta^n_{(j)} = [\beta^n_1, \ldots, \beta^n_j + \Delta\beta_j, \ldots, \beta^n_N]^\top$
    - Solve $R(\beta^n_{(j)}, Q^n_{(j)}) = 0$
    - $\frac{\mathrm{d}\mathfrak{I}}{\mathrm{d}\beta_j} \approx \frac{\mathfrak{I}(\beta^n_{(j)}, Q^n_{(j)}) - \mathfrak{I}(\beta^n, Q^n)}{\Delta\beta_j}$
  - Steepest descent step

$$\beta^{n+1} = \beta^n - s^n \frac{\mathrm{d}\mathfrak{I}}{\mathrm{d}\beta}(\beta^n)$$

Cost of FD-based steepest-descent

$$\text{Cost} = O(N+1)N_{iter} = O(N+1)O(N) = O(N^2)$$

Accuracy of FD: Choice of step size

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x_0) = \frac{f(x_0+\delta) - f(x_0)}{\delta} + O(\delta)$$

In principle, if we choose a small δ, we reduce the error.

But computers have finite precision. Instead of $f(x_0)$ the computers gives $f(x_0) + O(\epsilon)$ where ϵ = machine precision.

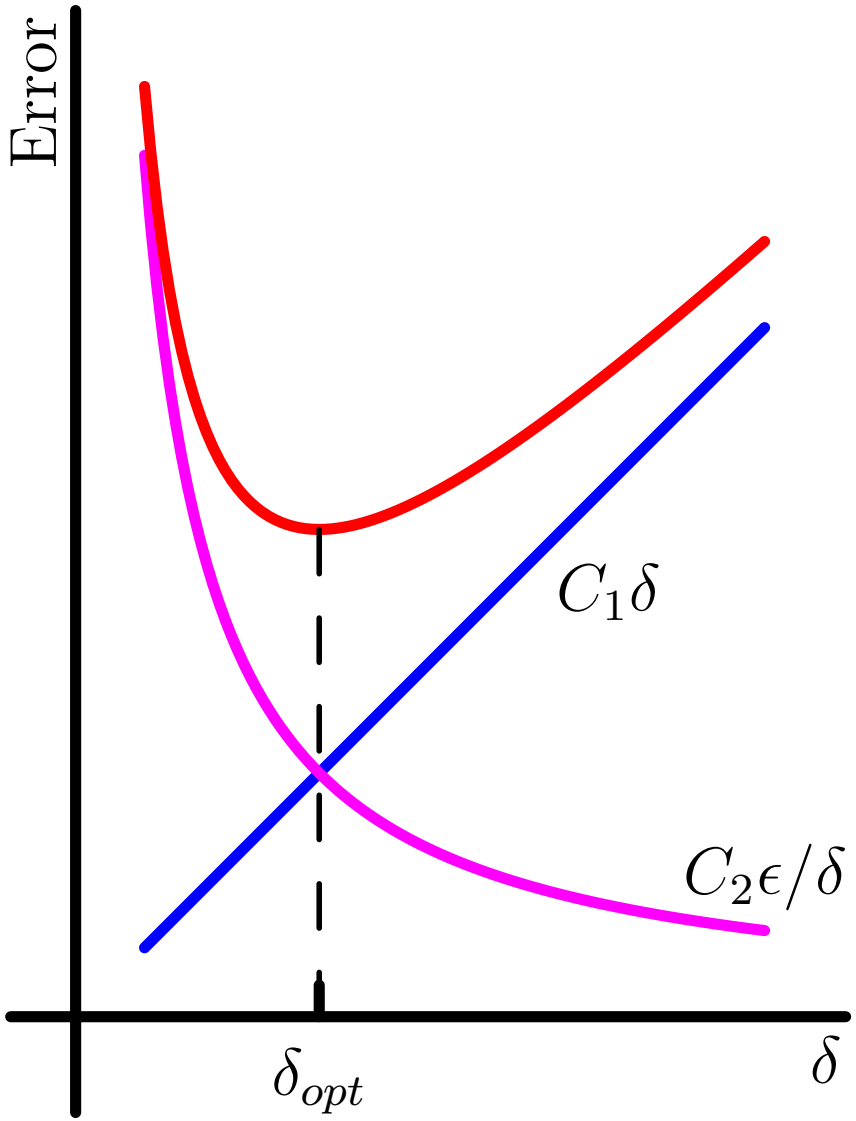
$$\frac{[f(x_0+\delta) + O(\epsilon)] - [f(x_0) + O(\epsilon)]}{\delta} = \frac{f(x_0+\delta) - f(x_0)}{\delta} + C_1 \frac{\epsilon}{\delta}$$

$$= \frac{\mathrm{d}}{\mathrm{d}x} f(x_0) + \underbrace{C_2\delta + C_1\frac{\epsilon}{\delta}}_{\text{Total error}}, \qquad C_1, C_2 \text{ depend on } f, x_0, \text{precision}$$

Total error is least when

$$\delta = \delta_{\text{opt}} = C_3\sqrt{\epsilon}, \quad C_3 = \sqrt{C_1/C_2}$$

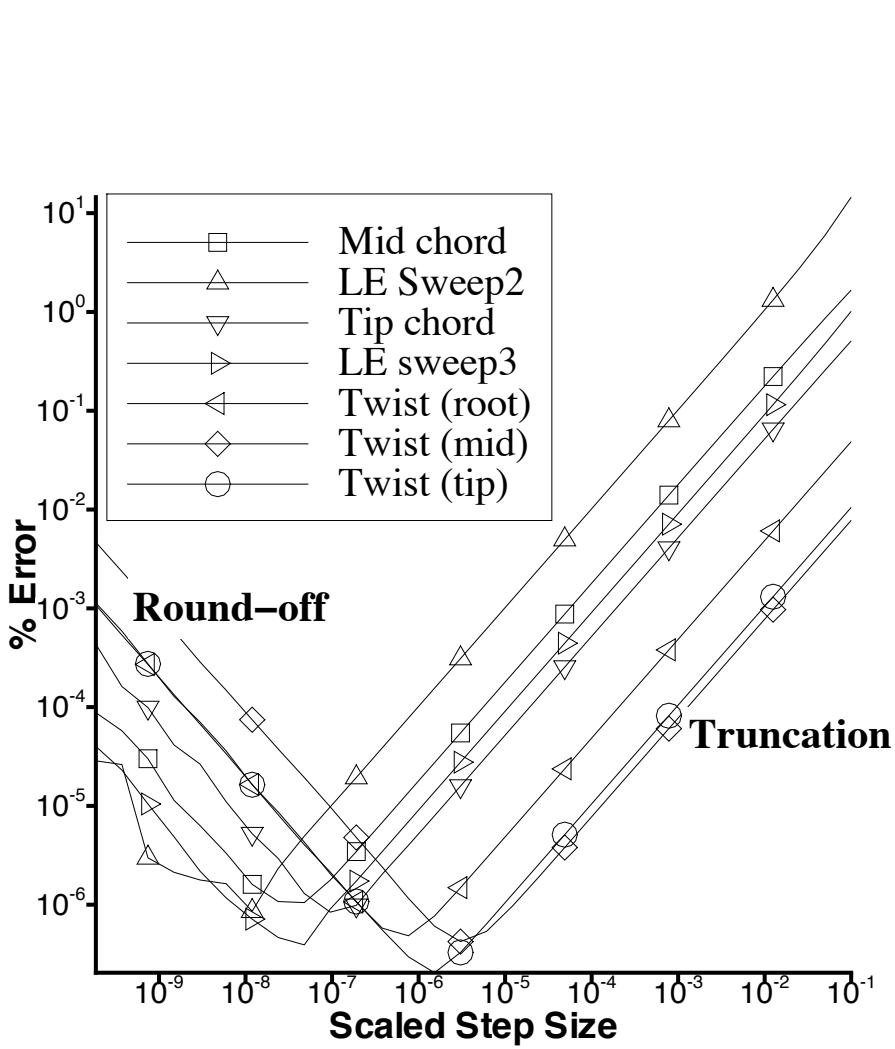
Accuracy of FD: Choice of step size

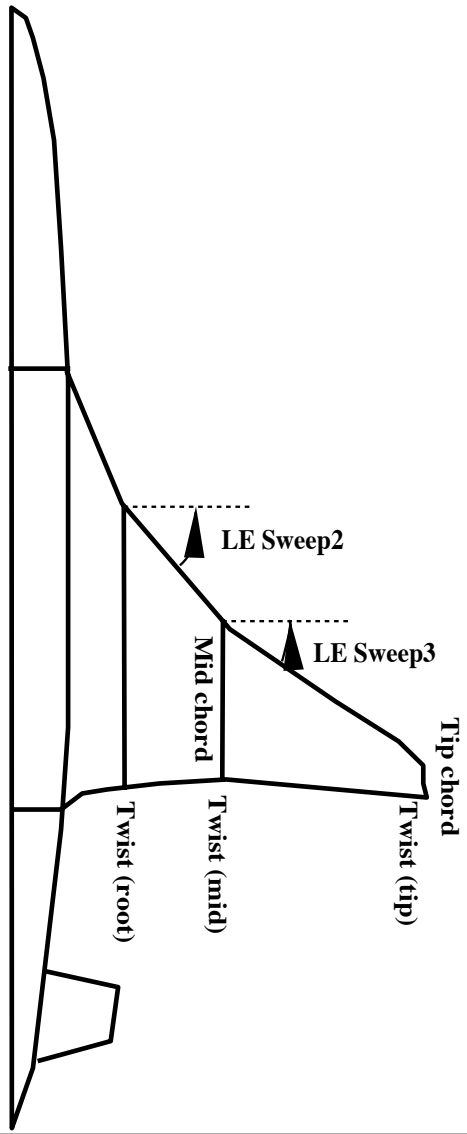


Precision	ϵ	δ_{opt}
Single	10^{-8}	10^{-4}
Double	10^{-16}	10^{-8}

See:
Brian J. McCartin, *Seven Deadly Sins of Numerical Computation* The American Mathematical Monthly, Vol. 105, No. 10 (Dec., 1998), pp. 929-941

Drag gradient using FD (Samareh)





Iterative problems

$$\mathfrak{I}(\beta, Q), \quad \text{where} \quad R(\beta, Q) = 0$$

- Q is implicitly defined, require an iterative solution method.
- Assume a Q^0 and iterate $Q^n \longrightarrow Q^{n+1}$ until $||R(\beta, Q^n)|| \leq$ `TOL`
- If `TOL` is too small, need too many iterations
- Many problems, we cannot reduce $||R(\beta, Q^n)||$ to small values
- This means that numerical value of $\mathfrak{I}$ will be noisy
  Finite difference will contain too much error, and is useless

RAE5243 airfoil, Mach=0.68, Re=19 million, AOA=2.5 deg.

iter	Lift	Drag
41496	0.824485788042416	1.627593747613790E-002
41497	0.824485782714867	1.627593516695762E-002
41498	0.824485777387834	1.627593285794193E-002
41499	0.824485772061306	1.627593054909022E-002
41500	0.824485766735297	1.627592824040324E-002

Complex variable method

$$f(x_0 + i\delta) = f(x_0) + i\delta f'(x_0) + O(\delta^2) + iO(\delta^3)$$

$$f'(x_0) = \frac{1}{\delta}\text{imag}[f(x_0 + i\delta)] + O(\delta^2)$$

- No roundoff error
- We can take δ to be very small, $\delta = 10^{-20}$
- Can be easily implemented
  - fortran: redeclare real variables as complex
  - matlab: no change
- Iterative problems: $\beta \longrightarrow \beta + i\Delta\beta$
  - Obtain $\tilde{Q} = Q(\beta + i\Delta\beta)$ by solving $R(\beta + i\Delta\beta, \tilde{Q}) = 0$
  - Then gradient

$$\mathfrak{I}'(\beta) \approx \frac{1}{\Delta\beta}\text{imag}[\mathfrak{I}(\beta + i\Delta\beta, Q(\beta + i\Delta\beta)]$$

- Computational cost is $O(N^2)$ or higher (due to complex arithmetic)

Objectives and controls

- Objective function
$$J(\alpha) = J(\alpha, u)$$
mathematical representation of system performance
- Control variables α
  - Parametric controls $\alpha \in \mathbb{R}^n$
  - Infinite dimensional controls $\alpha : X \to Y$
  - Shape $\alpha \in$ set of admissible shapes
- State variable u: solution of an ODE or PDE
$$R(\alpha, u) = 0$$

Mathematical formulation

- Constrained minimization problem

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

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

$$\begin{aligned} \delta J &= \frac{\partial J}{\partial \alpha}\delta\alpha + \frac{\partial J}{\partial u}\delta u \\ &= \frac{\partial J}{\partial \alpha}\delta\alpha + \frac{\partial J}{\partial u}\frac{\partial u}{\partial \alpha}\delta\alpha \\ &= \left[\frac{\partial J}{\partial \alpha} + \frac{\partial J}{\partial u}\frac{\partial u}{\partial \alpha}\right]\delta\alpha =: G\delta\alpha \end{aligned}$$

- 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 $\delta J < 0$

$$\begin{aligned} \delta J &= \frac{\partial J}{\partial \alpha}\delta\alpha + \frac{\partial J}{\partial u}\delta u \\ &= \frac{\partial J}{\partial \alpha}\delta\alpha + \frac{\partial J}{\partial u}\frac{\partial u}{\partial \alpha}\delta\alpha \\ &= \left[\frac{\partial J}{\partial \alpha} + \frac{\partial J}{\partial u}\frac{\partial u}{\partial \alpha}\right]\delta\alpha =: G\delta\alpha \end{aligned}$$

- Steepest descent

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

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

Sensitivity approach

- Linearized state equation $R(\alpha, u) = 0$

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

or

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

- Solve sensitivity equation iteratively, e.g.,

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

- Gradient

$$\frac{\mathrm{d}J}{\mathrm{d}\alpha} = \frac{\partial J}{\partial \alpha} + \frac{\partial J}{\partial u}\frac{\partial u}{\partial \alpha}$$

Sensitivity approach: Computational cost

- n design variables: $\alpha = (\alpha_1, \ldots, \alpha_n)$
- Solve primal problem $R(\alpha, u) = 0$ to get $u(\alpha)$
- For $i = 1, \ldots, n$
  - Solve sensitivity equation wrt α_i

$$\frac{\partial R}{\partial u}\frac{\partial u}{\partial \alpha_i} = -\frac{\partial R}{\partial \alpha_i}$$

  - Compute derivative wrt α_i

$$\frac{\mathrm{d}J}{\mathrm{d}\alpha_i} = \frac{\partial J}{\partial \alpha_i} + \frac{\partial J}{\partial u}\frac{\partial u}{\partial \alpha_i}$$

- One primal equation, n sensitivity equations
  Computational cost $= n + 1$

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

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

Adjoint approach: Computational cost

- n design variables: $\alpha = (\alpha_1, \ldots, \alpha_n)$
- Solve primal problem $R(\alpha, u) = 0$ to get $u(\alpha)$
- Solve adjoint problem

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

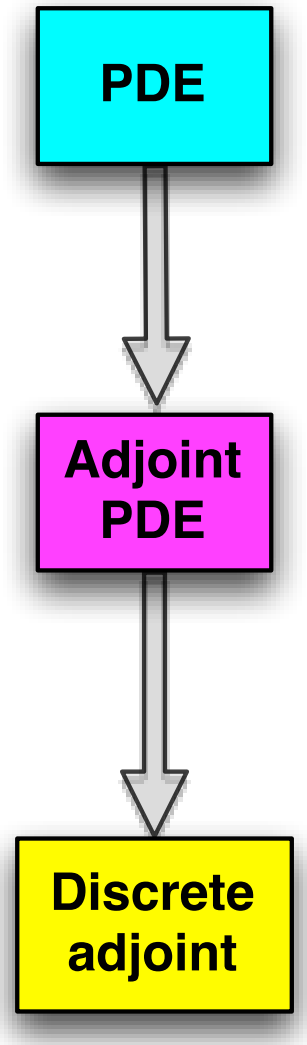
- For $i = 1, \ldots, n$
  - Compute derivative wrt α_i

$$\frac{\mathrm{d}J}{\mathrm{d}\alpha_i} = \frac{\partial J}{\partial \alpha_i} + v^\top \frac{\partial R}{\partial \alpha_i}$$

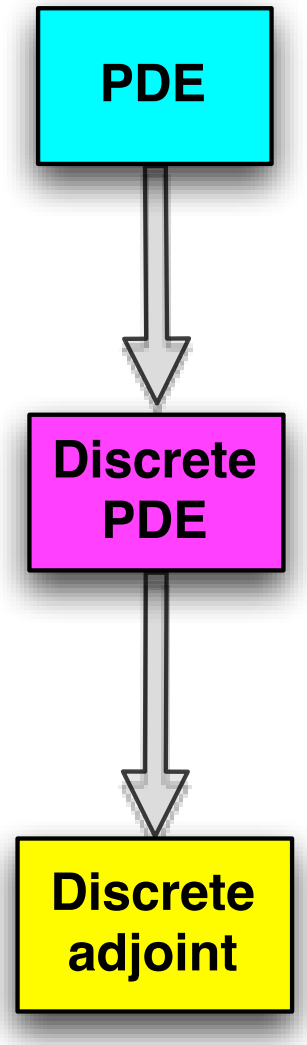
- One primal equation, one adjoint equation
  Computational cost $= 2$, independent of n

Adjoint: Two approaches

Continuous or differentiate-discretize



Discrete or discretize-differentiate



Techniques for computing gradients

- Hand differentiation
- Finite difference method
- Complex variable method
- Automatic Differentiation (AD)
  - Computer code to compute $J(\alpha, u)$ and $R(\alpha, u)$
  - Chain rule of differentiation
  - Generates a code to compute derivatives
  - ADIFOR, ADOLC, ODYSEE, TAMC, TAF, TAPENADE
    see `http://www.autodiff.org`

Derivatives

- Given a program `P` computing a function F

$$\begin{array}{rcccc} F & : & \mathbb{R}^m & \rightarrow & \mathbb{R}^n \\ & & X & \rightarrow & Y \end{array}$$

- build a program that computes derivatives of F
- X : independent variables
- Y : dependent variables

Derivatives

- Jacobian matrix: $J = \left[\frac{\partial Y_j}{\partial X_i} \right]$
- Directional or tangent derivative

$$\dot{Y} = J\dot{X}$$

- Adjoint mode

$$\bar{X} = J^\top \bar{Y}$$

- Gradients ($n = 1$ output)

$$J = \left[\frac{\partial Y}{\partial X_i} \right] = \nabla Y$$

Forward differentiation

- Program `P` is a sequence of computations F_k
- $T_0 = X$, given
- k'th line

$$t_k = F_k(T_{k-1}), \qquad T_k = \begin{bmatrix} T_{k-1} \\ t_k \end{bmatrix}$$

- Function is a composition

$$F = F_p \circ F_{p-1} \circ \ldots \circ F_2 \circ F_1$$

- Chain rule

$$\dot{Y} = F'(X)\dot{X} = F'_p(T_{p-1})F'_{p-1}(T_{p-2}) \ldots F'_2(T_1)F'_1(T_0)\dot{X}$$

$$X, \dot{X} \to Y, \dot{Y}$$

- $cost(\dot{Y}) = 4 * cost(Y)$

Differentiation: Example

- A simple example

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

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

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

F77 code: `costfunc.f`

```
      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

- Apply chain rule of differentiation

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

- $\boxed{\dot{x} = 1,\ \dot{y} = 0,\ \dot{t}_{10} = \frac{\partial f}{\partial x}}$ and $\boxed{\dot{x} = 0,\ \dot{y} = 1,\ \dot{t}_{10} = \frac{\partial f}{\partial y}}$
- `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
```

Backward differentiation

- Program `P` is a sequence of computations F_k
- $T_0 = X$, given
- k'th line

$$t_k = F_k(T_{k-1}), \qquad T_k = \begin{bmatrix} T_{k-1} \\ t_k \end{bmatrix}$$

- Function is a composition

$$F = F_p \circ F_{p-1} \circ \ldots \circ F_2 \circ F_1$$

- Chain rule

$$\bar{X} = [F'(X)]^\top \bar{Y} = [F_1'(T_0)]^\top [F_2'(T_1)]^\top \ldots [F_{p-1}'(T_{p-2})]^\top [F_p'(T_{p-1})]^\top \bar{Y}$$

$$X, \bar{Y} \to \bar{X}$$

- $cost(\bar{X}) = 4 * cost(Y)$

Differentiation: Reverse mode

- Apply chain rule of differentiation in reverse

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

- $\bar{t}_1 = \frac{\partial f}{\partial x}$, $\bar{t}_2 = \frac{\partial f}{\partial y}$
- `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

$$F : \mathbb{R}^m \to \mathbb{R}^n$$

- Direct mode
$$cost(J) = m * 4 * cost(P)$$
- Reverse mode
$$cost(J) = n * 4 * cost(P)$$
- Scalar output $F \in \mathbb{R}$, $n = 1$
  - Direct mode gives $\nabla F \cdot \dot{X}$ for given vector $\dot{X}$
  - Reverse mode gives ∇F, hence preferred
- Vector output $F \in \mathbb{R}^n$
  - Direct mode gives $\nabla F \cdot \dot{X}$ for given vector $\dot{X}$
    use for sensitivity equation approach
  - Reverse mode gives $(\nabla F)^\top \cdot \bar{Y}$
    use for adjoint approach

Black-box AD

- `cost` depends on `alpha`

```
call ComputeCost(alpha, cost)
```

- Subroutine for cost function

```
subroutine ComputeCost(alpha, cost)
   call SolveState(alpha, u)
   call CostFun(alpha, u, cost)
end
```

- Reverse differentiation using AD

```
tapenade -backward \
         -head ComputeCost \
         -vars alpha \
         -outvars cost \
         ComputeCost.f SolveState.f CostFun.f
```

Black-box AD

- Diffentiated subroutines:
  `ComputeCost_b.f`, `SolveState_b.f`, `CostFun_b.f`

```
subroutine ComputeCost_b(alpha,alphab,cost,costb)
    call SolveState(alpha, u)
    call CostFun_b(alpha, alphab, u, ub, cost, costb)
    call SolveState_b(alpha, alphab, u, ub)
end
```

- Compute gradient using

```
costb = 1.0
call ComputeCost_b(alpha, alphab, cost, costb)
```

- Gradient given by `alphab`

$$\texttt{alphab} = \frac{\partial(\texttt{cost})}{\partial(\texttt{alpha})}$$

AD for iterative problems

- Solve state equation $R(\alpha, u) = 0$ as steady state of

$$\frac{\mathrm{d}u}{\mathrm{d}t} + R(\alpha, u) = 0, \quad u(0) = u_o$$

- State solver

```
subroutine SolveState(alpha, u)
   u = 0.0
   do
      call Residue(alpha, u, res)
      u = u - dt * res
   while( abs(res) > TOL)
end subroutine
```

AD for iterative problems

Adjoint solver, hand written

```
subroutine SolveState_b(alpha,alphab,u,ub)
resb = 0.0
do
   ub1 = 0.0
   call Residue_bu(alpha,u,ub1,res,resb)
   resb = resb - (ub + ub1)
while( abs(ub+ub1) .gt. 1.0e-5)
call Residue_ba(alpha,alphab,u,res,resb)
end subroutine
```

$$\texttt{resb} = \text{Adjoint variable}$$

$$\texttt{ub} = \frac{\partial J}{\partial u}, \quad \texttt{ub1} = \left[\frac{\partial R}{\partial u}\right]^{\top} v$$

Adjoint iterative scheme

- Forward iterations linearly stable

$$u^{n+1} = u^n - \Delta t \; R(\alpha, u^n), \quad \Delta t < S(\sigma(R'))$$

- Adjoint iteration

$$v^{n+1} = v^n - \Delta t \left\{ [R'(\alpha, u^\infty)]^\top v^n + \frac{\partial J}{\partial u} \right\}$$

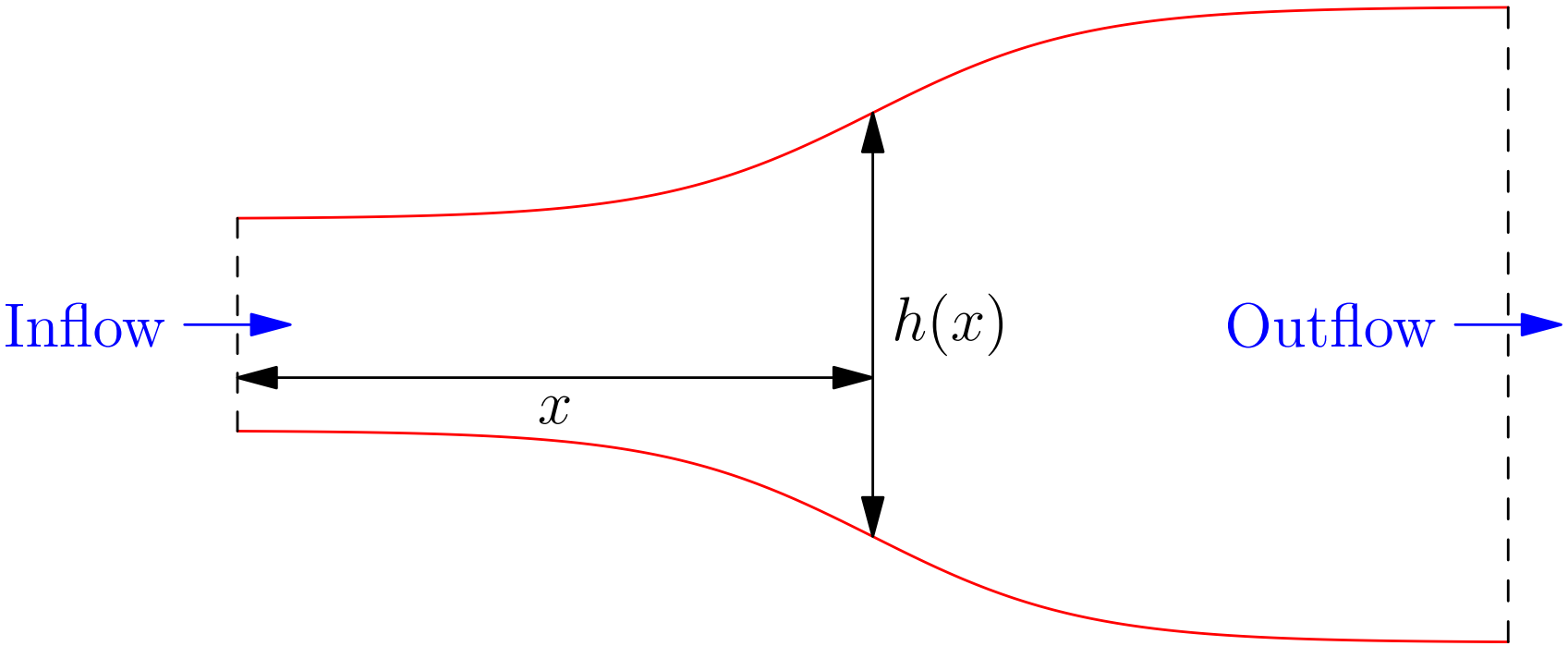
- $[R']^\top$ has same eigenvalues as $R' \Longrightarrow$ adjoint iterations stable under same condition on Δt
- Preconditioner for adjoint = (preconditioner for primal problem)$^\top$

AD tools

- Source transformation (ST) and operator overloading (OO)
- See `http://www.autodiff.org`

Tool	Modes	Method	Lang
ADIFOR	F	ST	Fortran
ADOLC	F/B	OO	C
TAPENADE	F/B	ST	Fortran/C
TAF/TAMC	F/B	ST	Fortran/C
MAD	F	OO	Matlab

Quasi 1-D flow



Quasi 1-D flow

- Quasi 1-D flow in a duct

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

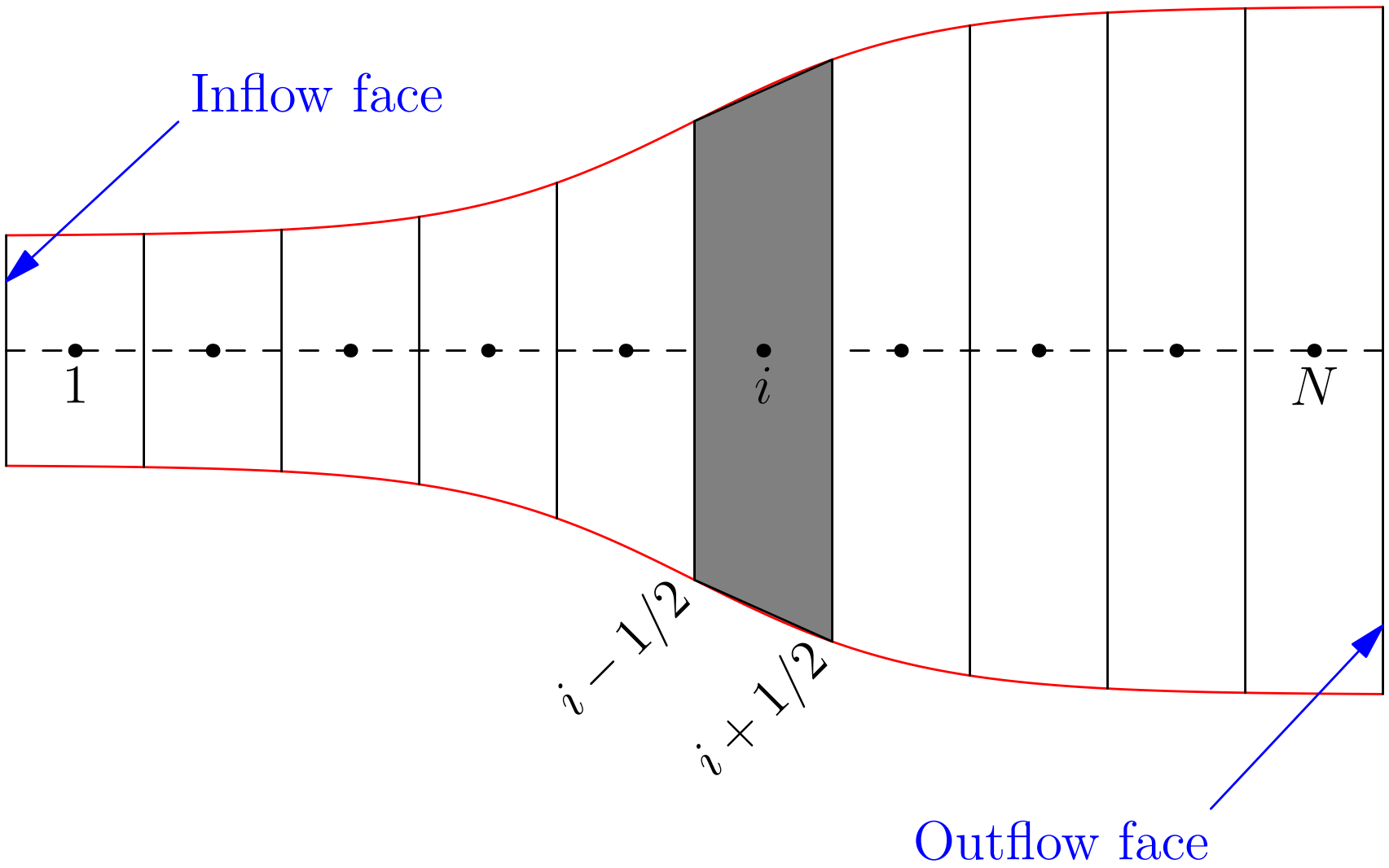
$$U = \begin{bmatrix} \rho \\ \rho u \\ E \end{bmatrix}, \quad f = \begin{bmatrix} \rho u \\ p + \rho u^2 \\ (E+p)u \end{bmatrix}, \quad P = \begin{bmatrix} 0 \\ p \\ 0 \end{bmatrix}$$

$$h(x) = \text{cross-section height of duct}$$

- 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 flow



Quasi 1-D flow

- Finite volume scheme for cell $C_i = [x_{i-1/2}, x_{i+1/2}]$

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

- Discrete cost function

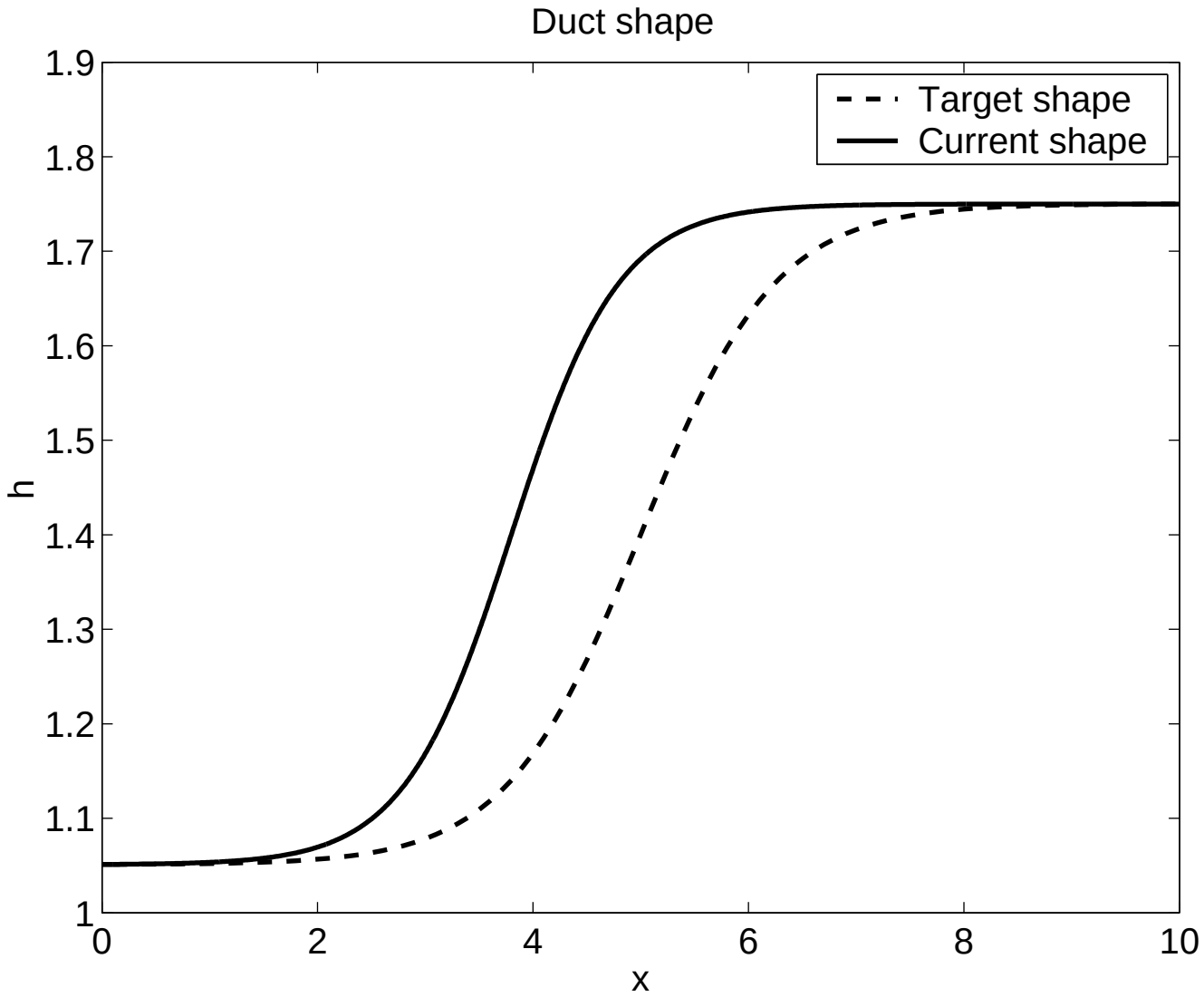
$$J = \sum_{i=1}^{N} (p_i - p_i^*)^2$$

- Control variables

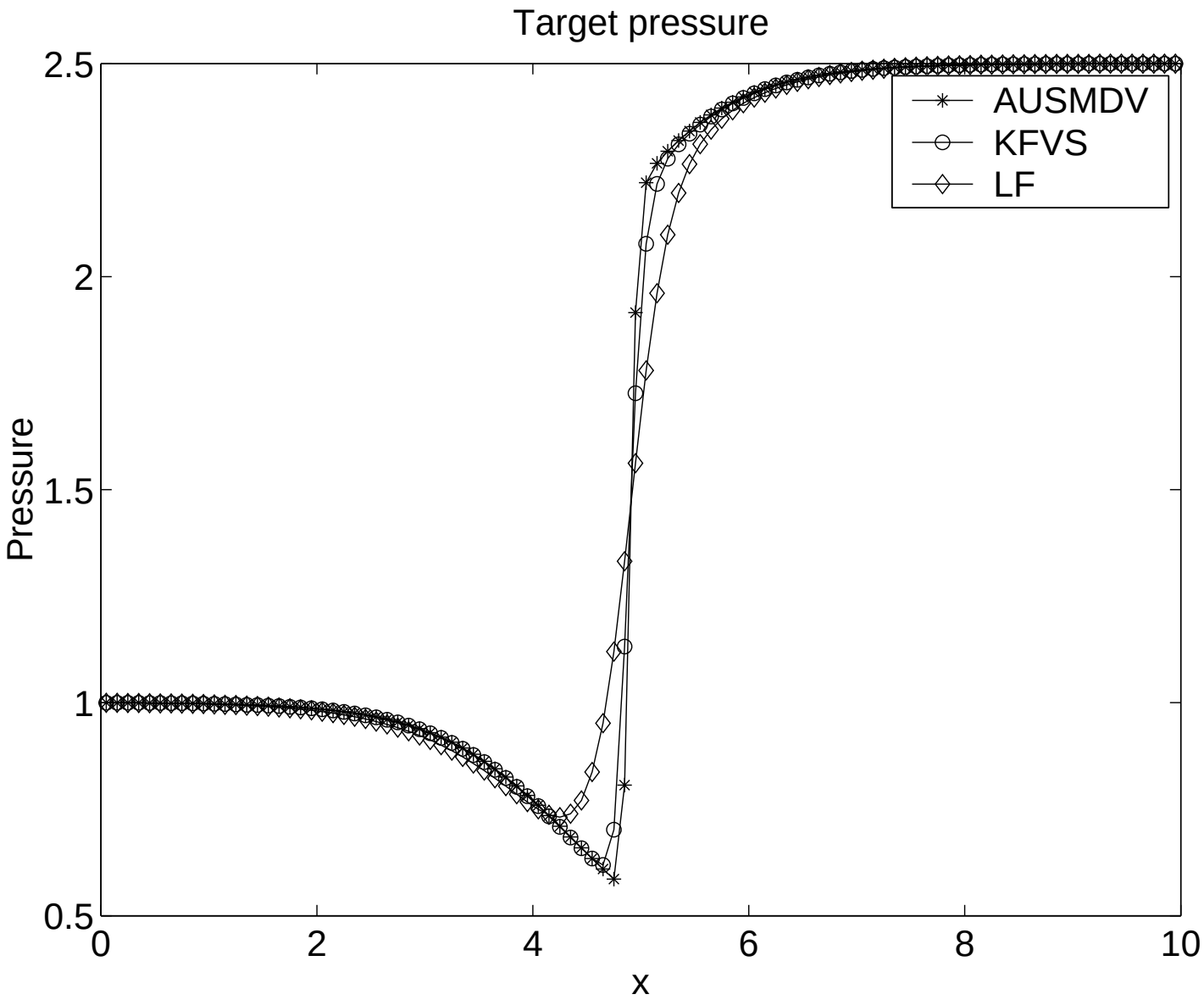
$$h_{1/2}, h_{1+1/2}, \ldots, h_{i+1/2}, \ldots, h_{N+1/2}$$

- $N = 100$

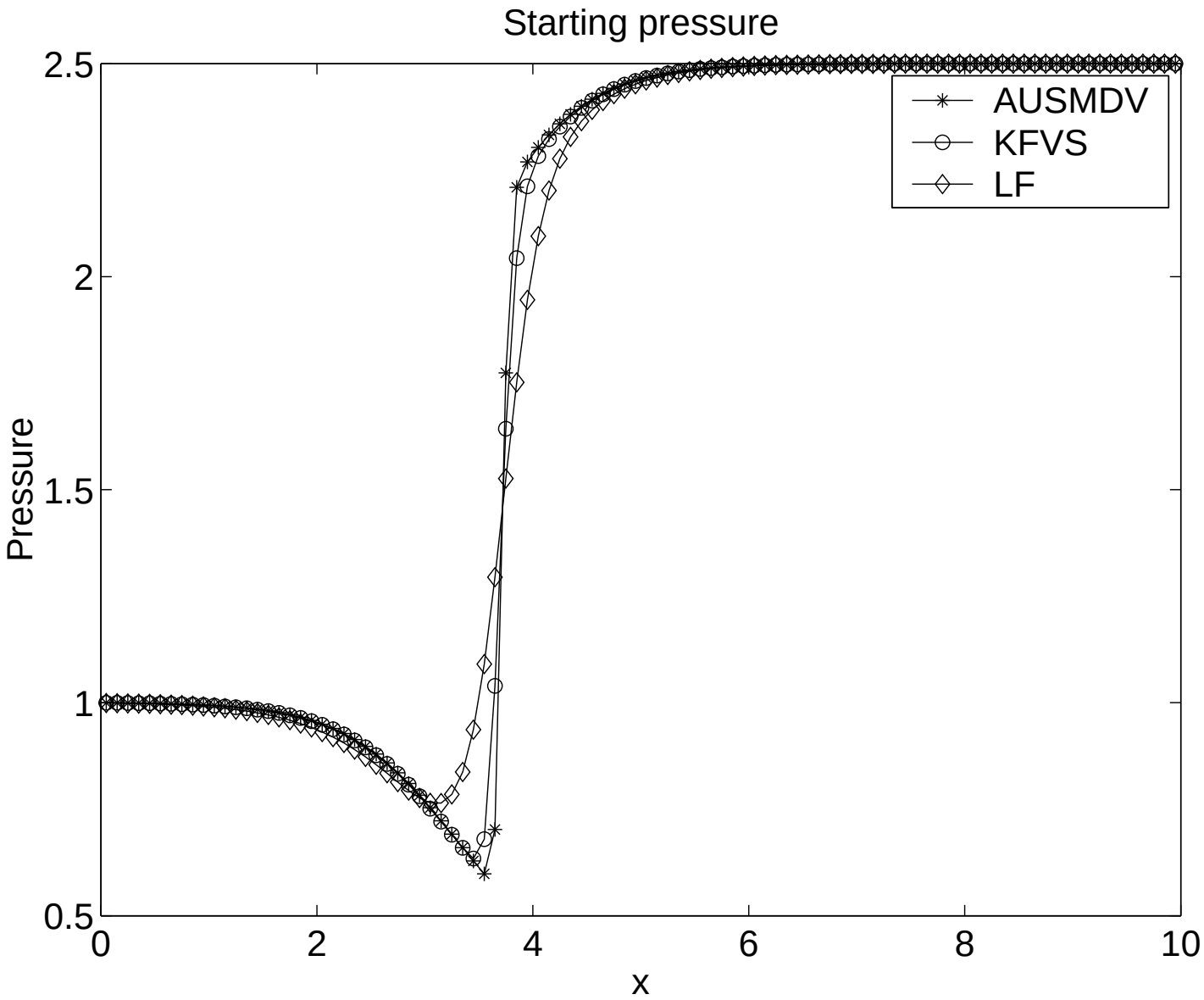
Duct shape


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

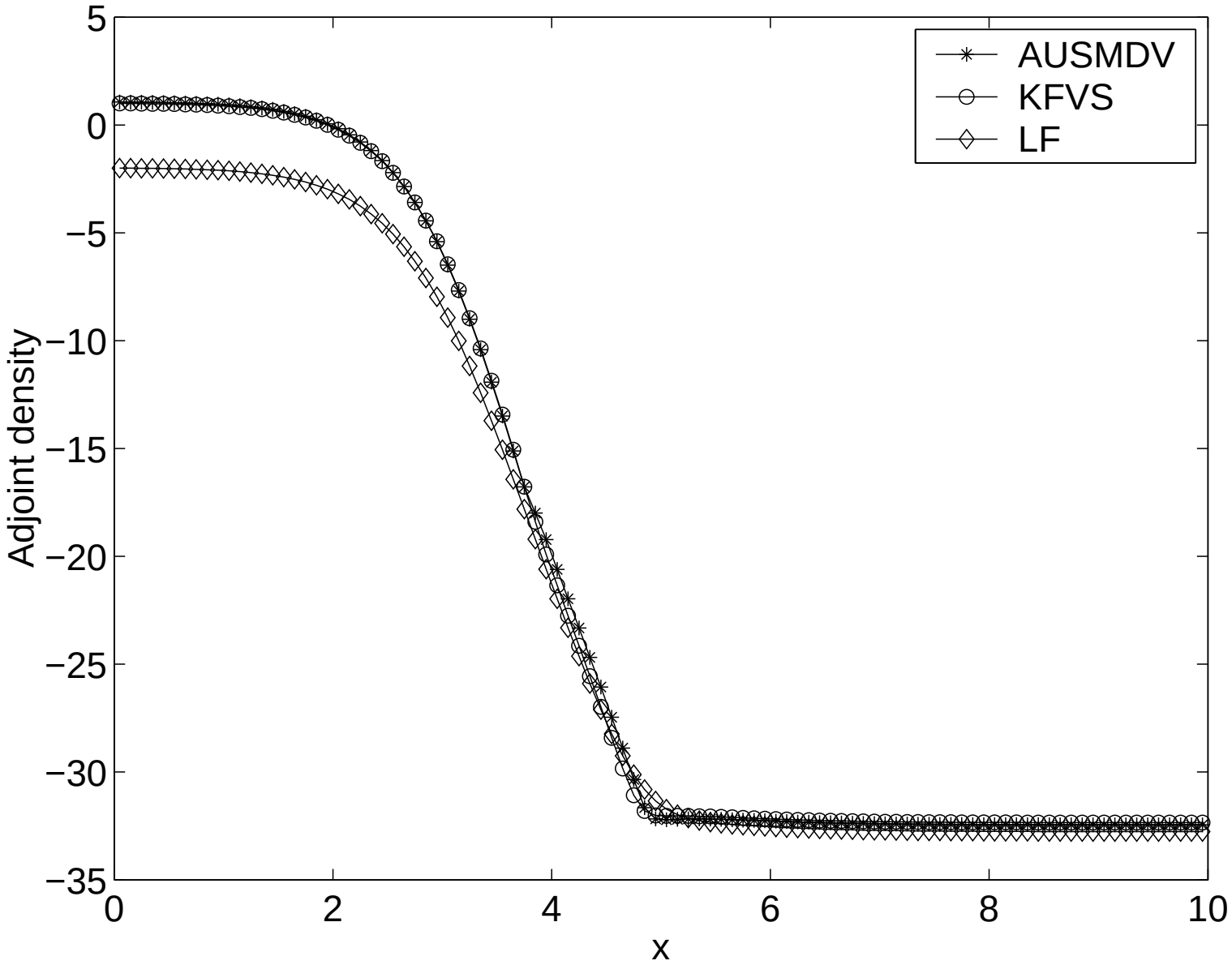
Target pressure distribution p^*



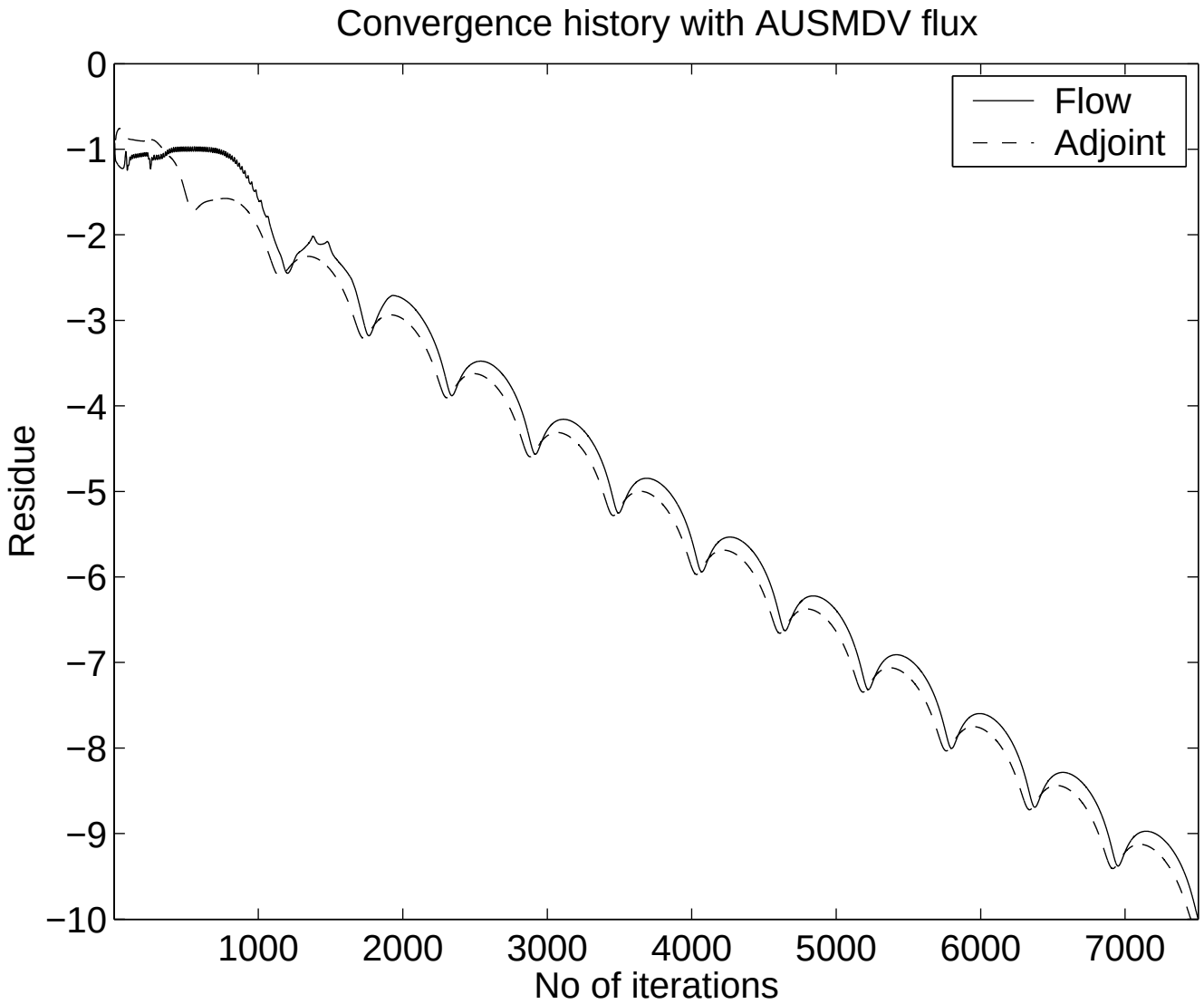
Current pressure distribution



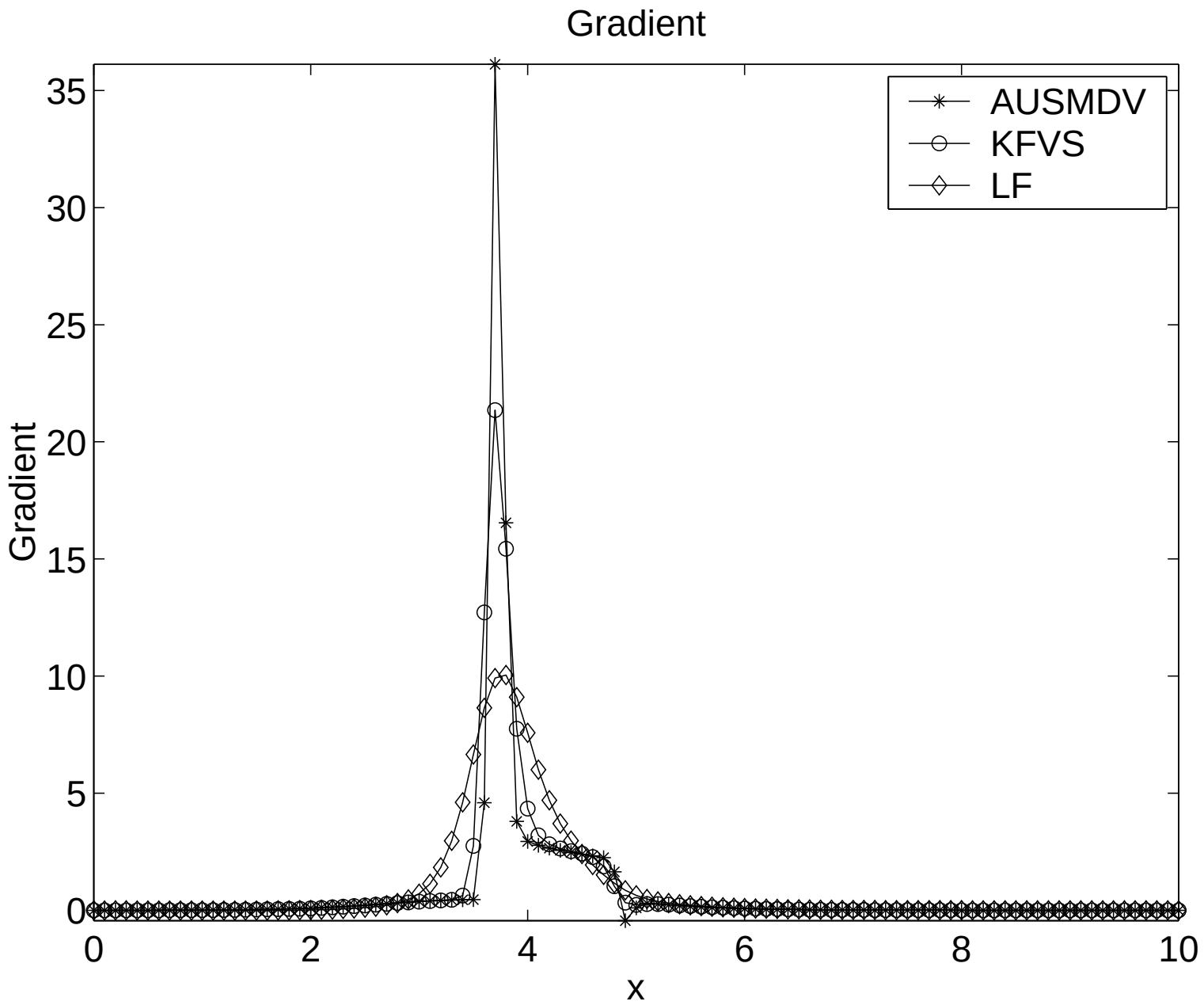
Adjoint density



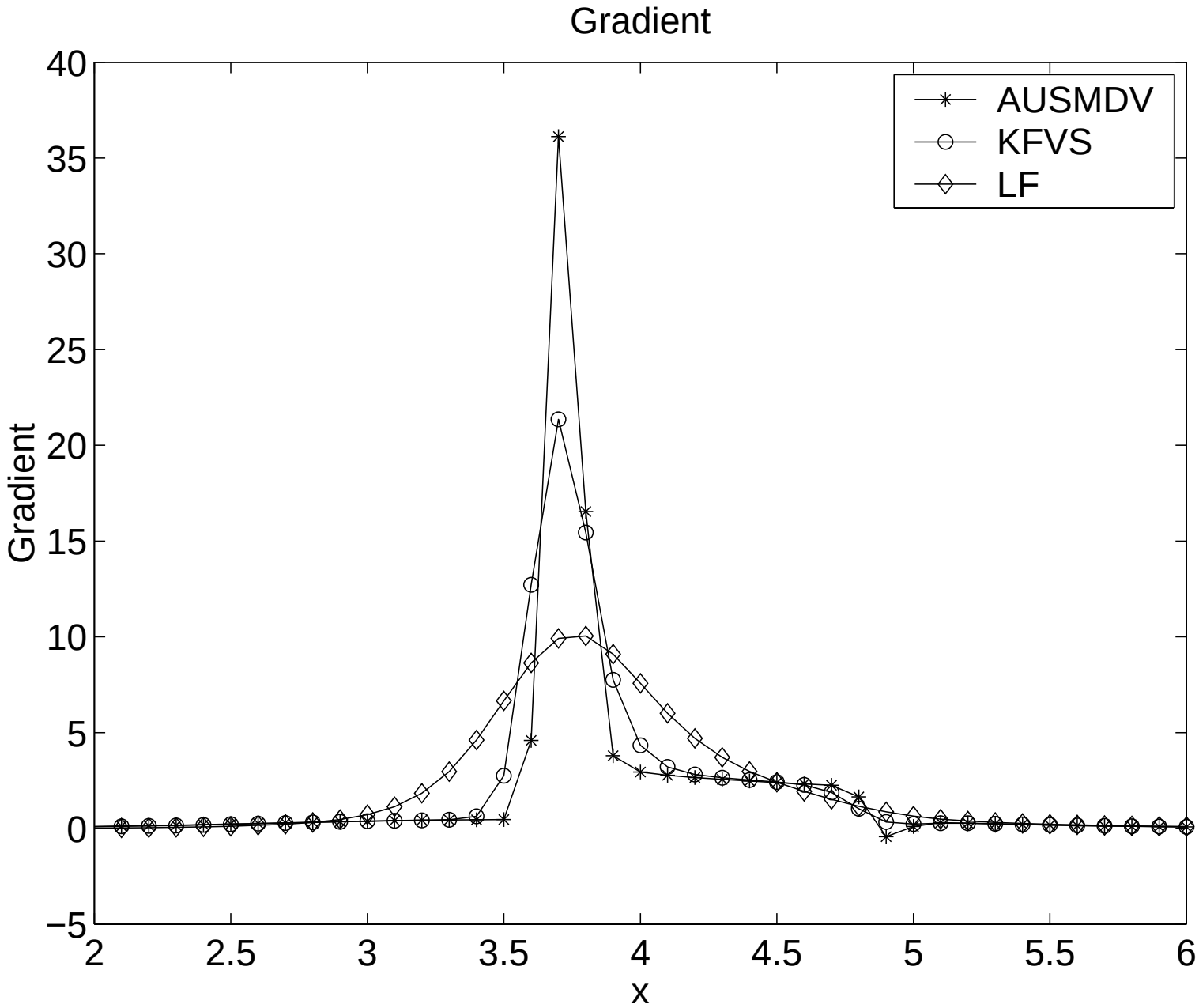
Convergence history: Explicit Euler



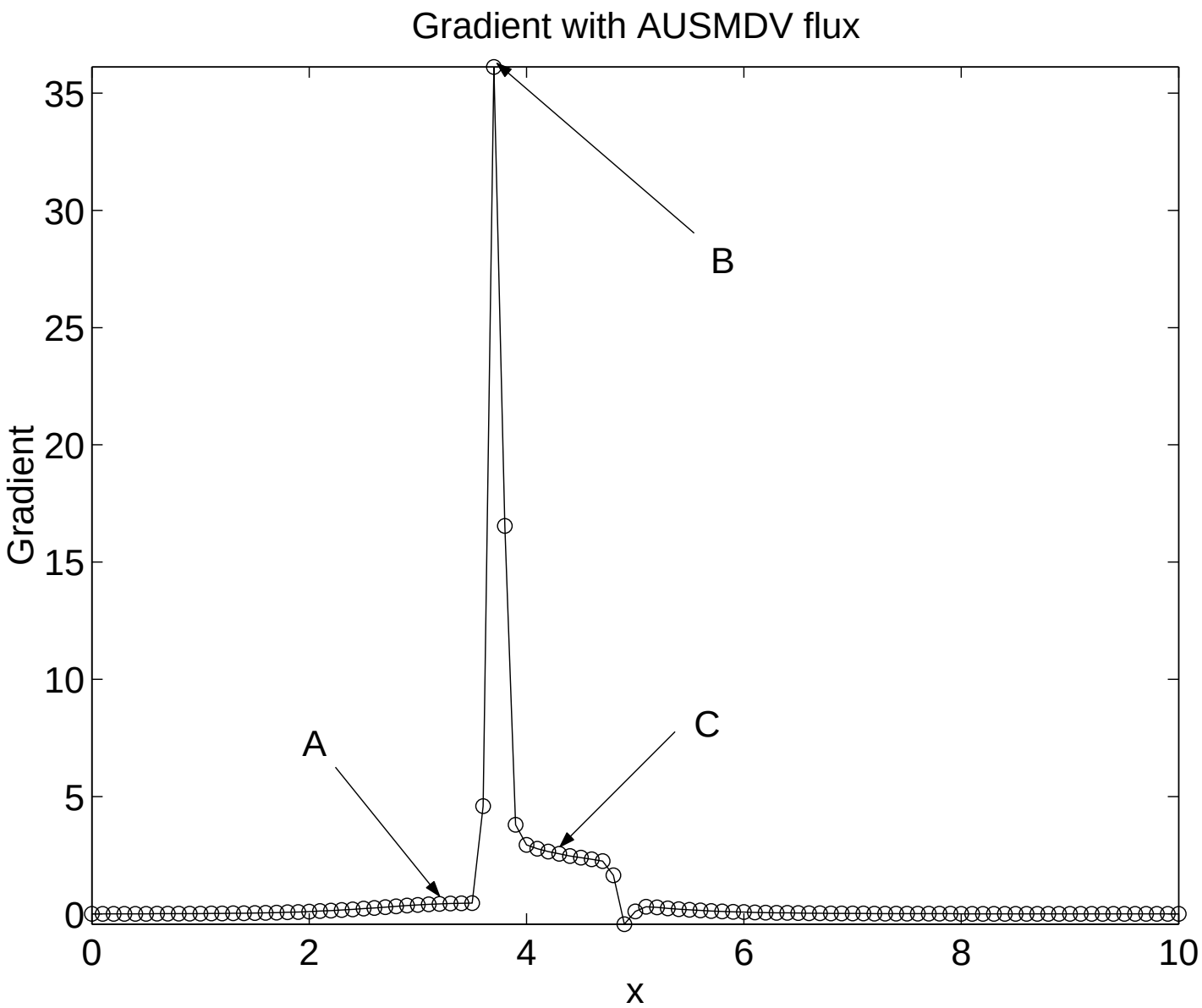
Shape gradient



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

Gradient smoothing

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

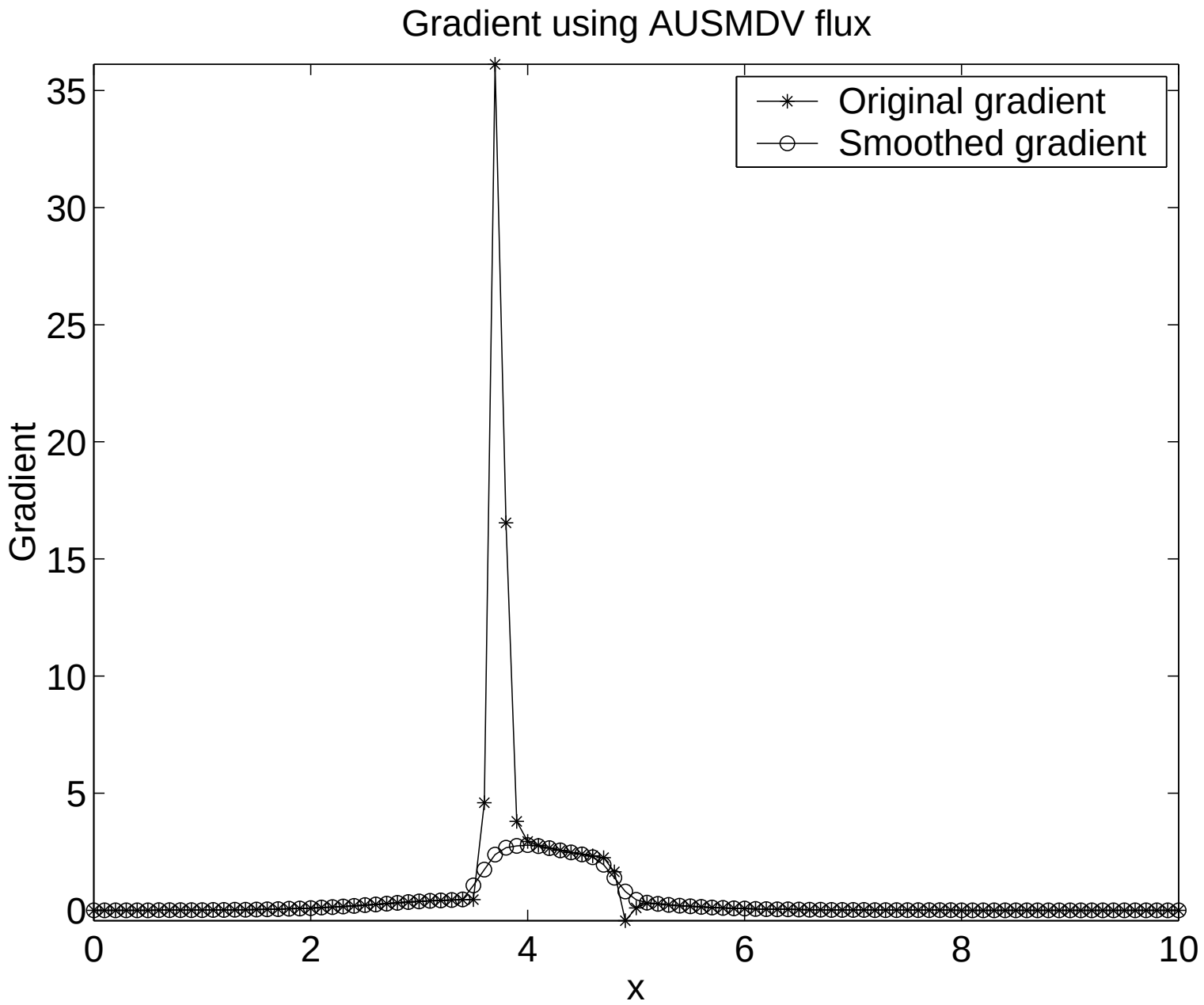
$$\left(1 - \epsilon(x)\frac{\mathrm{d}^2}{\mathrm{d}x^2}\right)\bar{G}(x) = G(x)$$

$$\epsilon_i = \{|G_{i+1} - G_i| + |G_i - G_{i-1}|\}L_i$$

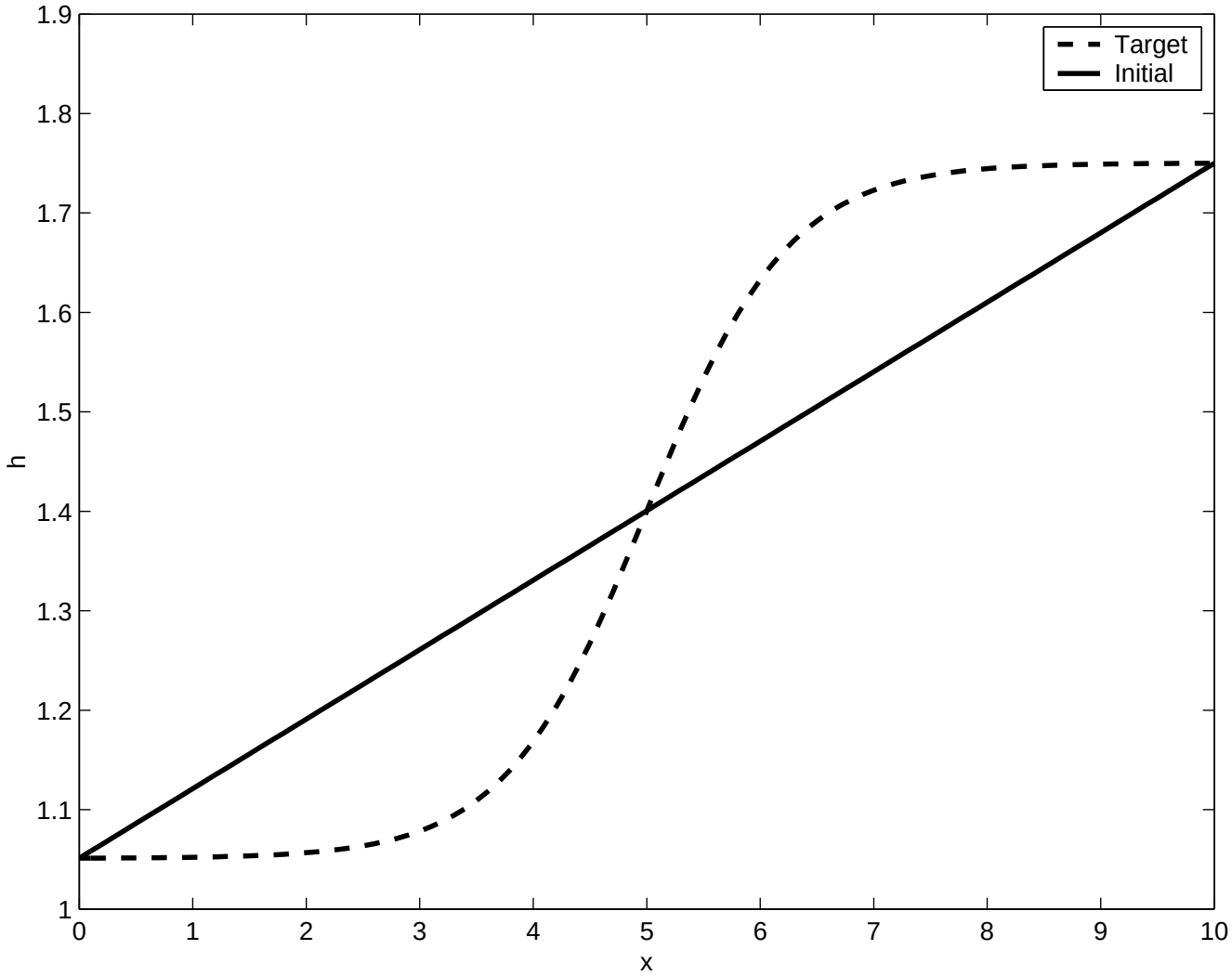
$$L_i = \frac{|G_{i+1} - 2G_i + G_{i-1}|}{\max(|G_{i+1} - G_i| + |G_i - G_{i-1}|, \mathrm{TOL})}$$

- Finite difference with Jacobi iterations

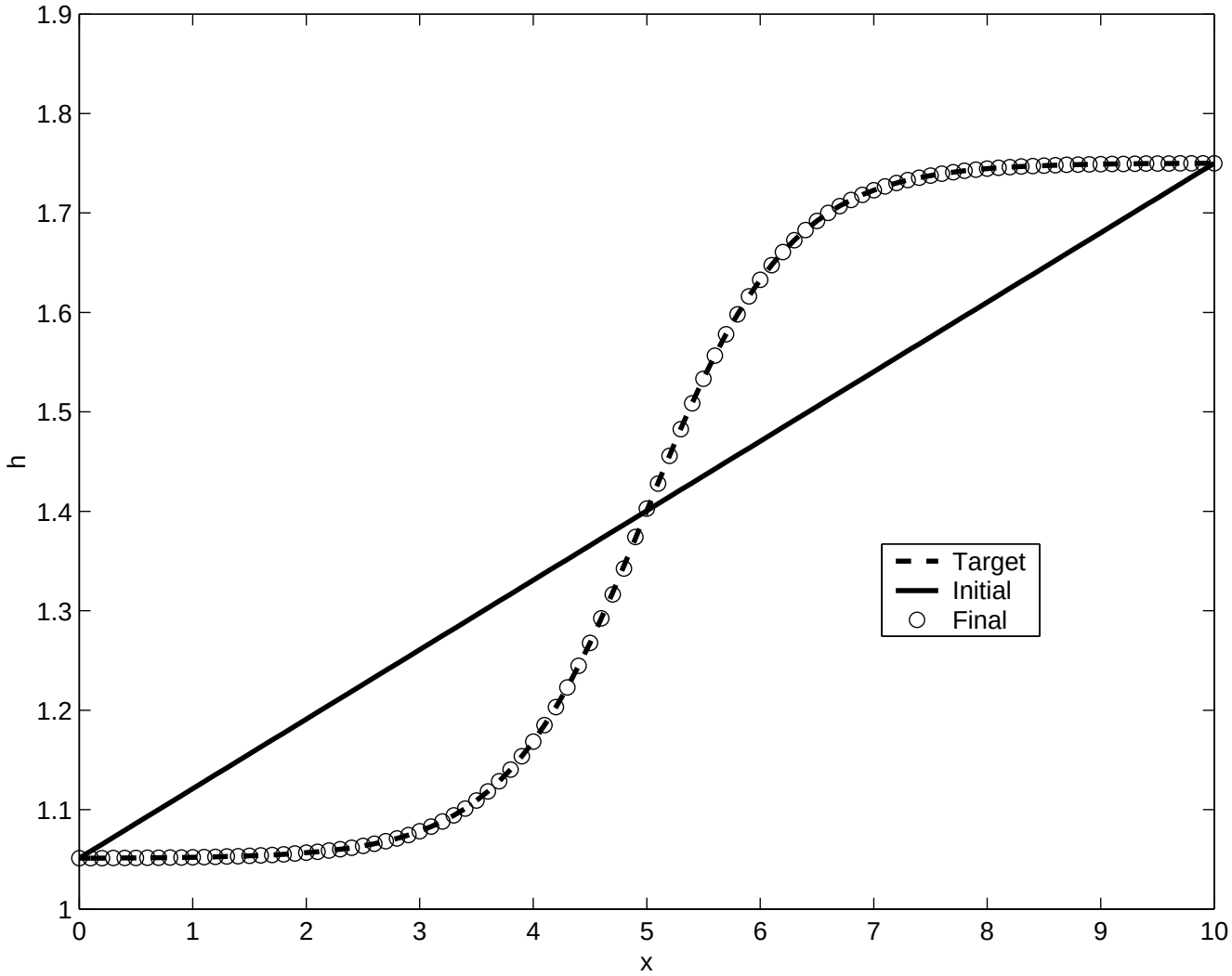
Gradient smoothing



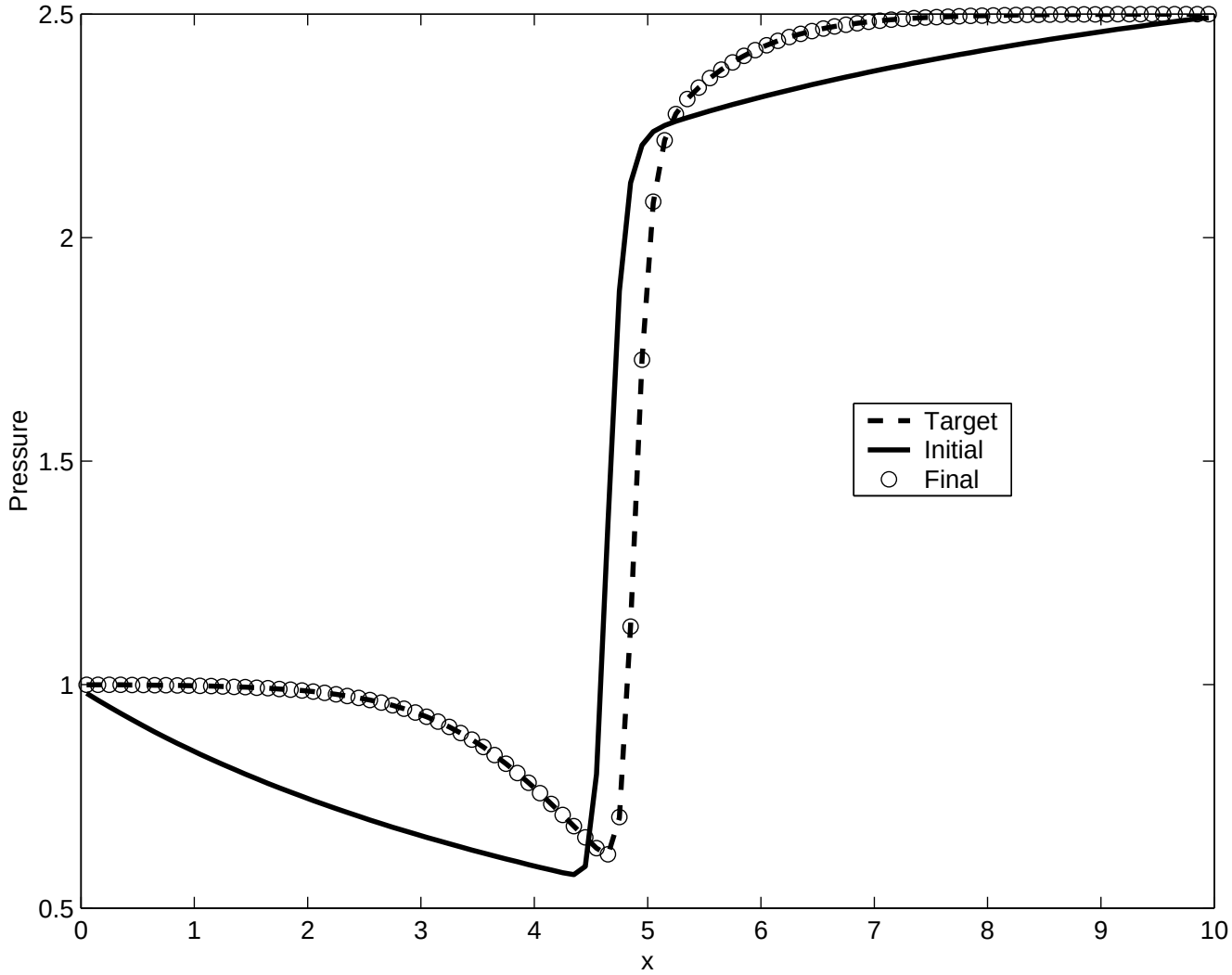
Quasi 1-D optimization: Shape



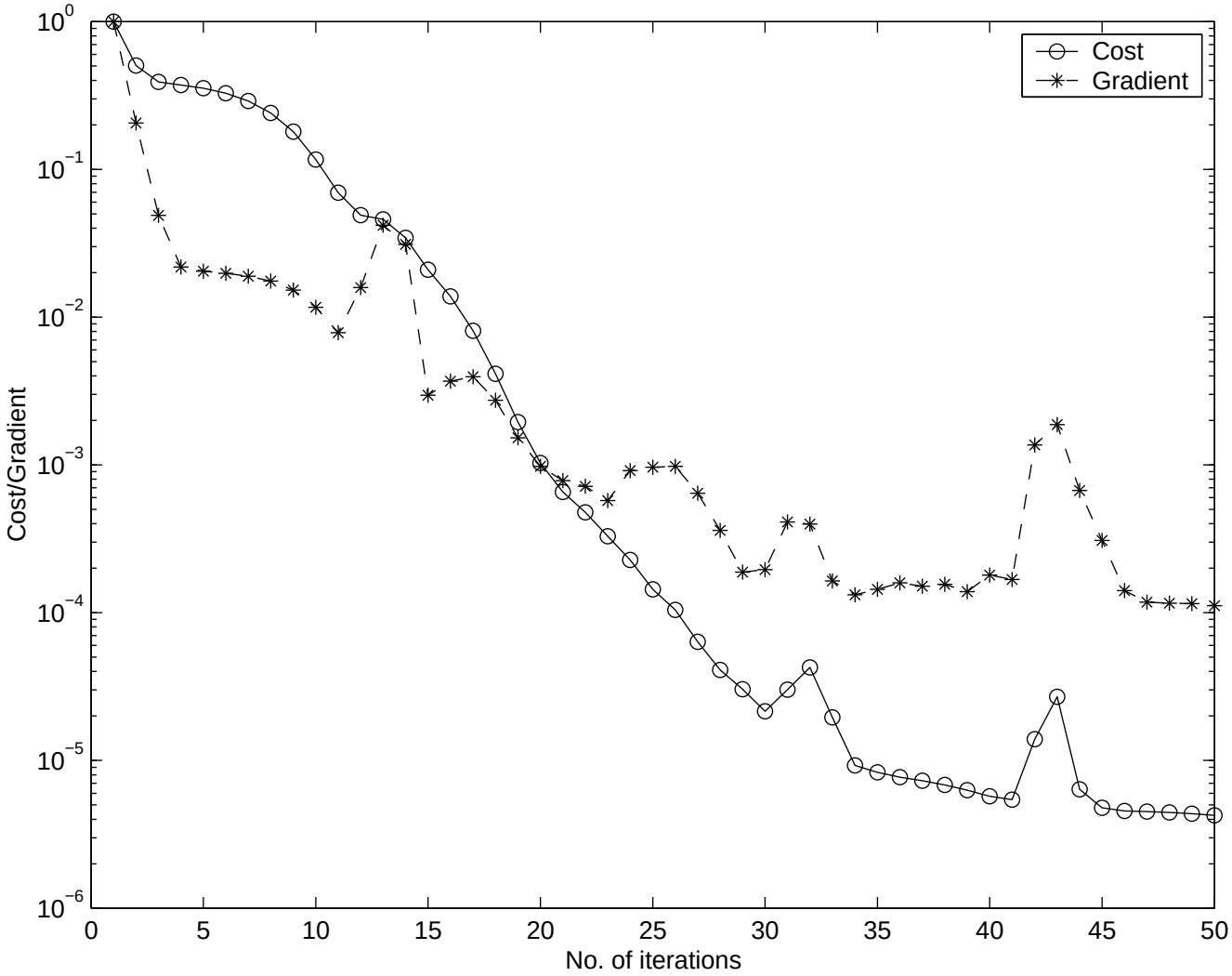
Quasi 1-D optimization: Final shape



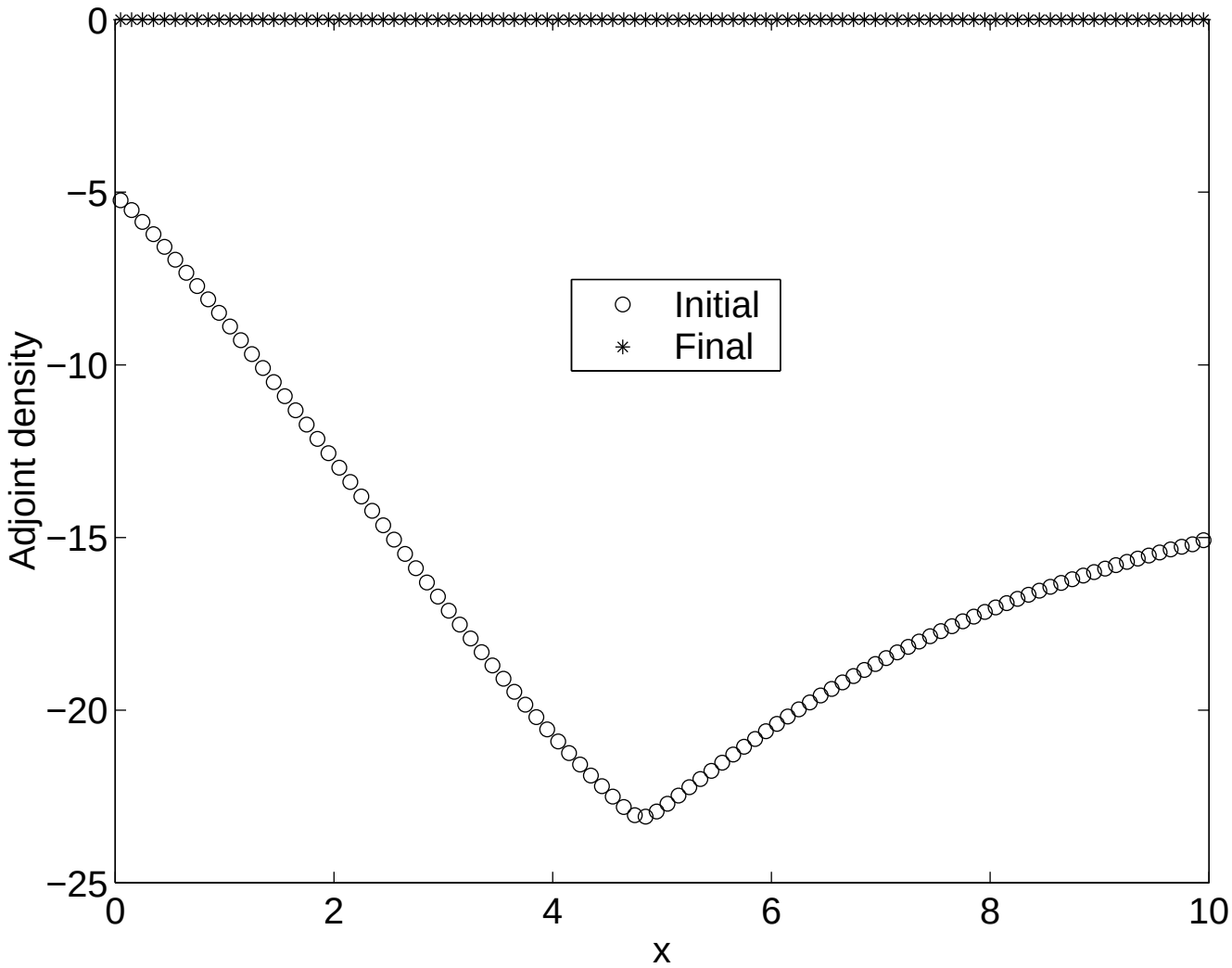
Quasi 1-D optimization: Pressure



Quasi 1-D optimization: Convergence



Quasi 1-D optimization: Adjoint density



Example codes

- 1-D example: nozzle flow (TAPENADE)
  `http://cfdlab.googlecode.com`
- 1-D example: nozzle flow (ADOLC)
  `http://cfdlab.googlecode.com`
- 2-D example: unstructured grid Euler solver (TAPENADE)
  `http://euler2d.sourceforge.net`
- 2/3-D example: structured grid Euler solver (TAPENADE)
  `http://nuwtun.berlios.de`