Adjoint approach to optimization using automatic differentiation (AD)

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

Tata Institute of Fundamental Research
Center for Applicable Mathematics
Bangalore - 560 065
http://math.tifrbng.res.in

Indo-German Workshop
IIT Madras
November 2008
Outline

1. Mathematical formulation
2. Computing gradients
3. Quasi 1-D flow
4. Gradient smoothing
5. Quasi 1-D optimization: Pressure matching
6. Example codes
Outline

1. Mathematical formulation
2. Computing gradients
3. Quasi 1-D flow
4. Gradient smoothing
5. Quasi 1-D optimization: Pressure matching
6. Example codes
Maturing of high fidelity analysis tools like Computational Fluid Dynamics (CFD) Finite Element Method (FEM)

Increase in computational power

Shift towards optimization and control

Fluid dynamics

- Design aircraft wing shape to reduce drag
- Ship hull shape optimization to reduce drag
- Minimize unsteady forces through boundary suction/blowing
- Suppress boundary layer separation
- Enhance mixing
Introduction

- Maturing of high fidelity analysis tools like Computational Fluid Dynamics (CFD) Finite Element Method (FEM)
- Increase in computational power
- Shift towards optimization and control
- Fluid dynamics
  - Design aircraft wing shape to reduce drag
  - Ship hull shape optimization to reduce drag
  - Minimize unsteady forces through boundary suction/blowing
  - Suppress boundary layer separation
  - Enhance mixing
Objectives and controls

- Objective function $J(\alpha) = J(\alpha, u)$
  mathematical representation of system performance

- Control variables $\alpha$
  - Parametric controls $\alpha \in \mathbb{R}^n$
  - Infinite dimensional controls $\alpha : X \rightarrow Y$
  - 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 \( \delta J < 0 \)

\[
\delta J = \frac{\partial J}{\partial \alpha} \delta \alpha + \frac{\partial J}{\partial u} \delta u
\]

\[
= \frac{\partial J}{\partial \alpha} \delta \alpha + \frac{\partial J}{\partial u} \frac{\partial J}{\partial \alpha} \delta \alpha
\]

\[
= \left[ \frac{\partial J}{\partial \alpha} + \frac{\partial J}{\partial u} \frac{\partial J}{\partial \alpha} \right] \delta \alpha =: G \delta \alpha
\]

- 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 \( \delta J < 0 \)

\[
\delta J = \frac{\partial J}{\partial \alpha} \delta \alpha + \frac{\partial J}{\partial u} \delta u
\]

\[
= \frac{\partial J}{\partial \alpha} \delta \alpha + \frac{\partial J}{\partial u} \frac{\partial u}{\partial \alpha} \delta \alpha
\]

\[
= \left[ \frac{\partial J}{\partial \alpha} + \frac{\partial J}{\partial u} \frac{\partial u}{\partial \alpha} \right] \delta \alpha =: G \delta \alpha
\]

• Steepest descent

\[
\delta \alpha = -\epsilon G^\top
\]

\[
\delta J = -\epsilon GG^\top = -\epsilon \|G\|^2 < 0
\]
Sensitivity approach

- Linearized state equation

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

or

\[ \frac{\partial R}{\partial u} \frac{\partial u}{\partial \alpha} = -\frac{\partial R}{\partial \alpha} \]

- Solve sensitivity equation iteratively

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

- Gradient

\[ \frac{dJ}{d\alpha} = \frac{\partial J}{\partial \alpha} + \frac{\partial J}{\partial u} \frac{\partial u}{\partial \alpha} \]
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 \( \alpha_i \)
    \[
    \frac{\partial R}{\partial u} \frac{\partial u}{\partial \alpha_i} = -\frac{\partial R}{\partial \alpha_i}
    \]
  - Compute derivative wrt \( \alpha_i \)
    \[
    \frac{\mathrm{d}J}{\mathrm{d}\alpha_i} = \frac{\partial J}{\partial \alpha_i} + \frac{\partial J}{\partial u} \frac{\partial u}{\partial \alpha_i}
    \]
- 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 \( v \)

\[ \delta J = \frac{\partial J}{\partial \alpha} \delta \alpha + \frac{\partial J}{\partial u} \delta u + v^\top \left( \frac{\partial R}{\partial \alpha} \delta \alpha + \frac{\partial R}{\partial u} \delta u \right) \]

\[ = \left( \frac{\partial J}{\partial \alpha} + v^\top \frac{\partial R}{\partial \alpha} \right) \delta \alpha + \left( \frac{\partial J}{\partial u} + v^\top \frac{\partial R}{\partial u} \right) \delta u \]

- Adjoint equation

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

- 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 $\alpha_i$

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

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

Continuous approach:
- Start with governing PDE $R(\alpha, u) = 0$
- Derive adjoint PDE and boundary conditions
- Discretize adjoint PDE and solve
- Must be re-derived whenever cost function changes
- Gradient is not consistent: discretization error

Discrete approach:
- Start with discrete approximation $R(\alpha, u) = 0$
- Derive discrete adjoint equations
- Solve discrete adjoint equations
- True gradient of discrete solution
- Can be automated using AD
Outline

1. Mathematical formulation
2. Computing gradients
3. Quasi 1-D flow
4. Gradient smoothing
5. Quasi 1-D optimization: Pressure matching
6. Example codes
Techniques for computing gradients

- Hand differentiation
- Finite difference method
- Complex variable method
- **Automatic Differentiation (AD)**
  - Computer code to compute some function
  - Chain rule of differentiation
  - Generates a code to compute derivatives
  - ADIFOR, ADOLC, ODYSEE, TAMC, TAF, TAPENADE
  - see [http://www.autodiff.org](http://www.autodiff.org)
Derivatives

Given a program $P$ computing a function $F$

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

$X \rightarrow Y$

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

- Jacobian matrix: \( J = \begin{bmatrix} \frac{\partial y_j}{\partial x_i} \end{bmatrix} \)
- Directional or tangent derivative
  \[ \dot{Y} = J \dot{X} \]
- Adjoint mode
  \[ \bar{X} = J^\top \bar{Y} \]
- Gradients \((n = 1 \text{ output})\)
  \[ J = \begin{bmatrix} \frac{\partial y}{\partial x_i} \end{bmatrix} \]
Forward differentiation

- Program $P$ is a sequence of instructions $F_k$
- $T_o = X$, given
- $k$’th line
  \[ T_k = F_k(T_{k-1}) \]
- Function is a composition
  \[ F = F_p \circ F_{p-1} \circ \ldots \circ F_1 \]
- Chain rule
  \[ \dot{Y} = F'(X)\dot{X} = F'_p(T_{p-1})F'_{p-1}(T_{p-2})\ldots F'_1(T_o)\dot{X} \]
  \[ X, \dot{X} \rightarrow Y, \dot{Y} \]
- $\text{cost}(\dot{Y}) = 4 * \text{cost}(Y)$
Differentiation: Example

- A simple example

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

- Computer code, \( f = t_{10} \)

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

\[
\begin{align*}
t_1 &= x \\
t_2 &= y \\
t_3 &= t_1 t_2 \\
t_4 &= \sin(t_1) \\
t_5 &= t_3 + t_4 \\
t_6 &= t_5 + 4 \\
t_7 &= t_2^2 \\
t_8 &= 3t_7 \\
t_9 &= t_8 + 6 \\
t_{10} &= t_6 t_9 \\
\end{align*}
\]

\[
\begin{align*}
\dot{t}_1 &= \dot{x} \\
\dot{t}_2 &= \dot{y} \\
\dot{t}_3 &= \dot{t}_1 t_2 + t_1 \dot{t}_2 \\
\dot{t}_4 &= \cos(t_1) \dot{t}_1 \\
\dot{t}_5 &= \dot{t}_3 + t_4 \\
\dot{t}_6 &= \dot{t}_5 \\
\dot{t}_7 &= 2t_2 \dot{t}_2 \\
\dot{t}_8 &= 3 \dot{t}_7 \\
\dot{t}_9 &= \dot{t}_8 \\
\dot{t}_{10} &= \dot{t}_6 t_9 + t_6 \dot{t}_9 \\
\end{align*}
\]

- \(x = 1, \dot{y} = 0, \dot{t}_{10} = \frac{\partial f}{\partial x}\) and \(\dot{x} = 0, \dot{y} = 1, \dot{t}_{10} = \frac{\partial f}{\partial y}\)

- tapenade -d -vars "x y" -outvars f costfunc.f

Praveen. C (TIFR-CAM)
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 $\mathcal{P}$ is a sequence of instructions $F_k$
- $T_o = X$, given
- $k$'th line
  \[ T_k = F_k(T_{k-1}) \]
- Function is a composition
  \[ F = F_p \circ F_{p-1} \circ \ldots \circ F_1 \]
- Chain rule
  \[ \bar{X} = [F'(X)]^\top \bar{Y} = [F'_1(T_o)]^\top [F'_2(T_1)]^\top \ldots [F'_p(T_{p-1})]^\top \bar{Y} \]
  \[ X, \bar{Y} \rightarrow \bar{X} \]
- $\text{cost}(\bar{X}) = 4 \times \text{cost}(Y)$
Differentiation: Reverse mode

- Apply chain rule of differentiation in reverse

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

- \( \bar{t}_1 = \frac{\partial f}{\partial x}, \bar{t}_2 = \frac{\partial f}{\partial y} \)

- `tapenade -b -vars "x y" -outvars f costfunc.f`
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

\[ F : \mathbb{R}^m \rightarrow \mathbb{R}^n \]

- **Direct mode**
  \[ \text{cost}(J) = m \times 4 \times \text{cost}(P) \]

- **Reverse mode**
  \[ \text{cost}(J) = n \times 4 \times \text{cost}(P) \]

- **Scalar output** \( F \in \mathbb{R}, \ n = 1 \)
  - Direct mode gives \( \nabla F \cdot \dot{X} \) for given vector \( \dot{X} \)
  - Reverse mode gives \( \nabla F \), 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
Issues in reverse AD

- Intermediate variables required in reverse order
- Some variables may be over-written
- Variables may be stored in a stack (PUSH/POP)
- Iterative solvers
  - Only final 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
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
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

\[
\text{alphab} = \frac{\partial (\text{cost})}{\partial (\text{alpha})}
\]
Solve state equation \( R(\alpha, u) = 0 \) as steady state of

\[
\frac{du}{dt} + R(\alpha, u) = 0, \quad u(0) = u_0
\]

State solver

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

Adjoint solver, hand written

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

call Residue_ba(alpha, alphab, u, res, resb)
end subroutine
```

\[ \text{resb} = \text{Adjoint variable} \]

\[ \text{ub} = \frac{\partial J}{\partial u}, \quad \text{ub1} = \left[ \frac{\partial R}{\partial u} \right]^\top \nu \]
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 \(R' \implies\) adjoint iterations stable under same condition on \(\Delta t\)

- Preconditioner for adjoint = (preconditioner for primal problem)\(^\top\)
Outline

1. Mathematical formulation
2. Computing gradients
3. Quasi 1-D flow
4. Gradient smoothing
5. Quasi 1-D optimization: Pressure matching
6. Example codes
1-D flow equations

- 1-D conservation law
  \[ \frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = 0, \quad U \in \mathbb{R}^3, \quad F(U) \in \mathbb{R}^3 \]

- Finite volume scheme
  \[ \frac{U_{i}^{n+1} - U_{i}^{n}}{\Delta t} + \frac{F_{i+1/2}^{n} - F_{i-1/2}^{n}}{\Delta x} = 0 \]

- Update equation
  \[ U_{i}^{n+1} = U_{i}^{n} - \frac{\Delta t}{\Delta x} R_{i}^{n}, \quad R_{i}^{n} = F_{i+1/2}^{n} - F_{i-1/2}^{n} \]
Discrete 1-D adjoint equations

- Finite volume residual for i’th cell, steady state
  \[ R_i := F_{i+1/2} - F_{i-1/2} = 0 \]

- Numerical flux function
  \[ F = F(X, Y), \quad F_{i+1/2} = F(U_i, U_{i+1}) \]

- Perturbation equation
  \[
  \delta R_i = \frac{\partial}{\partial X} F_{i+1/2} \delta U_i + \frac{\partial}{\partial Y} F_{i+1/2} \delta U_{i+1} \\
  - \frac{\partial}{\partial X} F_{i-1/2} \delta U_{i-1} - \frac{\partial}{\partial Y} F_{i-1/2} \delta U_i + \frac{\partial R_i}{\partial \alpha} \delta \alpha = 0
  \]

- 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 $\delta U_i$

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

• 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

```plaintext
res = 0.0
fluxinflow(u(1), res(1))
do i=1,N-1
   fluxinterior(u(i), u(i+1), res(i), res(i+1))
endo
downto
   fluxoutflow(u(N), res(N))
do i=1,N
   u(i) = u(i) - (dt/dx)*res(i)
dendo
dowhile
```
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))
endo
  fluxoutflow_b(u(N), ub2(N), res(N), v(N))
do i=1,N
    v(i) = v(i) - (dt/dx)*(ub1(i) + ub2(i))
endo
endwhile
Quasi 1-D flow

Inflow $\rightarrow$ $x$ $\rightarrow$ Outflow

$h(x)$
Quasi 1-D flow

- Quasi 1-D flow in a duct
  \[
  \frac{\partial}{\partial t}(hU) + \frac{\partial}{\partial x}(hf) = \frac{dh}{dx}P, \quad x \in (a, b) \quad t > 0
  \]
  \[
  U = \begin{bmatrix} \rho \\ \rho u \\ E \end{bmatrix}, \quad f = \begin{bmatrix} \rho u \\ p + \rho u^2 \\ (E + p)u \end{bmatrix}, \quad P = \begin{bmatrix} 0 \\ p \\ 0 \end{bmatrix}
  \]
  \[h(x) = \text{cross-section height of duct}\]

- Inverse design: find shape \(h\) to get pressure distribution \(p^*\)

- Optimization problem: find the shape \(h\) which minimizes
  \[
  J = \int_a^b (p - p^*)^2 \, dx
  \]
Quasi 1-D flow

Inflow face

Outflow face
Quasi 1-D flow

- Finite volume scheme

\[ h_i \frac{dU_i}{dt} + \frac{h_{i+1/2}F_{i+1/2} - h_{i-1/2}F_{i-1/2}}{\Delta x} = \frac{(h_{i+1/2} - h_{i-1/2})}{\Delta x} p_i \]

- Discrete cost function

\[ J = \sum_{i=1}^{N} (p_i - p^*_i)^2 \]

- Control variables

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

- \( N = 100 \)
Target pressure distribution $p^*$

![Graph showing target pressure distribution]

- AUSMDV
- KFVS
- LF
Current pressure distribution

Starting pressure

- AUSMDV
- KFVS
- LF

Pressure

x
Adjoint density

- AUSMDV
- KFVS
- LF
Convergence history: Explicit Euler

Convergence history with AUSMDV flux

Residue vs. No of iterations

Flow
Adjoint
Shape gradient
Validation of Shape gradient

Gradient with AUSMDV flux

Gradient

0 2 4 6 8 10

0 5 10 15 20 25 30 35

A

B

C

0 2 4 6 8 10

Praveen. C (TIFR-CAM) Adjoint and optimization IITM, Nov 2008
Validation of shape gradient

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

<table>
<thead>
<tr>
<th>(\Delta h)</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>0.4191069499</td>
<td>35.18452823</td>
<td>2.545316345</td>
</tr>
<tr>
<td>0.001</td>
<td>0.4231223499</td>
<td>36.10982621</td>
<td>2.556461900</td>
</tr>
<tr>
<td>0.0001</td>
<td>0.4231624999</td>
<td>36.11933154</td>
<td>2.556573499</td>
</tr>
<tr>
<td>0.00001</td>
<td>0.4231599998</td>
<td>36.11942125</td>
<td>2.556575000</td>
</tr>
<tr>
<td>0.000001</td>
<td>0.4231999994</td>
<td>36.11942305</td>
<td>2.556550001</td>
</tr>
<tr>
<td>0.0000001</td>
<td>0.4229999817</td>
<td>36.11942329</td>
<td>2.556499971</td>
</tr>
<tr>
<td>AD</td>
<td>0.4231628330</td>
<td>36.11941951</td>
<td>2.556574450</td>
</tr>
</tbody>
</table>
Outline

1. Mathematical formulation
2. Computing gradients
3. Quasi 1-D flow
4. Gradient smoothing
5. Quasi 1-D optimization: Pressure matching
6. Example codes
Gradient smoothing

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

\[
\left( 1 - \epsilon \frac{d^2}{dx^2} \right) \tilde{G} = G
\]

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

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

- Finite difference with Jacobi iterations
Gradient smoothing

Gradient using AUSMDV flux

- Original gradient
- Smoothed gradient
1. Mathematical formulation
2. Computing gradients
3. Quasi 1-D flow
4. Gradient smoothing
5. Quasi 1-D optimization: Pressure matching
6. Example codes
Quasi 1-D optimization: Shape
Quasi 1-D optimization: Final shape
Quasi 1-D optimization: Pressure
Quasi 1-D optimization: Convergence

![Graph showing convergence of cost and gradient over iterations. The graph plots the cost and gradient on a logarithmic scale against the number of iterations. The cost decreases rapidly initially, with spikes at certain iteration points, while the gradient shows a more gradual decrease.]
Quasi 1-D optimization: Adjoint density

![Graph showing adjoint density over x-axis from 0 to 10, with initial and final points indicated.](image-url)
Outline

1. Mathematical formulation
2. Computing gradients
3. Quasi 1-D flow
4. Gradient smoothing
5. Quasi 1-D optimization: Pressure matching
6. Example codes
Source transformation tool
Forward and backward mode
F77, F90, F95, C (beta as of Nov 2008)
Free
http://www-sop.inria.fr/tropics
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