UNIFORMLY CONVERGING SIMULTANEOUS TIME-STEPPING METHODS FOR OPTIMAL DESIGN

J.-D. Müller[1], P. Cusdin, A. Jaworski[2]

[1]: Dept. of Engineering, Queen Mary, University of London
[2]: The Faculty of Power and Aeronautical Engineering, Warsaw University of Technology

Cerfacs, Sparse Days, June 2006

Motivation

- Numerical optimisation is important
- Adjoint methods: (in theory) cost is independent of the number of design variables,
- AD tools make use of adjoints feasible for large scale CFD optimisation: 50^7 DOF in the primal/dual, would like to use 10^4 design variables.
- Major advances needed in reducing the cost of computing the optimal solution

Optimality Equ.
Adj. Eq.
Optimality System
Sim. Time-Stepping
gRMS
Implementation
Solver
Testcase

Motivation

- Numerical optimisation is important
- Adjoint methods: (in theory) cost is independent of the number of design variables,
- AD tools make use of adjoints feasible for large scale CFD optimisation: 50^7 DOF in the primal/dual, would like to use 10^4 design variables.
- Major advances needed in reducing the cost of computing the optimal solution

Motivation

- Numerical optimisation is important
- Adjoint methods: (in theory) cost is independent of the number of design variables,
- AD tools make use of adjoints feasible for large scale CFD optimisation: 50^7 DOF in the primal/dual, would like to use 10^4 design variables.
- Major advances needed in reducing the cost of computing the optimal solution

Motivation

- Numerical optimisation is important
- Adjoint methods: (in theory) cost is independent of the number of design variables,
- AD tools make use of adjoints feasible for large scale CFD optimisation: 50^7 DOF in the primal/dual, would like to use 10^4 design variables.
- Major advances needed in reducing the cost of computing the optimal solution

Large Scale Optimisation in CFD

- Jameson (90ies): converge primal and adjoint fully at each evaluation
- Kuruvila (1995): “one-shot” methods, fully coupled with multigrid, potential flow
- Iollo, Ta’asan (1995): “simultaneous timestepping”, iterate primal and adjoint a few steps, solve a full optimality problem exactly near the boundary
- Dervieux et. al (2004): exploring multiple update strategies
- Griewank (2002-6), “piggy-back”. Uses full ‘shifted’ Hessians.
- Hazra (2005-6), 2D Euler, one iteration each for primal adjoint and design, not uniformly convergent.

Large Scale Optimisation in CFD

- Jameson (90ies): converge primal and adjoint fully at each evaluation
- Kuruvila (1995): “one-shot” methods, fully coupled with multigrid, potential flow
- Iollo, Ta’asan (1995): “simultaneous timestepping”, iterate primal and adjoint a few steps, solve a full optimality problem exactly near the boundary
- Dervieux et. al (2004): exploring multiple update strategies
- Griewank (2002-6), “piggy-back”. Uses full ‘shifted’ Hessians.
- Hazra (2005-6), 2D Euler, one iteration each for primal adjoint and design, not uniformly convergent.

Large Scale Optimisation in CFD

- Jameson (90ies): converge primal and adjoint fully at each evaluation
- Kuruvila (1995): “one-shot” methods, fully coupled with multigrid, potential flow
- Iollo, Ta’asan (1995): “simultaneous timestepping”, iterate primal and adjoint a few steps, solve a full optimality problem exactly near the boundary
- Dervieux et. al (2004): exploring multiple update strategies
- Griewank (2002-6), “piggy-back”. Uses full ‘shifted’ Hessians.
- Hazra (2005-6), 2D Euler, one iteration each for primal adjoint and design, not uniformly convergent.

Large Scale Optimisation in CFD

- Jameson (90ies): converge primal and adjoint fully at each evaluation
- Kuruvila (1995): "one-shot" methods, fully coupled with multigrid, potential flow
- Iollo, Ta'asan (1995): "simultaneous timestepping", iterate primal and adjoint a few steps, solve a full optimality problem exactly near the boundary
- Dervieux et. al (2004): exploring multiple update strategies
- Griewank (2002-6), "piggy-back". Uses full 'shifted' Hessians.
- Hazra (2005-6), 2D Euler, one iteration each for primal adjoint and design, not uniformly convergent.

Large Scale Optimisation in CFD

- Jameson (90ies): converge primal and adjoint fully at each evaluation
- Kuruvila (1995): “one-shot” methods, fully coupled with multigrid, potential flow
- Iollo, Ta’asan (1995): “simultaneous timestepping”, iterate primal and adjoint a few steps, solve a full optimality problem exactly near the boundary
- Dervieux et. al (2004): exploring multiple update strategies
- Griewank (2002-6), “piggy-back”. Uses full ‘shifted’ Hessians.
- Hazra (2005-6), 2D Euler, one iteration each for primal adjoint and design, not uniformly convergent.

Large Scale Optimisation in CFD

- Jameson (90ies): converge primal and adjoint fully at each evaluation
- Kuruvila (1995): “one-shot” methods, fully coupled with multigrid, potential flow
- Iollo, Ta’asan (1995): “simultaneous timestepping”, iterate primal and adjoint a few steps, solve a full optimality problem exactly near the boundary
- Dervieux et. al (2004): exploring multiple update strategies
- Griewank (2002-6), “piggy-back”. Uses full ‘shifted’ Hessians.
- Hazra (2005-6), 2D Euler, one iteration each for primal adjoint and design, not uniformly convergent.

Outline

The Optimality System
The Adjoint Equations
The Optimality System
Simultaneous Time-Stepping Schemes
Adaptive Convergence Scaling

Implementation
Solver
Testcase

Outline

The Optimality System

The Adjoint Equations

The Optimality System

Simultaneous Time-Stepping Schemes

Adaptive Convergence Scaling

Implementation

Solver

Testcase

Outline

The Optimality System
The Adjoint Equations
The Optimality System
Simultaneous Time-Stepping Schemes
Adaptive Convergence Scaling

Implementation
Solver
Testcase

Tangent Linearisation

Navier Stokes equtions (steady state):

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

Linearisation with respect to a design (control) variable α

$$\frac{\partial R}{\partial U}\frac{\partial U}{\partial \alpha} = -\frac{\partial R}{\partial \alpha},$$
$$\mathbf{A}u = f.$$

Sensitivity of a cost functional L with respect to α

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} + \frac{\partial L}{\partial U}\frac{\partial U}{\partial \alpha} = \frac{\partial L}{\partial \alpha} + g^T u$$

$\frac{\partial L}{\partial \alpha}$ is directly computable, $g^T u$ requires an expensive solve for the perturbation flow field u for each α_i.

Tangent Linearisation

Navier Stokes equtions (steady state):

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

Linearisation with respect to a design (control) variable α

$$\frac{\partial R}{\partial U}\frac{\partial U}{\partial \alpha} = -\frac{\partial R}{\partial \alpha},$$
$$\mathbf{A}u = f.$$

Sensitivity of a cost functional L with respect to α

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} + \frac{\partial L}{\partial U}\frac{\partial U}{\partial \alpha} = \frac{\partial L}{\partial \alpha} + g^T u$$

$\frac{\partial L}{\partial \alpha}$ is directly computable, $g^T u$ requires an expensive solve for the perturbation flow field u for each α_i.

Optimality Equ.
Adj. Eq.
Optimality System
Sim. Time-Stepping
gRMS
Implementation
Solver
Testcase

Tangent Linearisation

Navier Stokes equtions (steady state):

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

Linearisation with respect to a design (control) variable α

$$\frac{\partial R}{\partial U}\frac{\partial U}{\partial \alpha} = -\frac{\partial R}{\partial \alpha},$$
$$\mathbf{A}u = f.$$

Sensitivity of a cost functional L with respect to α

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} + \frac{\partial L}{\partial U}\frac{\partial U}{\partial \alpha} = \frac{\partial L}{\partial \alpha} + g^T u$$

$\frac{\partial L}{\partial \alpha}$ is directly computable, $g^T u$ requires an expensive solve for the perturbation flow field u for each α_i.

Tangent Linearisation

Navier Stokes equtions (steady state):

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

Linearisation with respect to a design (control) variable α

$$\frac{\partial R}{\partial U}\frac{\partial U}{\partial \alpha} = -\frac{\partial R}{\partial \alpha},$$
$$\mathbf{A}u = f.$$

Sensitivity of a cost functional L with respect to α

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} + \frac{\partial L}{\partial U}\frac{\partial U}{\partial \alpha} = \frac{\partial L}{\partial \alpha} + g^T u$$

$\frac{\partial L}{\partial \alpha}$ is directly computable, $g^T u$ requires an expensive solve for the perturbation flow field u for each α_i.

Optimality Equ.
Adj. Eq.
Optimality System
Sim. Time-Stepping
gRMS
Implementation
Solver
Testcase

Tangent Linearisation

Navier Stokes equtions (steady state):

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

Linearisation with respect to a design (control) variable α

$$\frac{\partial R}{\partial U}\frac{\partial U}{\partial \alpha} = -\frac{\partial R}{\partial \alpha},$$
$$\mathbf{A}u = f.$$

Sensitivity of a cost functional L with respect to α

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} + \frac{\partial L}{\partial U}\frac{\partial U}{\partial \alpha} = \frac{\partial L}{\partial \alpha} + g^T u$$

$\frac{\partial L}{\partial \alpha}$ is directly computable, $g^T u$ requires an expensive solve for the perturbation flow field u for each α_i.

The Adjoint Equations

$$\left(\frac{\partial R}{\partial U}\right)^T v = \left(\frac{\partial L}{\partial U}\right)^T,$$
$$\mathbf{A}^T v = g.$$

From this follows the *Adjoint Equivalence*

$$g^T u = (\mathbf{A}^T v)^T u = v^T \mathbf{A} u = v^T f$$

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} + g^T u = \frac{\partial L}{\partial \alpha} + v^T f$$

Using $v^T f$, needs a single solve of $\mathbf{A}^T v = g$ and the evaluation of f_i for each α_i.

The Adjoint Equations

$$\left(\frac{\partial R}{\partial U}\right)^T v = \left(\frac{\partial L}{\partial U}\right)^T,$$
$$\mathbf{A}^T v = g.$$

From this follows the *Adjoint Equivalence*

$$g^T u = (\mathbf{A}^T v)^T u = v^T \mathbf{A} u = v^T f$$

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} + g^T u = \frac{\partial L}{\partial \alpha} + v^T f$$

Using $v^T f$, needs a single solve of $\mathbf{A}^T v = g$ and the evaluation of f_i for each α_i.

The Adjoint Equations

$$\left(\frac{\partial R}{\partial U}\right)^T v = \left(\frac{\partial L}{\partial U}\right)^T,$$
$$\mathbf{A}^T v = g.$$

From this follows the *Adjoint Equivalence*

$$g^T u = (\mathbf{A}^T v)^T u = v^T \mathbf{A} u = v^T f$$

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} + g^T u = \frac{\partial L}{\partial \alpha} + v^T f$$

Using $v^T f$, needs a single solve of $\mathbf{A}^T v = g$ and the evaluation of f_i for each α_i.

The Optimality System

$$\mathbf{R}(Q,\alpha) = 0 \qquad \text{(Navier-Stokes)}$$

$$\mathbf{A}^T v = g \qquad \text{(Adjoint)}$$

$$\frac{\partial L}{\partial \alpha} + v^T f = 0 \qquad \text{(Design)}$$

The Optimality System

$$\mathbf{R}(Q,\alpha) = 0 \qquad \text{(Navier-Stokes)}$$

$$\mathbf{A}^T v = g \qquad \text{(Adjoint)}$$

$$\frac{\partial L}{\partial \alpha} + v^T f = 0 \qquad \text{(Design)}$$

Optimality Equ.
Adj. Eq.
Optimality System
Sim. Time-Stepping
gRMS
Implementation
Solver
Testcase

The Optimality System

$$
\begin{aligned}
\mathbf{R}(Q,\alpha) &= 0 && \text{(Navier-Stokes)}\\
\mathbf{A}^T v &= g && \text{(Adjoint)}\\
\frac{\partial L}{\partial \alpha} + v^T f &= 0 && \text{(Design)}
\end{aligned}
$$

The Optimality System with Time-Stepping

Optimality Equ.
Adj. Eq.
Optimality System
Sim. Time-Stepping
gRMS
Implementation
Solver
Testcase

$$\frac{\partial Q}{\partial t} = -\mathbf{R}(Q,\alpha) \quad \text{(Navier-Stokes)}$$

$$\frac{\partial v}{\partial t} = g - \mathbf{A}^T v \quad \text{(Adjoint)}$$

$$\frac{\partial \alpha}{\partial t} = -\frac{\partial L}{\partial \alpha} - v^T f \quad \text{(Design)}$$

The Optimality System with Time-Stepping

$$\frac{\partial Q}{\partial t} = -\mathbf{R}(Q, \alpha) \quad \text{(Navier-Stokes)}$$

$$\frac{\partial v}{\partial t} = g - \mathbf{A}^T v \quad \text{(Adjoint)}$$

$$\frac{\partial \alpha}{\partial t} = -\frac{\partial L}{\partial \alpha} - v^T f \quad \text{(Design)}$$

The Optimality System with Time-Stepping

$$\frac{\partial Q}{\partial t} = - \mathbf{R}(Q, \alpha) \quad \text{(Navier-Stokes)}$$

$$\frac{\partial v}{\partial t} = g - \mathbf{A}^T v \quad \text{(Adjoint)}$$

$$\frac{\partial \alpha}{\partial t} = - \frac{\partial L}{\partial \alpha} - v^T f \quad \text{(Design)}$$

Simultaneous Time-Stepping Schemes: Block Gauß-Seidel

1. march the primal to sufficient convergence
2. march the dual to sufficient convergence
3. compute the gradient
4. stop if primal and dual are fully converged and the gradient is below an acceptable threshold
5. approximate the Hessian
6. choose a descent direction and step size
7. update the design
8. go to 1.

Simultaneous Time-Stepping Schemes: Block Gauß-Seidel

1. march the primal to sufficient convergence
2. march the dual to sufficient convergence
3. compute the gradient
4. stop if primal and dual are fully converged and the gradient is below an acceptable threshold
5. approximate the Hessian
6. choose a descent direction and step size
7. update the design
8. go to 1.

Simultaneous Time-Stepping Schemes: Block Gauß-Seidel

1. march the primal to sufficient convergence
2. march the dual to sufficient convergence
3. compute the gradient
4. stop if primal and dual are fully converged and the gradient is below an acceptable threshold
5. approximate the Hessian
6. choose a descent direction and step size
7. update the design
8. go to 1.

Simultaneous Time-Stepping Schemes: Block Gauß-Seidel

1. march the primal to sufficient convergence
2. march the dual to sufficient convergence
3. compute the gradient
4. stop if primal and dual are fully converged and the gradient is below an acceptable threshold
5. approximate the Hessian
6. choose a descent direction and step size
7. update the design
8. go to 1.

Simultaneous Time-Stepping Schemes: Block Gauß-Seidel

1. march the primal to sufficient convergence
2. march the dual to sufficient convergence
3. compute the gradient
4. stop if primal and dual are fully converged and the gradient is below an acceptable threshold
5. approximate the Hessian
6. choose a descent direction and step size
7. update the design
8. go to 1.

Simultaneous Time-Stepping Schemes: Block Gauß-Seidel

1. march the primal to sufficient convergence
2. march the dual to sufficient convergence
3. compute the gradient
4. stop if primal and dual are fully converged and the gradient is below an acceptable threshold
5. approximate the Hessian
6. choose a descent direction and step size
7. update the design
8. go to 1.

Simultaneous Time-Stepping Schemes: Block Gauß-Seidel

1. march the primal to sufficient convergence
2. march the dual to sufficient convergence
3. compute the gradient
4. stop if primal and dual are fully converged and the gradient is below an acceptable threshold
5. approximate the Hessian
6. choose a descent direction and step size
7. update the design
8. go to 1.

Simultaneous Time-Stepping Schemes: Block Gauß-Seidel

1. march the primal to sufficient convergence
2. march the dual to sufficient convergence
3. compute the gradient
4. stop if primal and dual are fully converged and the gradient is below an acceptable threshold
5. approximate the Hessian
6. choose a descent direction and step size
7. update the design
8. go to 1.

Simultaneous Time-Stepping Schemes: Block Gauß-Seidel

1. march the primal to sufficient convergence
2. march the dual to sufficient convergence
3. compute the gradient
4. stop if primal and dual are fully converged and the gradient is below an acceptable threshold
5. approximate the Hessian
6. choose a descent direction and step size
7. update the design
8. go to 1.

Simultaneous Time-Stepping Schemes

The algorithms differ in

- how to sequence the block solves
- how to determine a suitable level of convergence for the state and dual,
- what algorithm to use to update the design and
- what step-size to choose for the design.

Simultaneous Time-Stepping Schemes

The algorithms differ in

- how to sequence the block solves
- how to determine a suitable level of convergence for the state and dual,
- what algorithm to use to update the design and
- what step-size to choose for the design.

Simultaneous Time-Stepping Schemes

The algorithms differ in

- how to sequence the block solves
- how to determine a suitable level of convergence for the state and dual,
- what algorithm to use to update the design and
- what step-size to choose for the design.

Simultaneous Time-Stepping Schemes

The algorithms differ in

- how to sequence the block solves
- how to determine a suitable level of convergence for the state and dual,
- what algorithm to use to update the design and
- what step-size to choose for the design.

Simultaneous Time-Stepping Schemes

The algorithms differ in

- how to sequence the block solves
- how to determine a suitable level of convergence for the state and dual,
- what algorithm to use to update the design and
- what step-size to choose for the design.

Adaptive Convergence Scaling

- Computing the gradient using $v^T f$ is of the cost of one iteration when using adjoint mesh sensititvities
- Time-steps are very small in compressible explicit codes
- Not effective to recompute the gradients at every iteration
- Here: at each evaluation scale convergence of primal and adjoint with gradient magnitude, subject to minimum number of solves

$$gRMS = \frac{|\nabla f|}{C}$$

- Functional and gradient are computed with increasing precision as the system converges uniformly

Adaptive Convergence Scaling

- Computing the gradient using $v^T f$ is of the cost of one iteration when using adjoint mesh sensititvities
- Time-steps are very small in compressible explicit codes
- Not effective to recompute the gradients at every iteration
- Here: at each evaluation scale convergence of primal and adjoint with gradient magnitude, subject to minimum number of solves

$$gRMS = \frac{|\nabla f|}{C}$$

- Functional and gradient are computed with increasing precision as the system converges uniformly

Adaptive Convergence Scaling

- Computing the gradient using $v^T f$ is of the cost of one iteration when using adjoint mesh sensititvities
- Time-steps are very small in compressible explicit codes
- Not effective to recompute the gradients at every iteration
- Here: at each evaluation scale convergence of primal and adjoint with gradient magnitude, subject to minimum number of solves

$$gRMS = \frac{|\nabla f|}{C}$$

- Functional and gradient are computed with increasing precision as the system converges uniformly

Adaptive Convergence Scaling

- Computing the gradient using $v^T f$ is of the cost of one iteration when using adjoint mesh sensititvities
- Time-steps are very small in compressible explicit codes
- Not effective to recompute the gradients at every iteration
- Here: at each evaluation scale convergence of primal and adjoint with gradient magnitude, subject to minimum number of solves

$$gRMS = \frac{|\nabla f|}{C}$$

- Functional and gradient are computed with increasing precision as the system converges uniformly

Adaptive Convergence Scaling

- Computing the gradient using $v^T f$ is of the cost of one iteration when using adjoint mesh sensititvities
- Time-steps are very small in compressible explicit codes
- Not effective to recompute the gradients at every iteration
- Here: at each evaluation scale convergence of primal and adjoint with gradient magnitude, subject to minimum number of solves

$$gRMS = \frac{|\nabla f|}{C}$$

- Functional and gradient are computed with increasing precision as the system converges uniformly

Adaptive Convergence Scaling

- Computing the gradient using $v^T f$ is of the cost of one iteration when using adjoint mesh sensititvities
- Time-steps are very small in compressible explicit codes
- Not effective to recompute the gradients at every iteration
- Here: at each evaluation scale convergence of primal and adjoint with gradient magnitude, subject to minimum number of solves

$$gRMS = \frac{|\nabla f|}{C}$$

- Functional and gradient are computed with increasing precision as the system converges uniformly

Outline

The Optimality System
- The Adjoint Equations
- The Optimality System
- Simultaneous Time-Stepping Schemes
- Adaptive Convergence Scaling

Implementation
- Solver
- Testcase

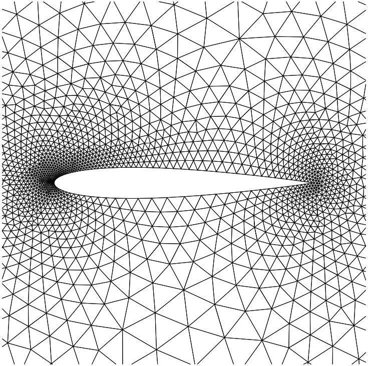
Adjoint Solvers using Automatic Differentation

- Explicit unstructured multi-grid finite volume solver, Roe-FDS.
- Discrete adjoint codes through automatic differentation
- Reorganisation of the primal code to obtain efficient adjoints: factor 1.3 in memory and CPU time
- Every surface meshpoint can be a design variable - toward multi-gridding of the design
- Optimisation using quasi-Newton methods (LBFGS)
- Aim is to develop fast optimisation methods for large steady (and unsteady) systems.

Adjoint Solvers using Automatic Differentation

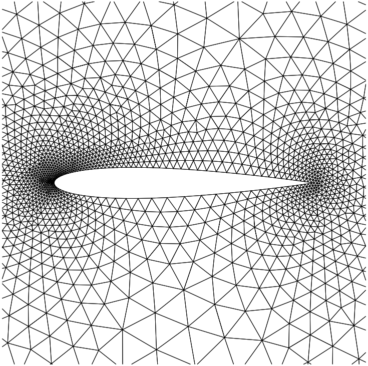
- Explicit unstructured multi-grid finite volume solver, Roe-FDS.
- Discrete adjoint codes through automatic differentation
- Reorganisation of the primal code to obtain efficient adjoints: factor 1.3 in memory and CPU time
- Every surface meshpoint can be a design variable - toward multi-gridding of the design
- Optimisation using quasi-Newton methods (LBFGS)
- Aim is to develop fast optimisation methods for large steady (and unsteady) systems.

Testcase: Inverse Design

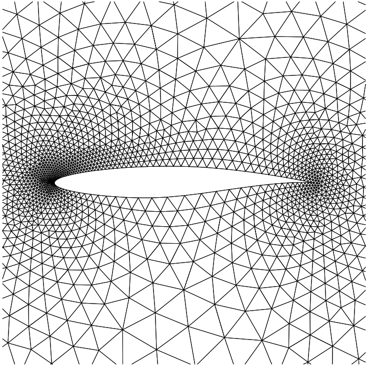


NACA 0012

Testcase: Inverse Design



NACA 0012



RAE 2822

Testcase Parameters

- subsonic, Ma=.43
- angle of attack: 0°
- 4500 triangles, 95 wall nodes
- convergence of adjoint and primal to machine precision in 200 cyc to 10^{-10}.
- all 95 wall nodes are design variables, except for fixed L.E and T.E.

Boundary Smoothing

- All boundary nodes can be design variables, or alternatively a fraction of them
- We seek a shape mode that allows multigrid restriction and prolongation
- Artificial Dissipation needs to be added to the design discretisation to dampen oscillations
- Jameson, Mohammadi, Dervieux: Implicit 'Sobolev' Smoothing on the gradient
- Linear boundary smoothing with a number of point-Jacobi iterations:

$$\alpha_i^{n+1} = \alpha_i^n - \frac{\beta}{2}\left(\alpha_{i-1}^n - 2\alpha_i^n + \alpha_{i+1}^n\right),$$

- Annihilation of the highest frequencies in one iteration with $\beta = 1/2$.
- Number of iter needs to be chosen.

Boundary Smoothing

- All boundary nodes can be design variables, or alternatively a fraction of them
- We seek a shape mode that allows multigrid restriction and prolongation
- Artificial Dissipation needs to be added to the design discretisation to dampen oscillations
- Jameson, Mohammadi, Dervieux: Implicit 'Sobolev' Smoothing on the gradient
- Linear boundary smoothing with a number of point-Jacobi iterations:

$$\alpha_i^{n+1} = \alpha_i^n - \frac{\beta}{2}\left(\alpha_{i-1}^n - 2\alpha_i^n + \alpha_{i+1}^n\right),$$

- Annihilation of the highest frequencies in one iteration with $\beta = 1/2$.
- Number of iter needs to be chosen.

Boundary Smoothing

- All boundary nodes can be design variables, or alternatively a fraction of them
- We seek a shape mode that allows multigrid restriction and prolongation
- Artificial Dissipation needs to be added to the design discretisation to dampen oscillations
- Jameson, Mohammadi, Dervieux: Implicit 'Sobolev' Smoothing on the gradient
- Linear boundary smoothing with a number of point-Jacobi iterations:

$$\alpha_i^{n+1} = \alpha_i^n - \frac{\beta}{2}\left(\alpha_{i-1}^n - 2\alpha_i^n + \alpha_{i+1}^n\right),$$

- Annihilation of the highest frequencies in one iteration with $\beta = 1/2$.
- Number of iter needs to be chosen.

Boundary Smoothing

- All boundary nodes can be design variables, or alternatively a fraction of them
- We seek a shape mode that allows multigrid restriction and prolongation
- Artificial Dissipation needs to be added to the design discretisation to dampen oscillations
- Jameson, Mohammadi, Dervieux: Implicit 'Sobolev' Smoothing on the gradient
- Linear boundary smoothing with a number of point-Jacobi iterations:

$$\alpha_i^{n+1} = \alpha_i^n - \frac{\beta}{2}\left(\alpha_{i-1}^n - 2\alpha_i^n + \alpha_{i+1}^n\right),$$

- Annihilation of the highest frequencies in one iteration with $\beta = 1/2$.
- Number of iter needs to be chosen.

Boundary Smoothing

- All boundary nodes can be design variables, or alternatively a fraction of them
- We seek a shape mode that allows multigrid restriction and prolongation
- Artificial Dissipation needs to be added to the design discretisation to dampen oscillations
- Jameson, Mohammadi, Dervieux: Implicit 'Sobolev' Smoothing on the gradient
- Linear boundary smoothing with a number of point-Jacobi iterations:

$$\alpha_i^{n+1} = \alpha_i^n - \frac{\beta}{2}\left(\alpha_{i-1}^n - 2\alpha_i^n + \alpha_{i+1}^n\right),$$

- Annihilation of the highest frequencies in one iteration with $\beta = 1/2$.
- Number of iter needs to be chosen.

Boundary Smoothing

- All boundary nodes can be design variables, or alternatively a fraction of them
- We seek a shape mode that allows multigrid restriction and prolongation
- Artificial Dissipation needs to be added to the design discretisation to dampen oscillations
- Jameson, Mohammadi, Dervieux: Implicit 'Sobolev' Smoothing on the gradient
- Linear boundary smoothing with a number of point-Jacobi iterations:

$$\alpha_i^{n+1} = \alpha_i^n - \frac{\beta}{2}\left(\alpha_{i-1}^n - 2\alpha_i^n + \alpha_{i+1}^n\right),$$

- Annihilation of the highest frequencies in one iteration with $\beta = 1/2$.
- Number of iter needs to be chosen.

Boundary Smoothing

- All boundary nodes can be design variables, or alternatively a fraction of them
- We seek a shape mode that allows multigrid restriction and prolongation
- Artificial Dissipation needs to be added to the design discretisation to dampen oscillations
- Jameson, Mohammadi, Dervieux: Implicit 'Sobolev' Smoothing on the gradient
- Linear boundary smoothing with a number of point-Jacobi iterations:

$$\alpha_i^{n+1} = \alpha_i^n - \frac{\beta}{2}\left(\alpha_{i-1}^n - 2\alpha_i^n + \alpha_{i+1}^n\right),$$

- Annihilation of the highest frequencies in one iteration with $\beta = 1/2$.
- Number of iter needs to be chosen.

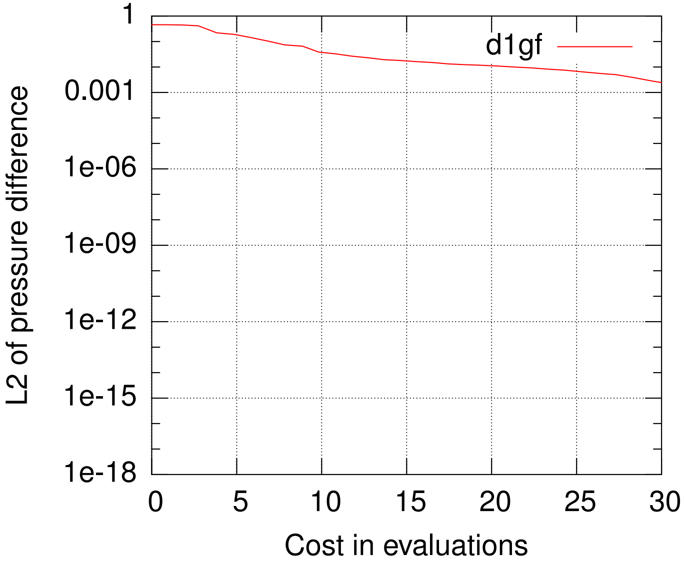
Boundary Smoothing

- All boundary nodes can be design variables, or alternatively a fraction of them
- We seek a shape mode that allows multigrid restriction and prolongation
- Artificial Dissipation needs to be added to the design discretisation to dampen oscillations
- Jameson, Mohammadi, Dervieux: Implicit 'Sobolev' Smoothing on the gradient
- Linear boundary smoothing with a number of point-Jacobi iterations:

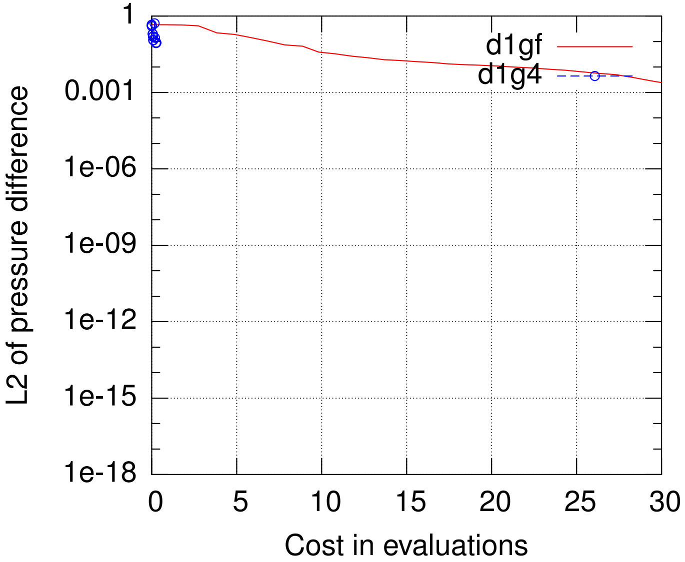
$$\alpha_i^{n+1} = \alpha_i^n - \frac{\beta}{2}\left(\alpha_{i-1}^n - 2\alpha_i^n + \alpha_{i+1}^n\right),$$

- Annihilation of the highest frequencies in one iteration with $\beta = 1/2$.
- Number of iter needs to be chosen.

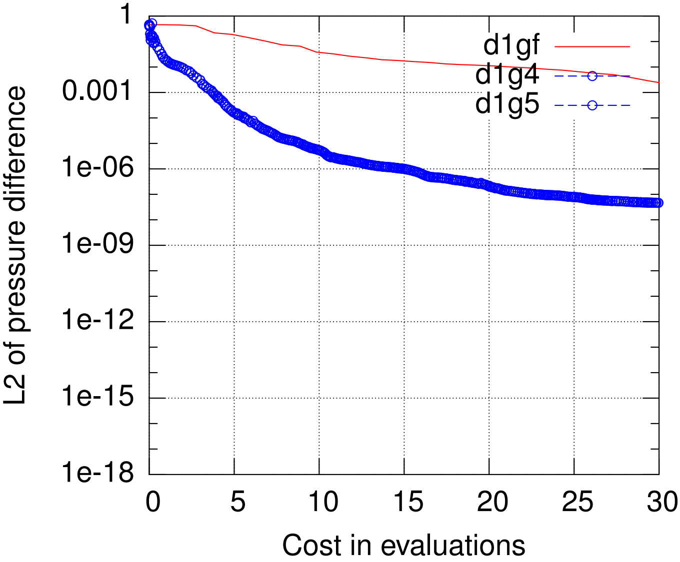
Explicit Smoothing



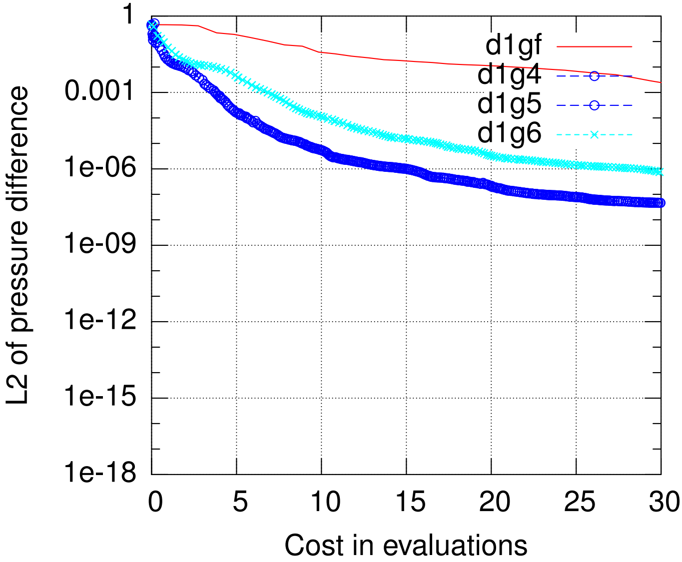
Explicit Smoothing



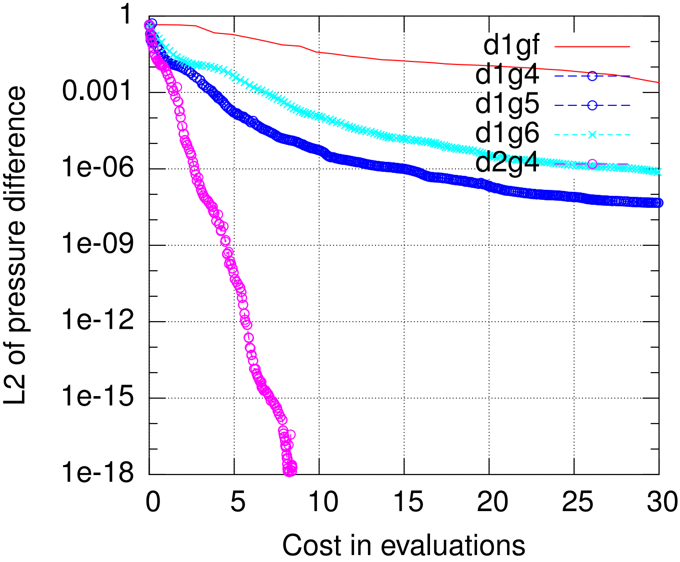
Explicit Smoothing



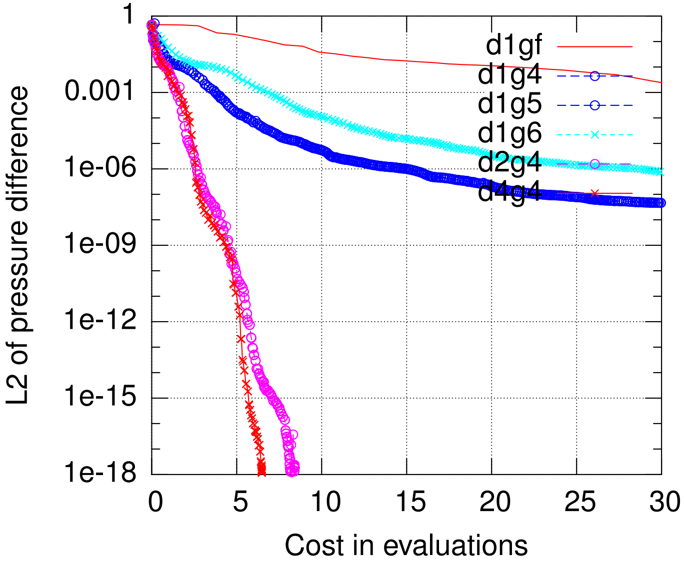
Explicit Smoothing



Explicit Smoothing

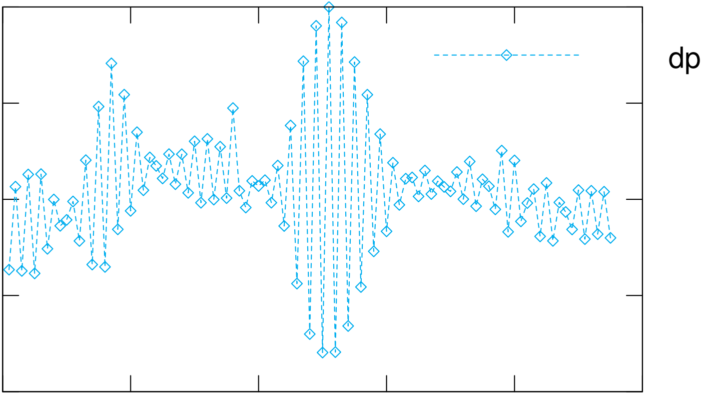


Explicit Smoothing



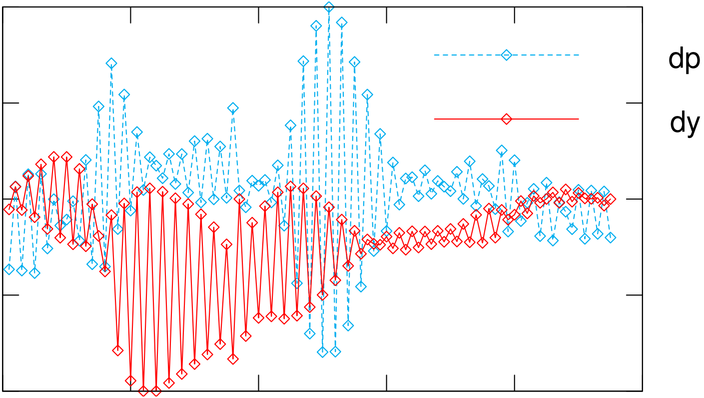
Oscillatory Solution

At (stalled) convergence, the shape, functional and gradients are highly oscillatory



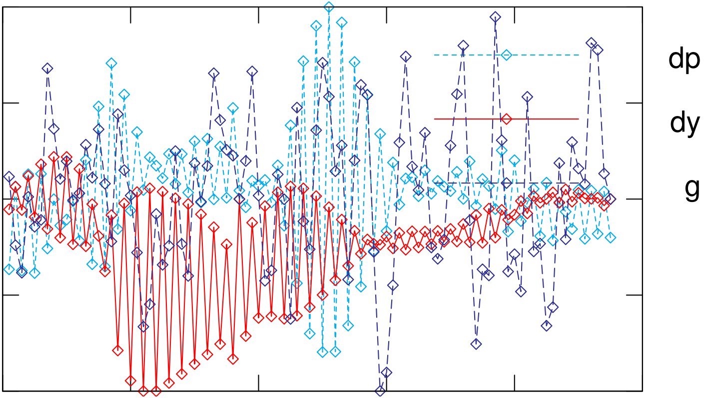
Oscillatory Solution

At (stalled) convergence, the shape, functional and gradients are highly oscillatory



Oscillatory Solution

At (stalled) convergence, the shape, functional and gradients are highly oscillatory



Smoothing Alternatives

The algorithm is block diagonal

Smoothing could be applied with any block solve

- on the shape displacement
- on the design variables
- on the gradients
- on the (pointwise) functional

Smoothing Alternatives

The algorithm is block diagonal

Smoothing could be applied with any block solve

- on the shape displacement
- on the design variables
- on the gradients
- on the (pointwise) functional

Smoothing Alternatives

The algorithm is block diagonal

Smoothing could be applied with any block solve

- on the shape displacement
- on the design variables
- on the gradients
- on the (pointwise) functional

Optimality Equ.
Adj. Eq.
Optimality System
Sim. Time-Stepping
gRMS
Implementation
Solver
Testcase

Smoothing Alternatives

The algorithm is block diagonal

Smoothing could be applied with any block solve

- on the shape displacement
- on the design variables
- on the gradients
- on the (pointwise) functional

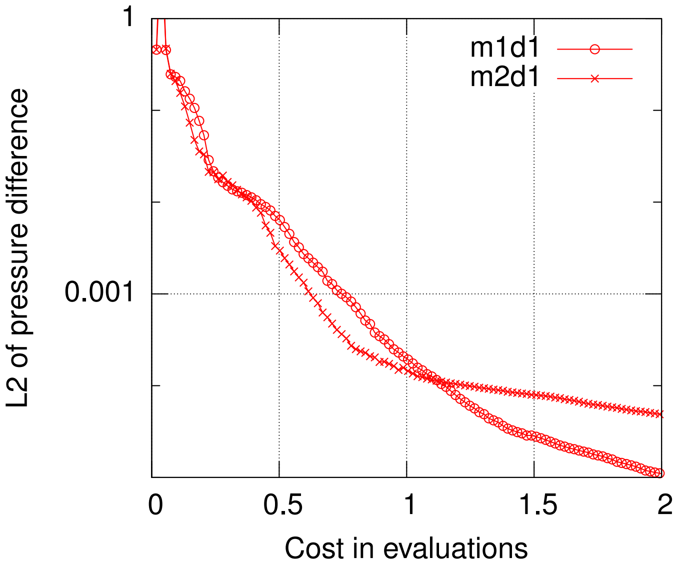
Smoothing Alternatives

The algorithm is block diagonal

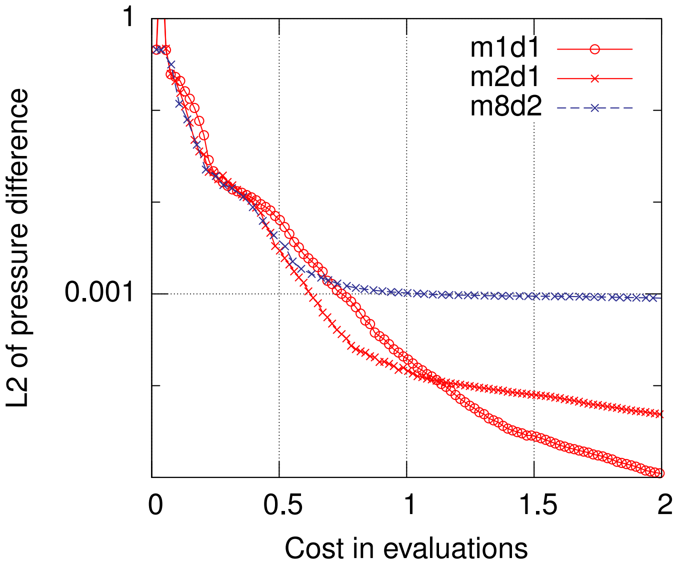
Smoothing could be applied with any block solve

- on the shape displacement
- on the design variables
- on the gradients
- on the (pointwise) functional

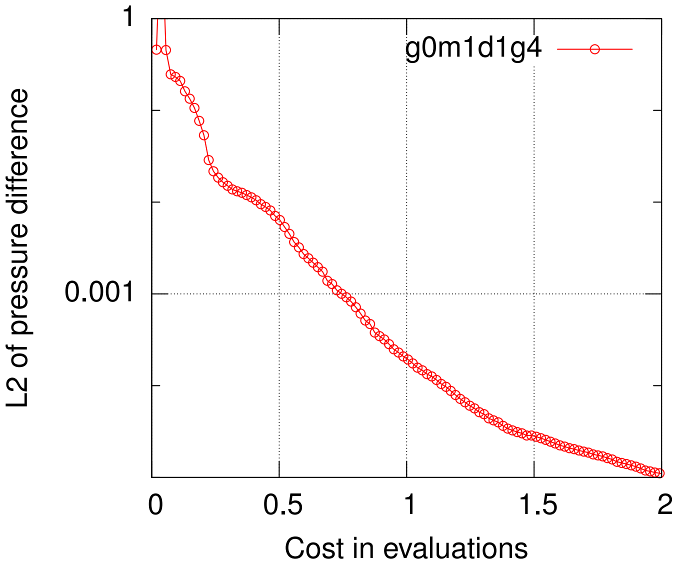
Smoothing the Displacement



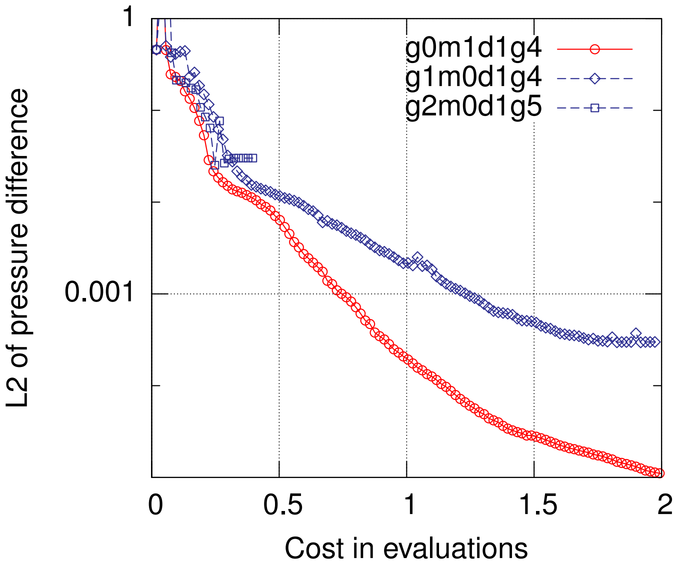
Smoothing the Displacement



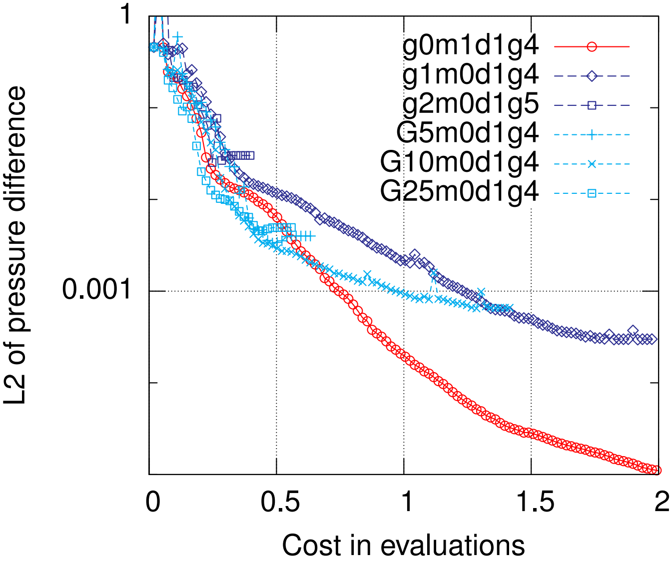
Smoothing the gradients



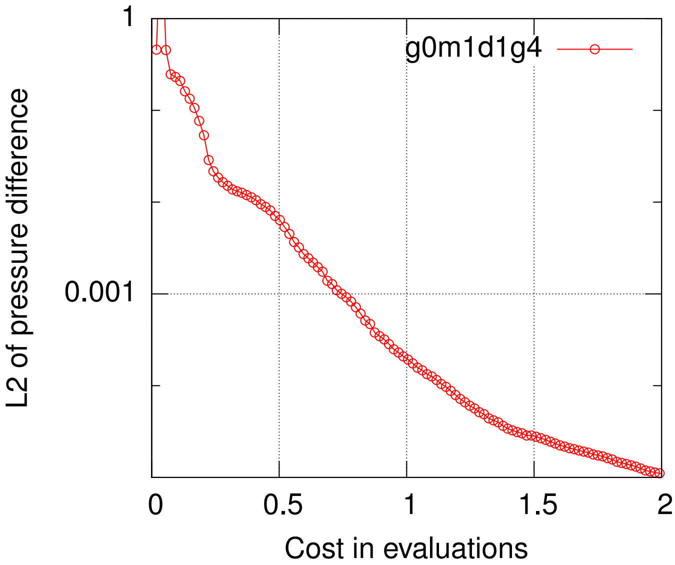
Smoothing the gradients



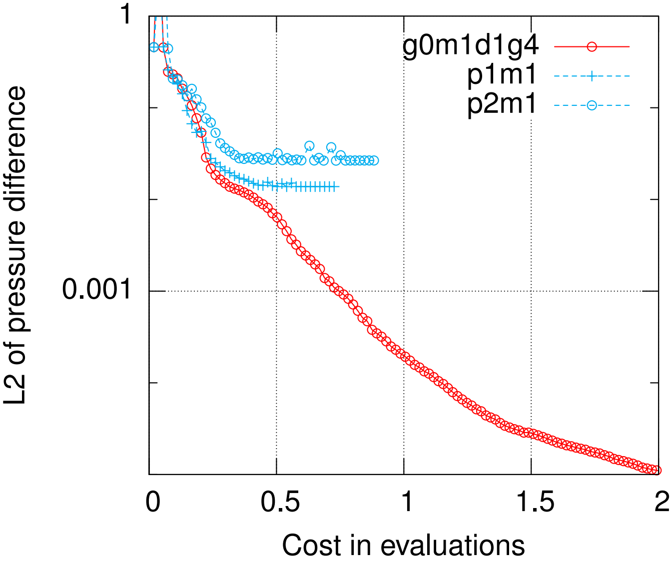
Smoothing the gradients



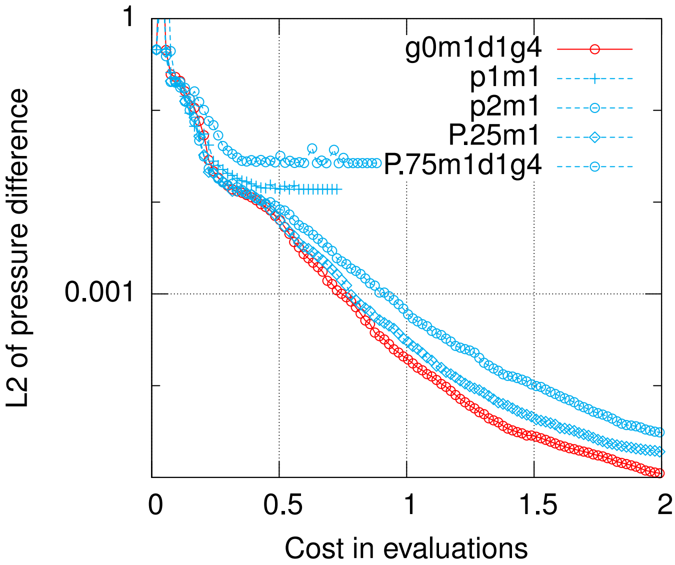
Smoothing the functional



Smoothing the functional

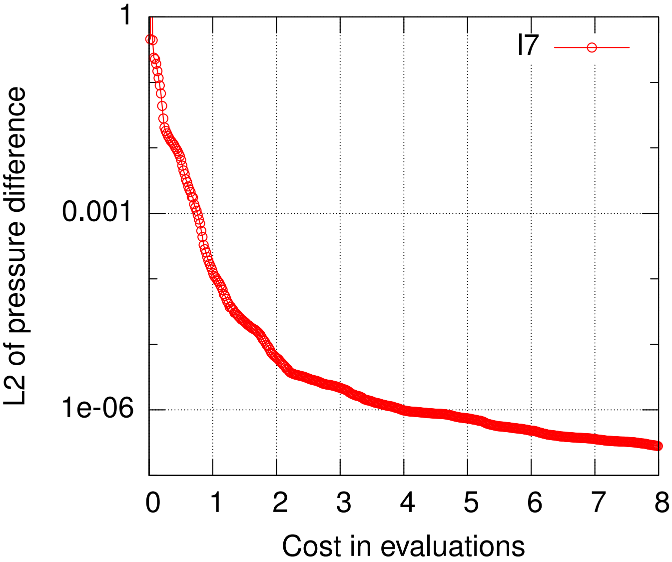


Smoothing the functional



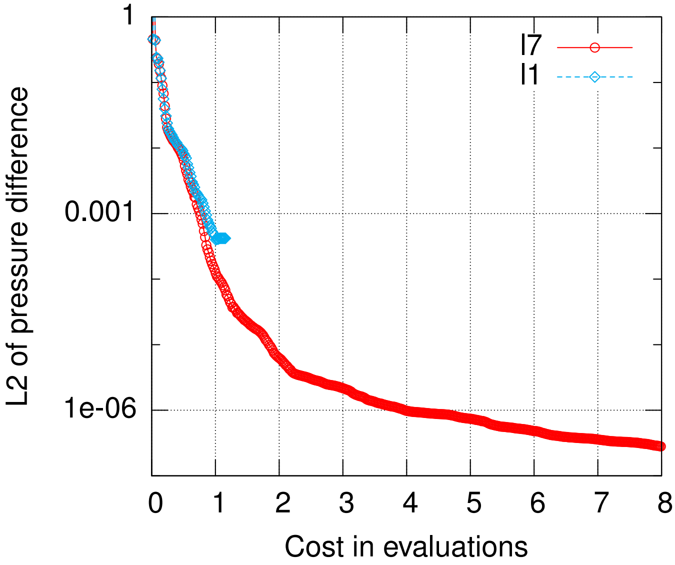
Influence of the approx. Hessian

Number of metric corrections in LBFGS



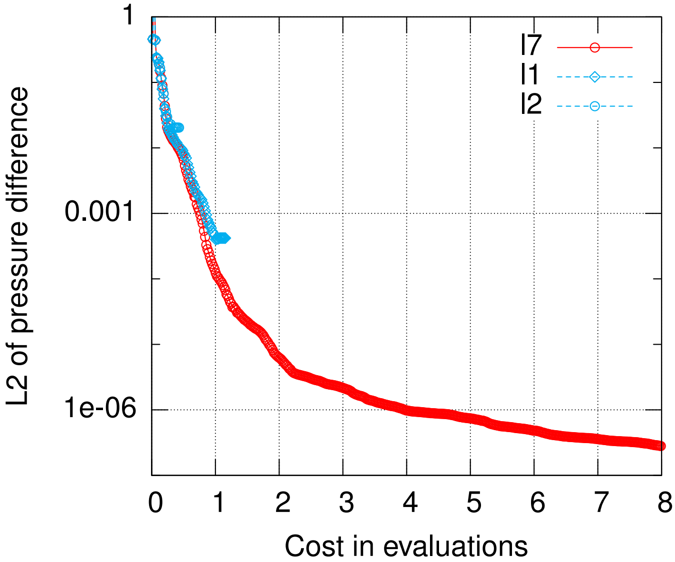
Influence of the approx. Hessian

Number of metric corrections in LBFGS



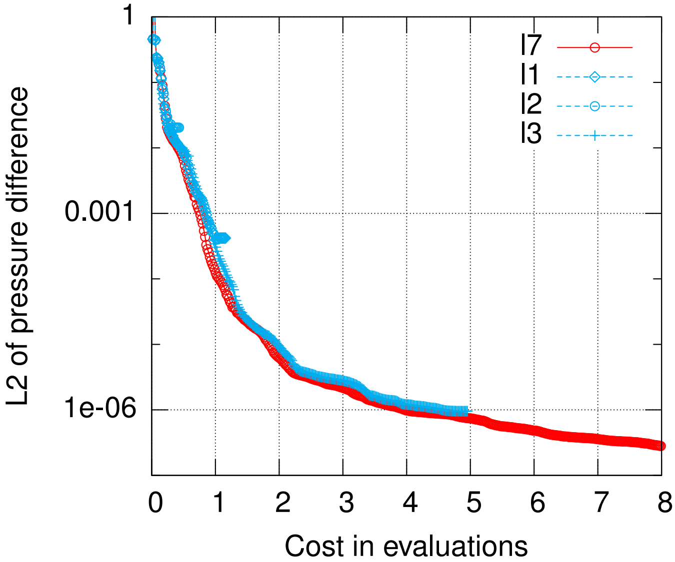
Influence of the approx. Hessian

Number of metric corrections in LBFGS



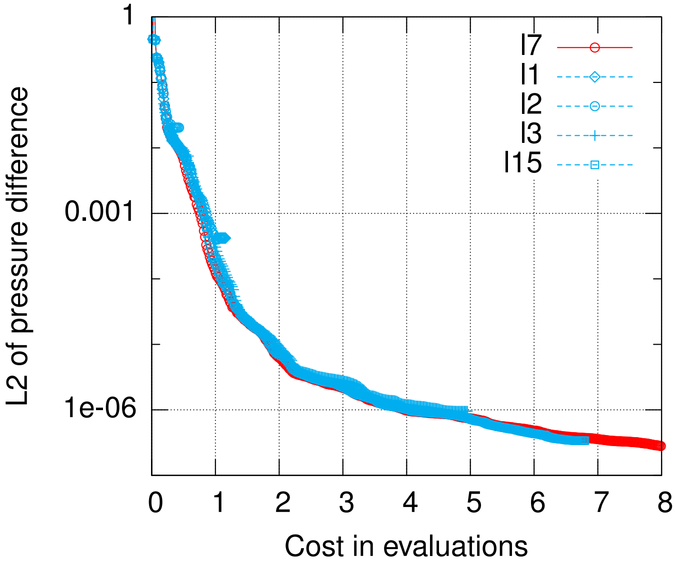
Influence of the approx. Hessian

Number of metric corrections in LBFGS



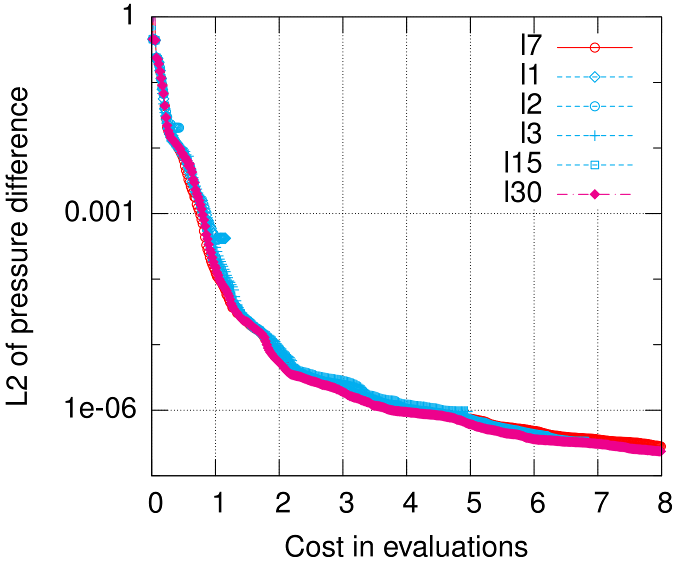
Influence of the approx. Hessian

Number of metric corrections in LBFGS



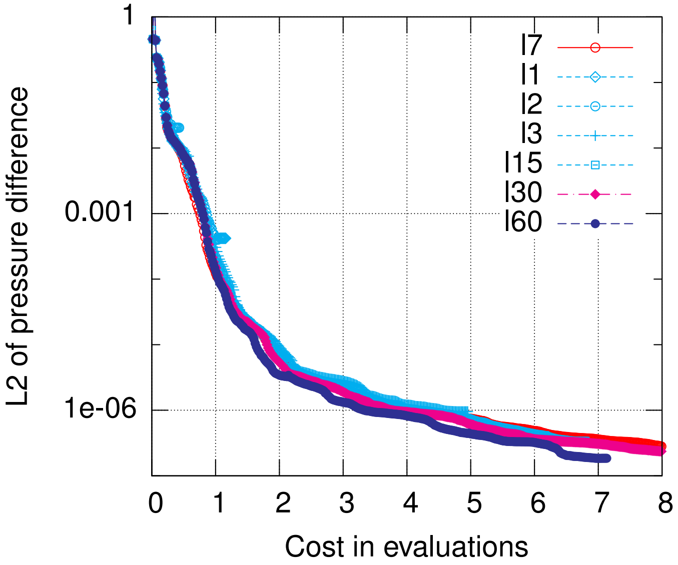
Influence of the approx. Hessian

Number of metric corrections in LBFGS



Influence of the approx. Hessian

Number of metric corrections in LBFGS



Summary

- Using the gradient-scaling of convergence as a preconditioner is effective for initial convergence of the optimality system
- Primal, adjoint and functional converge uniformly
- Convergence to engineering precision in cost of 5 evaluations
- Choice of scaling coefficient depends on the testcase
- Coarsening the design space improves initial convergence rate, but reduces final convergence level
- 1 Explicit Jacobi sweep on displacement provides best convergence rate and level
- Implicit Sobolev smoothing of gradients works less well

Summary

- Using the gradient-scaling of convergence as a preconditioner is effective for initial convergence of the optimality system
- Primal, adjoint and functional converge uniformly
- Convergence to engineering precision in cost of 5 evaluations
- Choice of scaling coefficient depends on the testcase
- Coarsening the design space improves initial convergence rate, but reduces final convergence level
- 1 Explicit Jacobi sweep on displacement provides best convergence rate and level
- Implicit Sobolev smoothing of gradients works less well

Summary

- Using the gradient-scaling of convergence as a preconditioner is effective for initial convergence of the optimality system
- Primal, adjoint and functional converge uniformly
- Convergence to engineering precision in cost of 5 evaluations
- Choice of scaling coefficient depends on the testcase
- Coarsening the design space improves initial convergence rate, but reduces final convergence level
- 1 Explicit Jacobi sweep on displacement provides best convergence rate and level
- Implicit Sobolev smoothing of gradients works less well

Summary

- Using the gradient-scaling of convergence as a preconditioner is effective for initial convergence of the optimality system
- Primal, adjoint and functional converge uniformly
- Convergence to engineering precision in cost of 5 evaluations
- Choice of scaling coefficient depends on the testcase
- Coarsening the design space improves initial convergence rate, but reduces final convergence level
- 1 Explicit Jacobi sweep on displacement provides best convergence rate and level
- Implicit Sobolev smoothing of gradients works less well

Summary

- Using the gradient-scaling of convergence as a preconditioner is effective for initial convergence of the optimality system
- Primal, adjoint and functional converge uniformly
- Convergence to engineering precision in cost of 5 evaluations
- Choice of scaling coefficient depends on the testcase
- Coarsening the design space improves initial convergence rate, but reduces final convergence level
- 1 Explicit Jacobi sweep on displacement provides best convergence rate and level
- Implicit Sobolev smoothing of gradients works less well

Summary

- Using the gradient-scaling of convergence as a preconditioner is effective for initial convergence of the optimality system
- Primal, adjoint and functional converge uniformly
- Convergence to engineering precision in cost of 5 evaluations
- Choice of scaling coefficient depends on the testcase
- Coarsening the design space improves initial convergence rate, but reduces final convergence level
- 1 Explicit Jacobi sweep on displacement provides best convergence rate and level
- Implicit Sobolev smoothing of gradients works less well

Summary

- Using the gradient-scaling of convergence as a preconditioner is effective for initial convergence of the optimality system
- Primal, adjoint and functional converge uniformly
- Convergence to engineering precision in cost of 5 evaluations
- Choice of scaling coefficient depends on the testcase
- Coarsening the design space improves initial convergence rate, but reduces final convergence level
- 1 Explicit Jacobi sweep on displacement provides best convergence rate and level
- Implicit Sobolev smoothing of gradients works less well

Further Work

- Improve convergence of coarsened design modes
- Experiment with alternative approximations to the Hessian
- Multigrid
- Inequality constraints
- Unsteady optimisation