Introduction to Gradient-Based Optimisation

Part 6: Adjoint methods

Dr. J.-D. Müller
School of Engineering and Materials Science,
Queen Mary, University of London
j.mueller@qmul.ac.uk

UK Fluids Network SIG on Numerical Optimisation with Fluids
Cambridge, 8-10 August 2018

© Jens-Dominik Müller, 2011-18, updated 8/8/18

Notes

Organisation of the lectures

1. Univariate optimisation
   - Bisection, Steepest Descent, Newton's method
2. Multivariate optimisation
   - Steepest descent, Newton's method
   - and line-search methods: Wolfe and Armijo conditions,
   - Quasi-Newton methods,
3. Constrained Optimisation:
   - Projected gradient methods,
   - Penalty methods, exterior and interior point methods,
   - SQP
4. Gradient computation
   - Manual derivation, Finite Differences
   - Algorithmic and automatic differentiation, fwd and bkwd.
5. Adjoint methods
   - Reversing time, Automatic Differentiation
   - Adjoint CFD codes

Notes

Outline

Notes

Outline

Notes

Physical meaning of the adjoint equations

The flow equations ask the question: "Where does a perturbation travel to?"

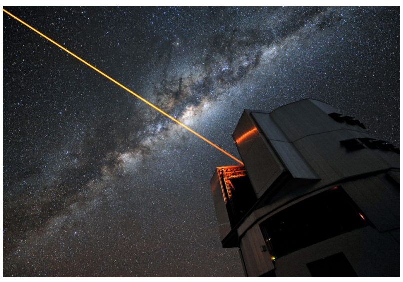
If we have N sticks, we need to ask N times

What if we could ask the question: "Where does a perturbation come from?"

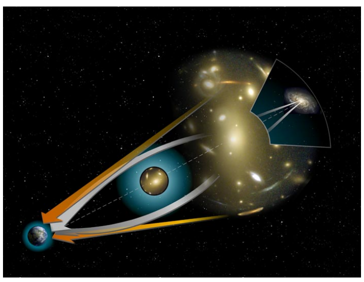
If we have M observation spots, we need to ask M times

Notes

Adjoint equations: the galactic view



Forward approach:
send a perturbation out



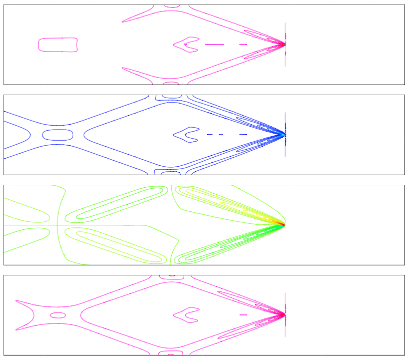
Reverse, adjoint approach:
trace back an incoming perturbation

Use the Force! Use the Force of the adjoint approach.

Notes

Physical meaning of the adjoint

- The adjoint asks: where does a perturbation come from.
- This reverses all time-like directions, or 'transposes' the system matrix.
- The adjoint solution quantifies the effect on the objective function brought by a unit source term in the conservation equations.



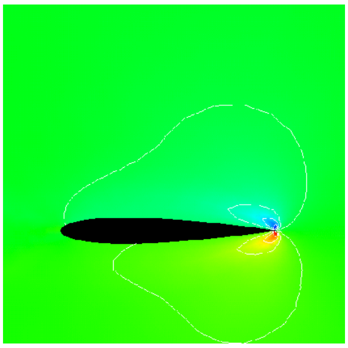
Adjoint solution for objective function of pressure in a point in supersonic flow in a channel from left to right.

Notes

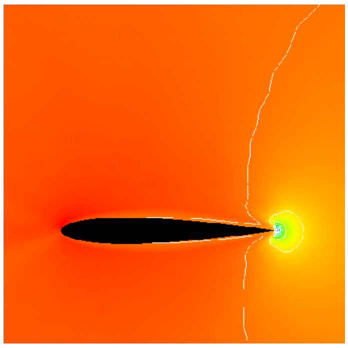
Example of an adjoint solution: aerofoil

NACA 0012, Ma=0.4, $\alpha = 2^{\circ}$
Sensitivity w.r.t. lift

mass flux



y-momentum



Notes

Outline

Notes

The continuous adjoint

Minimise the objective J, subject to the constraint to satisfy the conservation equations $\mathbf{R}(U, \alpha) = 0$:

$$I(U, \alpha) = J(U, \alpha) - \lambda^T \mathbf{R}(U, \alpha)$$

A linearised change in I is then

$$dI(U, \alpha) = \left(\frac{\partial J}{\partial U} - \lambda^T \frac{\partial \mathbf{R}}{\partial U} \right) dU + \left(\frac{\partial J}{\partial \alpha} - \lambda^T \frac{\partial \mathbf{R}}{\partial \alpha} \right) d\alpha$$

Choose λ to eliminate dU,

$$\left(\frac{\partial J(U, \alpha)}{\partial U} - \lambda^T \frac{\partial \mathbf{R}(U, \alpha)}{\partial U} \right) = 0$$

Then

$$dI(U, \alpha) = \left(\frac{\partial J(U, \alpha)}{\partial \alpha} - \lambda^T \frac{\partial \mathbf{R}(U, \alpha)}{\partial \alpha} \right) d\alpha$$

i.e., we no longer need to compute the state perturbation dU.

Notes

Example adjoint operators

primal	adjoint
$\frac{\partial u}{\partial x} - \varepsilon \frac{\partial^2 u}{\partial x^2}$	$-\frac{\partial v}{\partial x} - \varepsilon \frac{\partial^2 v}{\partial x^2}$
$\nabla \cdot (k \nabla u)$	$\nabla \cdot (k \nabla v)$
$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2}$	$-\frac{\partial v}{\partial t} - \frac{\partial^2 v}{\partial x^2}$
$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x}$	$-\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x}$

(Source: Giles, Pierce, 2001, "Introduction to the adjoint approach in design")

Notes

Outline

Notes

The discrete adjoint

Navier Stokes equations, fixed-point iteration to steady state:

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

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 an objective function L with respect to α

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} + \frac{\partial L}{\partial U}\frac{\partial U}{\partial \alpha} = \frac{\partial L}{\partial \alpha} + g^T u = \frac{\partial L}{\partial \alpha} + g^T \mathbf{A}^{-1} f$$

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

Notes

The Adjoint Equations

Regroup the terms in the sensitivity computation:

$$\frac{dL}{d\alpha} = \frac{\partial L}{\partial \alpha} + g^T \mathbf{A}^{-1} f = \frac{\partial L}{\partial \alpha} + \left(\mathbf{A}^{-T} g\right)^T f = \frac{\partial L}{\partial \alpha} + v^T f$$

leads to the definition of the adjoint equation:

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

From this follows the *Adjoint Equivalence*

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

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

Notes

Advantages of adjoint sensitivities

- Each design step requires a solve for $\mathbf{R}(\mathbf{U}) = 0$.
- Gradient-based optimisation requires a gradient for each design variable α_i.
- Using $g^T u$, each α_i needs a solve of $\mathbf{A}u = f$.
- Using $v^T f$, needs a single solve of $\mathbf{A}^T v = g$ and the evaluation of f_i for each α_i.
- Roughly speaking, solving $\mathbf{R}(\mathbf{U}) = 0$, $\mathbf{A}u = f$ and $\mathbf{A}^T v = g$ incur a similar cost.
- Computing f is of the order of a single explicit sweep, simplified boundary formulations exist.
- **Using the adjoint, the cost of gradient calculations for large design problems is essentially constant.**

Notes

Advantages of adjoint sensitivities (II)

- The forward method computes a perturbed flow field u and then the change in functional as $g^T u$.
- The adjoint solution directly computes the influence v of a source term f onto the functional L.
- We then need to evaluate the source f_i due to a design perturbation α_i.
- For a single design parameter, the cost of $g^T u$ and $v^T f$ are the same.
- **Using the adjoint the cost of gradient calculations for large design problems is essentially constant.**

Notes

Outline

Notes

Commercial

Commercial adjoint solvers are available, among others, from

- Ansys Fluent: incompressible, now also compressible. A mix of continuous and discrete.
- STAR CCM+: discrete
- Numeca: continuous

Aerospace:

- Rolls Royce: hydra (discrete)
- Airbus/DLR/Onera: tau (discr.), Flower (cont.), Elsa (cont.)
- MTU/DLR: trace (discr).

Notes

The need for open-source adjoint solvers

Existing open-source adjoint CFD solvers:

- OpenFOAM: incompressible flow solver, continuous adjoint. Needs substantial expertise to adjoint models, to overcome stability issues.
- SU2: compressible flow solver, continuous adjoint. Needs substantial expertise to adjoint models, to overcome stability issues.
- SU2: compressible flow solver, discrete adjoint using operator-overloading tool Codipack.
  - Adjoint code is 'in tape/call stack', not readable to the non-expert developer.
  - Memory requirements substantially improved over the years, but still multiples of the flow solver.
  - Obtaining similar performance for new models may need substantial expertise in Codipack.

Rationale for STAMPS

- provide a run-time and memory efficient open-source adjoint solver.

Notes

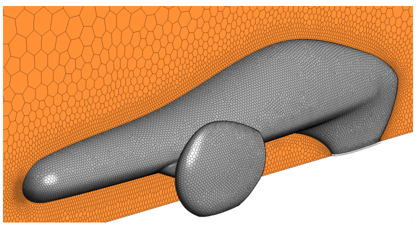
Assembling efficient discrete adjoints

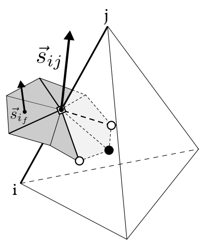
- We use a combination of AD and hand-differentiation:
  - Use AD for core routines like residual, gradients, limiters, fluxes...
  - Use hand-assembly for time-stepping, geometric multigrid and distributed-memory parallel communication
- Derivatives are a linearisation at a particular flow state. If the flow is at a steady state, only the final converged solution is needed to compute derivatives
- If the flow is unsteady (or the solver doesn't find a steady state), back-propagation of derivatives requires intermediate flow states. This typically requires large amount of memory, but can be reduced with check-pointing.
- In the first instance, focus on steady-state.

Notes

STAMPS: discretisation

Source-**T**ransformation **A**djoint **M**ulti-**P**arametrisation, (Physics, Parallelism) **S**olver





- Unstructured 3-D finite volume, vertex-centred solver.
- Physics: inviscid, laminar, RANS-turbulent ideal gas.
- Mesh-deformation coupled with a variety of geometric parametrisations
- Interfaces for FSI, CHT.

Notes

STAMPS: discretisation

Typical finite-volume compressible flow discretisation:

- compressible formulation with Roe and AUSM+ fluxes, MUSCL reconstruction up to second order accuracy
- node-centred discretisation, edge-based fluxes, edge- and cell-based gradients,
- Spalart-Allmaras turbulence model,
- explicit, block-Jacobi and implicit (JT-KIRK)[1]. timestepping for steady-state and unsteady flows (BDF2)
- GMRES + ILU preconditioner.
- Parallelisation with MPI.

[1] Xu, Müller: JT-KIRK, JCP 2015

Notes

STAMPS: design capabilities

STAMPS is specifically designed as a discrete adjoint CFD solver:

- discrete adjoint solver: derivatives are consistent with the flow solver: linear properties such as spectral radius of Jacobian are guaranteed.
- fully differentiable with AD Tool Tapenade (Inria, France) in tangent and adjoint mode: build of the adjoint code is completely automated.
- Tapenade uses source-transformation: the memory use and CPU-time per iteration are less than factor 2 to the flow, overall run-time of the adjoint can be down to 50% of the flow.
- coupled with a number of design parametrisation tools: node-based, NURBS-CAD-based and parameter-CAD-based.
- coupling with Calculix structures solver for FSI and CHT is currently undertaken.
- Adjoint-based mesh adaptation is currently being developed.

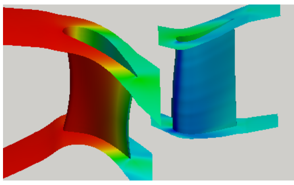
Notes

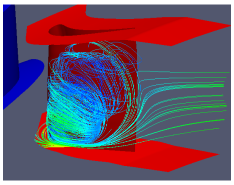
Outline

Notes

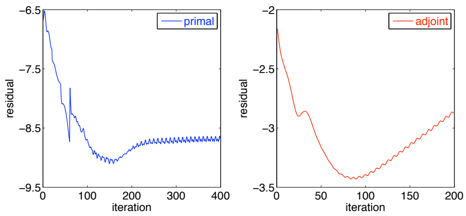
Convergence of the flow solver to limit cycles

A major problem with adjoint solvers is robustness.





Turbomachinery case in off-design condition, convergence of the CFD (left) to limit cycles, divergence of the adjoint solver (right).

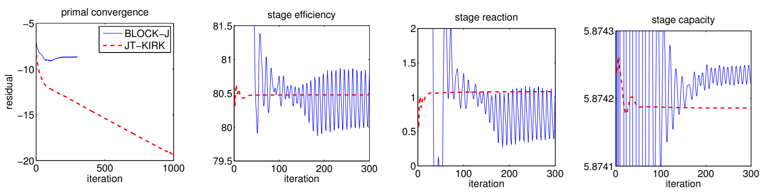


Notes

New iterative schemes for stable adjoints

In collaboration with Rolls Royce the group developed the more stable JT-KIRK time-stepping scheme[2] that is

- more efficient in runtime for the flow solver
- more stable in achieving full convergence for flow and adjoint.
- Typical cost functions such as efficiency, reaction, capacity converge much more rapidly to steady-state.

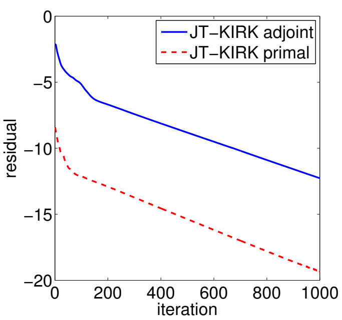


[2] S. Xu et al, "Stabilisation of discrete steady adjoint solvers", JCP 2015

Notes

Stable adjoints: essential for industrial optimisation

- Most importantly, convergence of the discrete adjoint can be achieved even for mildly unsteady flow situations.
- This is an essential ingredient for industrial application of gradient-based optimisation using adjoint methods.



Convergence history of both JT-KIRK primal and adjoint solvers.

Notes

Outline

Physical meaning of the adjoint equations

Continuous adjoints

Discrete adjoints

Available adjoint solvers

Stable discrete adjoint solvers

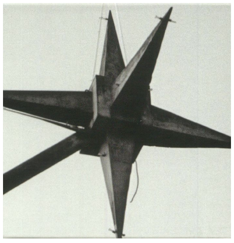
Automatic adjoint build

Notes

Framework for automatic application of S-T AD

Fully automated differentiation in tangent and reverse mode for

- fully coupled residual evaluation
- transport equations
- ILU precond. using AD'ed Jacobians
- Surface sensitivity projection
- adhering to coding templates ensures AD'ability
- two-layer halo MPI parallelisation: no MPI comm inside the FPI loop, no need to differentiate through MPI calls.
- Extensive use of Multi-Activity mode in Tapenade to derive efficient code for specialised derivative instances.



Notes

Provided fixed-point iterators

Simplified compressible fixed-point iterator

```
call initialise_flow ( ←U )
call metrics ( →X, ←Nrm )
do nIter = 1,mIt
  call residual ( →U, →Nrm, ←R )
  call update ( →R, ⇌U )
end do
call cost_fun ( →U, →Nrm, ←J )
```

Adjoint iterator using 'simple AD'.

```
U̅=0
call c̅o̅s̅t̅_̅f̅u̅n̅ ( ←U̅, ←N̅r̅m̅, 1 )
do nIter = mIt,1,-1
  call u̅p̅d̅a̅t̅e̅ ( ←R̅, ⇌U̅ )
  call r̅e̅s̅i̅d̅u̅a̅l̅ ( ←U̅ ←N̅r̅m̅, →R̅)
end do
call m̅e̅t̅r̅i̅c̅s̅ ( ←X̅, →N̅r̅m̅ )
```


Notes

Provided fixed-point iterators

Adjoint iterator using 'simple AD'.

```
U̅=0
call cost_fun̅ ( ←U̅, ←N̅r̅m̅, 1 )
do nIter = mIt,1,-1
  call update̅ ( ←R̅, ⇌U̅ )
  call residual̅ ( ←U̅ ←N̅r̅m̅, →R̅)
end do
call metrics̅ ( ←X̅, →N̅r̅m̅ )
```

- N̅r̅m̅ is recomputed at every iteration, but only used after exiting the FPI loop.
- Adjoint solution is accumulated, has to be initialised to U̅=0.

Notes

Provided fixed-point iterators

Adjoint iterator using 'simple AD'.

```
U̅=0
call cost_fun̅ ( ←U̅, ←N̅r̅m̅, 1 )
do nIter = mIt,1,-1
  call update̅ ( ←R̅, ⇌U̅ )
  call residual̅ ( ←U̅ ←N̅r̅m̅, →R̅)
end do
call metrics̅ ( ←X̅, →N̅r̅m̅ )
```

Adjoint iterator derived from the primal time-stepping (PTS)

```
call cost_fun̅ ( ←g, ←N̅r̅m̅, 1 )
do nIter = 1,mIt
  call residual_u̅ ( ←R̅, →U̅ )
  R̅ = R̅ - g
  call update ( →R̅, ⇌U̅ )
end do
call residual_nrm̅ ( →U̅, ←N̅r̅m̅ )
call metrics̅ ( ←X̅, →N̅r̅m̅ )
```

Notes

CPU and memory performance of multi-target AD

Runtime and memory performance of general and specialised (multi-target) differentiation.

	runtime	runtime (rel.)	memory	memory (rel.)
primal	211.1s	1	360.93MB	1
general	328.8s	1.56	431.68MB	1.20
special	249.1s	1.18	432.62MB	1.20
change	**-32%**		**0.2%**	

Peak memory use (measured with valgrind/massif)

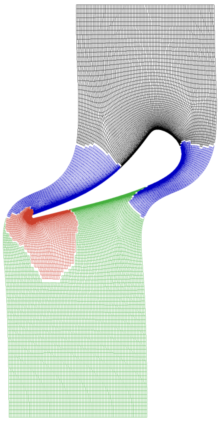
Case	flow Gb	adj. Gb	ratio
flatPlate, 2D quad, visc	0.217	0.260	1.20
rae2822, 2D quad, inv	0.199	0.231	1.16
DeathStar, 3D unstr, inv	0.331	0.368	1.12
TUB Stator, 3D hexa, visc	5.98	6.81	1.14

Notes

MPI parallelisation

The aim is to support a simple adjoint build that does not require the user to mange MPI comm inside the FPI loop.

- Partitioning with Metis graph partitioner
- Two layers of halo cells ensure that a 5 point stencil is rank-local, no MPI messages inside the FPI loop.
- Include periodic edges in the graph to have periodic pairs rank-local which avoids MPI communication inside the FPI loop.



Notes

Acknowledgements

Parts of this work have been conducted within the **About Flow**, and **IODA** projects at Queen Mary University of London

http://{aboutflow,ioda}.sems.qmul.ac.uk

We have received funding from the European Union's Seventh Framework Programme and Horizon 2020 under Grant Agreement Nos. 317006, 642959.

Notes

Notes