Adjoint Methods on Unstructured Grids:

the Discrete Approach, using Automatic Differentiation, Applications to Optimal Design.

Jens-Dominik Müller
Paul Cusdin

School of Aeronautical Engineering
Queen's University Belfast

Queen's University Belfast
School of Aeronautical Engineering

Section I: What is an Adjoint?

Outline:

- The adjoint equations
- What is the advantage of using adjoints?
- Physical interpretation of the adjoint solution.
- The discrete adjoint.

Adjoint Theory (I)

Discretised Navier-Stokes eq.

$$\frac{\partial U}{\partial t} + \mathsf{R}(U) = 0$$

Linearisation with respect to design variable α

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

$$\frac{\partial R}{\partial U}\frac{\partial U}{\partial \alpha} = -\frac{\partial R}{\partial \alpha},$$

$$\mathbf{A}u = f.$$

Adjoint Theory (II)

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, the term to be determined is

$$\frac{\partial L}{\partial U}\frac{\partial U}{\partial \alpha} = g^T u.$$

Requires computation of the perturbation flow field

$$u = \frac{\partial U}{\partial \alpha}.$$

Adjoint Theory (III)

Definition of the Adjoint Problem

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

$$\mathbf{A}^T v = g.$$

From this follows the *Adjoint Equivalence*

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

and the functional sensitivity becomes

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

Why use an Adjoint for Design (I)?

- 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 iteration.

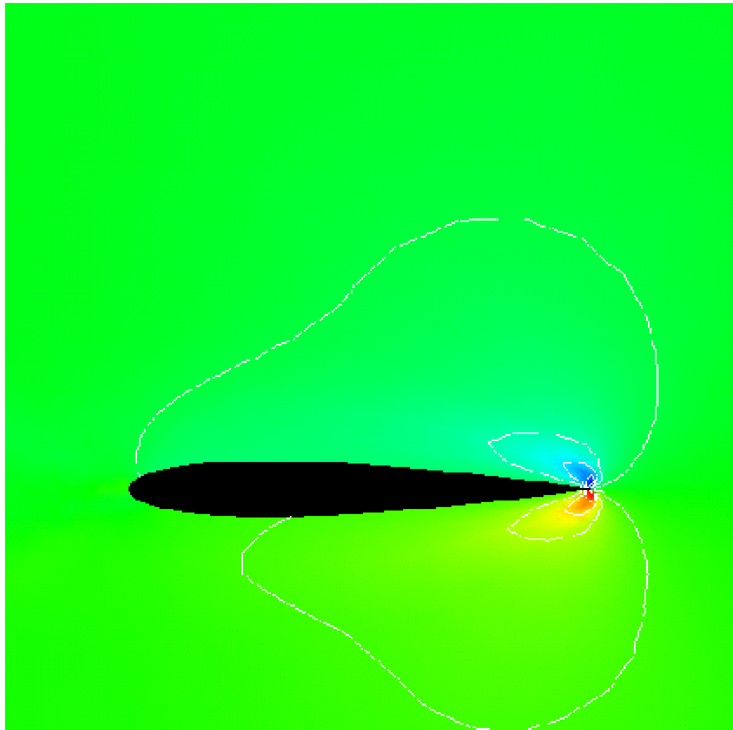
Why use an Adjoint for Design (II)?

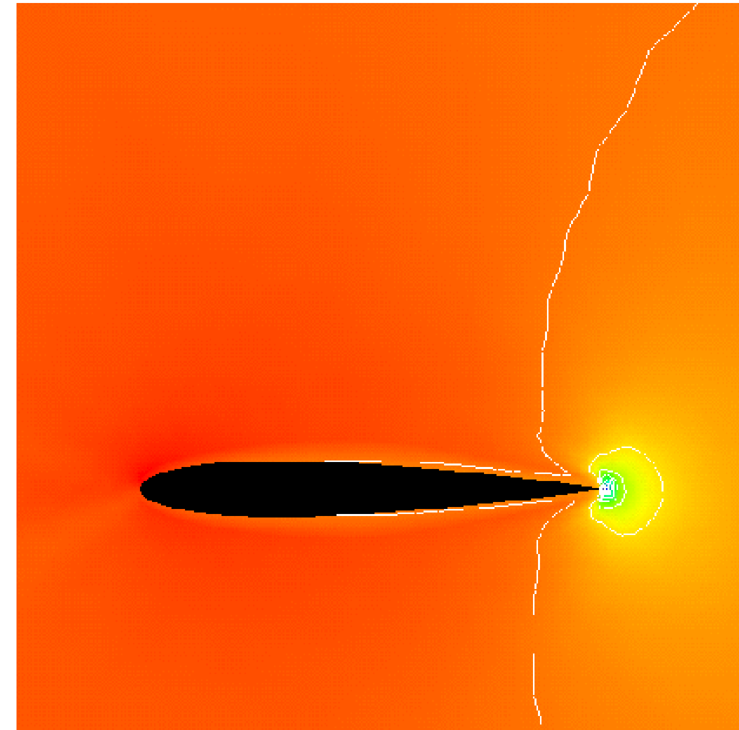
- The forward method computes a perturbed flow field u and then the change in functional as $g^T u$.
- The adjoint directly computes the influence v of a source term f onto the functional L.
- We then need to evaluate the source f_i due to a design perturbation α_i.
- **Using the adjoint the cost of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

Adjoint Equations: the Inverse Viewpoint

Example: NACA 0012, Ma=0.4, $\alpha = 2^\circ$

mass flux　　　　y-momentum





Pros and Cons of the Exact Discrete Approach

Advantages

- $\mathbf{A}^T$ contains the weak boundary conditions.
- $\mathbf{A}$ and $\mathbf{A}^T$ have the same eigenvals, eigenvecs.
- Convergence acceleration tools remain optimal.
- Functional converges at the exact same rate.
- Easy validation of the adjoint against the linear code.
- Validation of the linear code against the non-linear one and complex variable techniques.
- Process is automatable.

Disadvantages:

- Performance? Until recently the continuous approach was considered to be faster and less memory intensive.
- Work on the adjoint equations could give better insight.

Section II: Automatic Differentiation

Outline:

1. Explicit Finite Volume Schemes
2. AD tools used in our studies
3. AD procedure
4. Validation of the Adjoint
5. Performance

Preparing code: a typical finite volume code

```
do nIter = 1, mIter

  Reset to zero
  do nFV = 1, mFV
    res(nFV) = 0
    dt(nFV) = 0
  end do

  Calculate residuals
  do nFace = 1, mFace
    call flux(ql, qr, ds, cosal, sinal, lambda, r)
    res(L) = res(L) + r
    res(R) = res(R) - r
    dt(L) = dt(L) + lambda
    dt(R) = dt(R) + lambda
  end do

  Update
  do nFV = 1, mFV
    call update(dt, res, q)
  end do

end do
```

Nonlinear, Linear and Adjoint Flux calculations

$$\mathbf{U}^{n+1} = \mathbf{U}^{n} + \sigma \left(\frac{\Delta t}{A} \right) R(\mathbf{U})$$

$$u^{n+1} = u^{n} + \sigma \left(\frac{\Delta t}{A} \right) [\mathbf{A}u - f]$$

$$v^{n} = v^{n+1} + \sigma \left(\frac{\Delta t}{A} \right) [\mathbf{A}^T v - g]$$

$$R = \sum_e R_e(\mathbf{U})$$

$$\mathbf{A}u = \sum_e \mathbf{A}_e u$$

$$\mathbf{A}^T v = \sum_e \mathbf{A}_e^T v$$

Input and Output Variables of the Flux Routines

nonlinear:	`call flux (→UL, →UR, ←R)`
	`Res(L) += R`
	`Res(R) -= R`

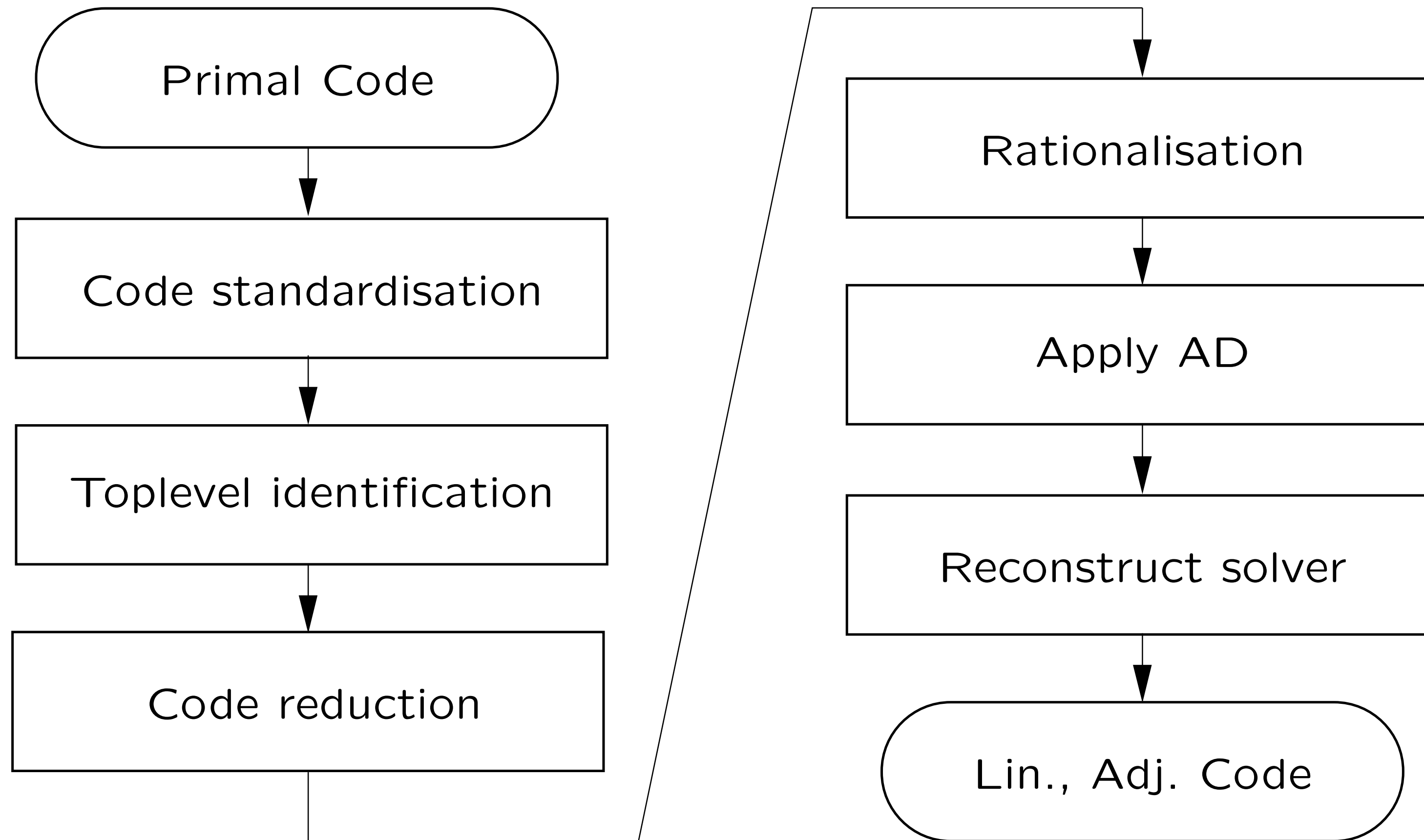
linear:	`call lin_flux (→uL, →uR, ←Au)`
	`lin_res(L) += Au`
	`lin_res(R) -= Au`

adjoint:	`v = vL - vR` `call adj_flux (←AtvL, ←AtvR, →v)`

Automatic Differentiation Tools

	ADIFOR 2.0	Tapenade	TAMC	TAF
Developers	ANL, Rice Univ.	INRIA	FastOpt	FastOpt
Research licence	Free, w. restr.	Free	Free, w. restr.	Fee
Commercial support	✗	✗	✗	✓
Local installation	✓	✓	✗	✗
Remote execution	✗	✓	✓	ssh
Reverse model compiler	via ADIFOR 3.0	✓	✓	✓
Uses library functions	✗	✗	✓	✓

Preparing code for Automatic Differentiation



What AD might not deal with

- Non-standard language extensions
- Pre-processing directives
- System calls
- Parallelisation and pointers
- External routines
- Global variables, common blocks
- Assumed length of arrays
- Aliasing
- Mismatched data types, precision
- goto
- Discontinuous intrinsic functions
- Time-stepping

AD deals with hard boundary conditions

primal:

```
do nIter = 1, mIter

 residual

 boundary

 update

end do
```

adjoint:

```
do nIter = mIter, 1, -1

 residual

 adboundary

 adresidual

 update

end do
```

AD deals with preconditioning

```
do nIter = 1, mIter
   call res ( → q, ← res, dt )
   call precon ( q, → res )
   call update ( → res, dt, ← q )
end do
```

	Tan. Orig.		Tan. Precon.		Adj. Orig.		Adj. Precon.	
Mach	Its	CPU (s)	Its	CPU (s)	Its	CPU (s)	Its	CPU (s)
0.1	1062	**17.59**	568	**12.13**	1062	**35.86**	568	**27.84**
0.05	1932	**31.82**	566	**12.11**	1932	**47.10**	566	**27.88**
0.01	10680	**176.1**	565	**12.05**	10680	**259.9**	565	**27.90**
0.005	21545	**354.6**	565	**12.12**	21545	**523.5**	565	**27.96**

Using AD to compute the source terms

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

$$f = \frac{\partial R}{\partial X}\frac{\partial X}{\partial \alpha}$$

```
subroutine res (q,→ds,→cos,→sin,←res)

subroutine gf_res (q,ds,→gf_ds,cos,→gf_cos,sin,→gf_sin,←gf_res)

subroutine metrics (x,y,→a,←ds ←cos,←sin)

subroutine gf_metrics (x,y,a,→gf_a,←gf_ds ←gf_cos,←gf_sin)
```

Validation of Adjoint and Linear Fluxes

Linear against complex-variable method:

$$\mathbf{A}u = \frac{\Im[R(\mathbf{U} + \epsilon u \mathbf{i})]}{\epsilon} + O(\epsilon^2)$$
$$f = \frac{\Im[R(\mathbf{U}(x + \epsilon x' \mathbf{i})]}{\epsilon} + O(\epsilon^2).$$

Adjoint against linear, e.g. Roe flux:

$$g^T u = \underbrace{[\mathbf{A}^T(v_L - v_R)]_L^T}_{\text{adj. out left}} u_L + \underbrace{[\mathbf{A}^T(v_L - v_R)]_R^T}_{\text{adj. out r}} u_R$$
$$= [(v_L - v_R)]^T \underbrace{\mathbf{A}u}_{\text{lin. out}} = v^T f$$

Adjoint Code Validation

Fluxes:

- Compare finite differences of the primal to the Jacobian $\mathbf{A}$ of the linearised code by setting u to unit vectors.
- Verify $\mathbf{A}u$ using the complex variable method.
- Verify the adjoint identity:

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

Timestepping:

- Verify identical convergence of the functional.

Linear against Adjoint validation at Each Timestep

Iteration	$g^T u$	$v^T f$
1	-4.16320068334459	-4.16320068334459
10	-17.5119538321940	-17.5119538321940
100	-17.6258008647212	-17.6258008647212
1000	-17.6339808335237	-17.6339808335237

Validation of the Sensitivity

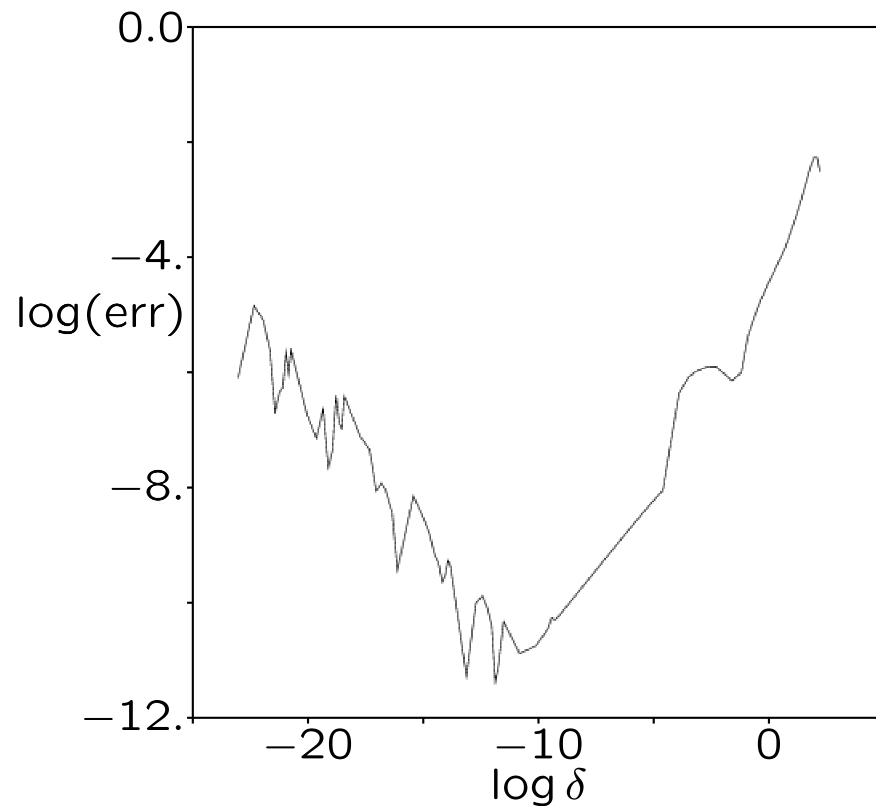
Sensitivity of Cl with respect to angle of attack $\partial c_L / \partial \alpha$:

	Mach = 0.68	Mach = 0.88
Adjoint	0.141 680 148 070 555	0.218 569 888 067 055
Linear	0.141 680 148 070 555	0.218 569 888 067 055
CV	0.141 680 148 070 556	0.218 569 888 067 055
FD	0.141 680 148 019 944	0.218 569 882 143 528

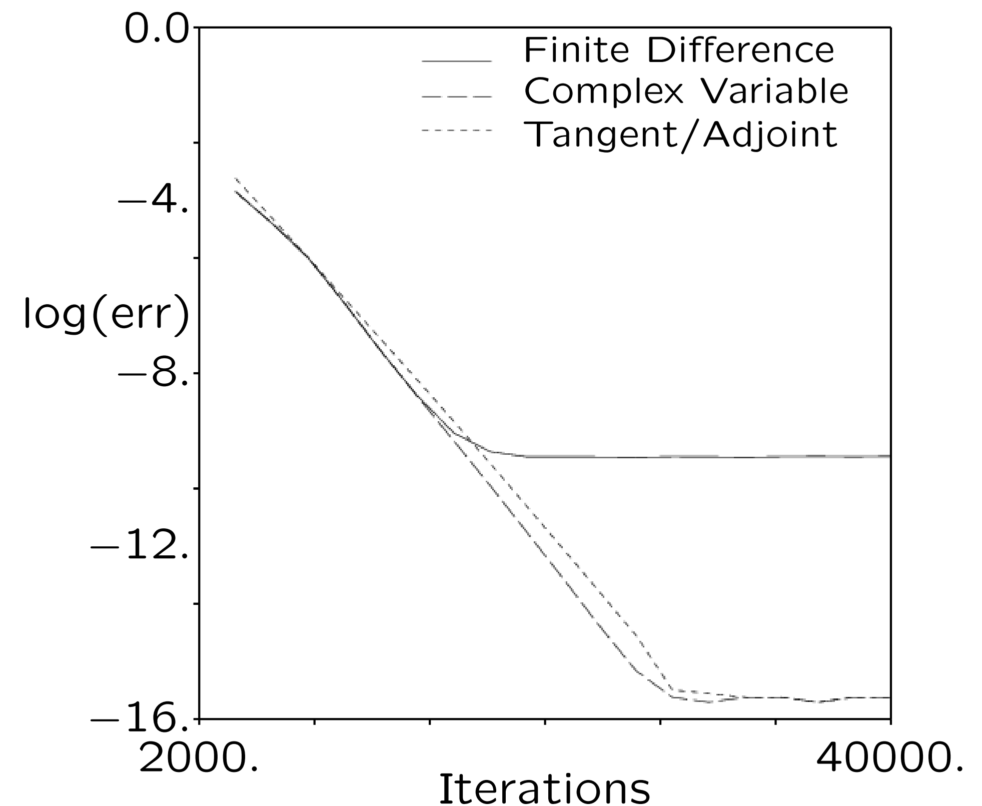
(FD: $\delta = 0.0002$, CV: $\epsilon =$ 1e-20):

Convergence of the functional

Finite difference error:



Iterative convergence:



Performance of AD code

Primal

```
   scale = max( scale, abs(diffs1 - diffs2) /
$ ( abs(diffs1) + abs(diffs2) + abs(sc(nv,n))))
```

TAMC tangent-mode transformation

```
   g_scalf = g_scalf*(0.5+sign(0.5d0,scale-scaleo/(scalen+
$scalem+scalel))-g_sc(nv,n)*(0.5-sign(0.5d0,scale-scaleo/(scalen+
$scalem+scalel)))*(scaleo/((scalen+scalem+scalel)*(scalen+scalem+
$scalel)))*sign(1.d0,sc(nv,n))-g_diffs2*(0.5-sign(0.5d0,scale-
$scaleo/(scalen+scalem+scalel)))*(scaleo/((scalen+scalem+scalel)*
$(scalen+scalem+scalel)))*sign(1.d0,diffs2)-g_diffs1*(0.5-
$sign(0.5d0,scale-scaleo/(scalen+scalem+scalel)))*(scaleo/((scalen
$scalem+scalel)*(scalen+scalem+scalel)))*sign(1.d0,diffs1)+
$(g_diffs1*sign(1.d0,diffs1-diffs2)-g_diffs2*sign(1.d0,diffs1-
$diffs2))*((0.5-sign(0.5d0,scale-scaleo/(scalen+scalem+scalel)))/
$(scalen+scalem+scalel))
```

Improving Performance of AD derived code

- Replace non-continuous intrinsic functions with `if-else-end`.
- Avoid overwritten (recycled) variables
- Introduce temporary variables to avoid recomputation
- Reduce the number of stored variables (Tapenade)
- Make the storage method ("tape") run faster (Tapenade)
- Reduce the right-hand-side length
- Dead code analysis

Performance Data: 2D Unstr. Euler code

	Itanium 2		Xeon		Athlon	
Primal (secs)	37.5	**33.9**	49.9	**47.2**	65.2	**64.8**
Hand tan	2.43		2.33		2.24	
Adifor 2	2.12	**2.12**	2.62	**2.52**	2.43	**2.39**
TAF tan	4.54	**2.27**	2.99	**2.23**	2.86	**2.10**
Tap.tan	1.87	**1.96**	2.58	**2.32**	2.22	**2.17**
(Tap.tan)	(1.96)		(2.31)		(2.06)	
Hand rev	2.24		2.65		2.35	
TAF rev	8.14	**2.46**	4.09	**2.92**	3.77	**2.65**
Tap.rev	11.53	**8.28**	12.02	**3.79**	8.82	**3.14**
(Tap.rev)	(2.32)		(2.89)		(2.49)	

Performance Data: 3D Unstr. N.-S. code

	Itanium 2		Xeon		Athlon	
Primal (secs)	104.4	**108.9**	315.0	**310.4**	521.8	**516.4**
Hand tan	1.88		1.20		1.13	
Adifor 2	4.59	**3.84**	1.71	**1.62**	1.68	**1.66**
TAF tan	4.35	**3.83**	1.25	**1.21**	1.18	**1.17**
Tap.tan	4.14	**3.97**	2.14	**2.10**	1.89	**1.87**
(Tap.tan)		(1.89)		(1.18)		(1.16)
Hand rev	1.73		1.19		1.16	
TAF rev	3.97	**3.62**	1.25	**1.21**	1.26	**1.22**
Tap.rev	28.03	**3.41**	11.64	**1.99**	10.16	**1.62**
(Tap.rev		(3.49)		(1.22)		(1.17)

Performance Data: Memory Use

3D unstructured Navier-Stokes Code:

	Linear		Adjoint	
Primal	343	**343**	343	**343**
Hand	1.27	**1.27**	1.27	**1.27**
Adifor	1.62	**1.62**		
TAF	1.27	**1.27**	1.27	**1.27**
Tap.	1.62	**1.62**	1.91	**1.62**

Conclusions on Automatic Differentiation

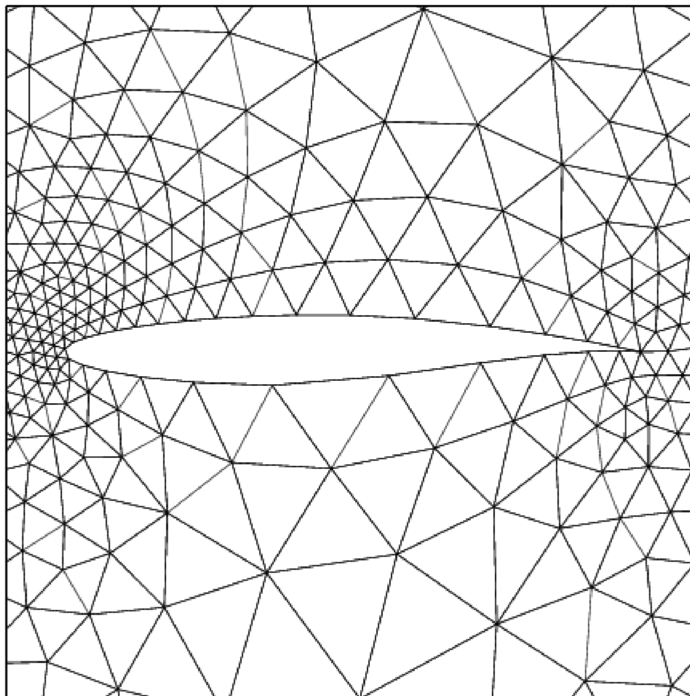
- AD tools mature enough to be used for unstructured CFD
- Process can be fully automated
- Advanced tools like TAF simplify treatment of iterative loops, parallel calls, etc.
- Performance can be influenced with the construction of the primal.

Section III: Adjoint Methods in Optimal Design

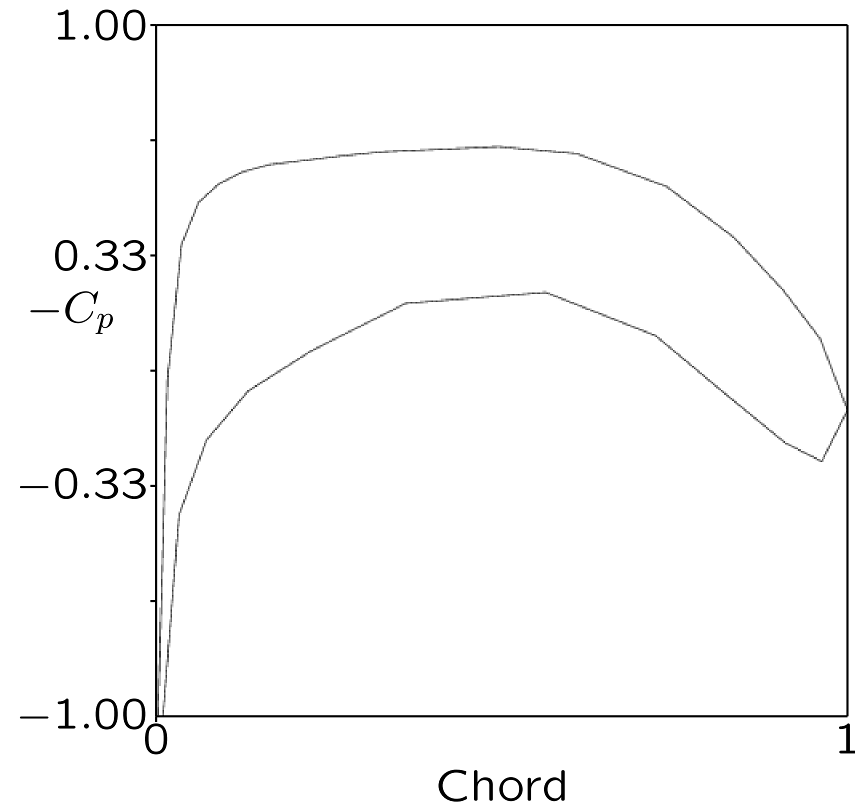
- The adjoint gradients replace the tangent gradients in the optimisation algorithm: $g^T u = v^T f$.
- The cost is dramatically reduced.
- AD-derived tangent and adjoint gradients are exact, as opposed to finite-difference gradients.
- In our examples: use sequential optimisation with a quasi-Newton method (L-BFGS) and well-converged primal and adjoint.

Example: inverse design, target RAE 2822

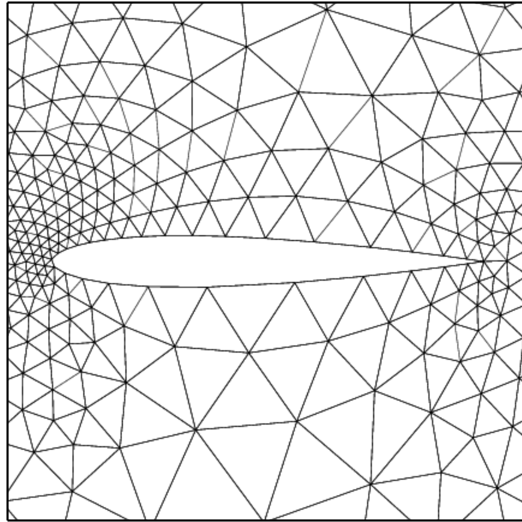
Mesh

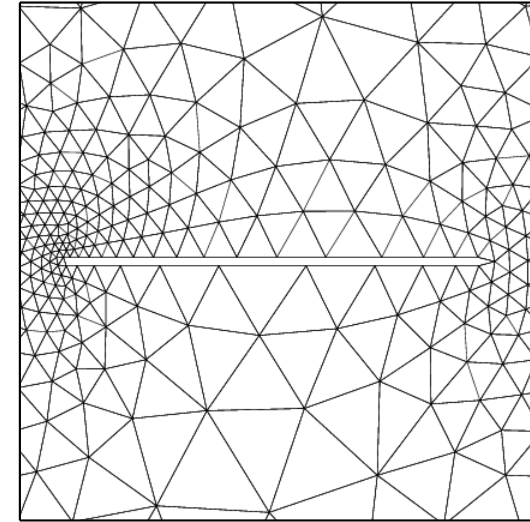


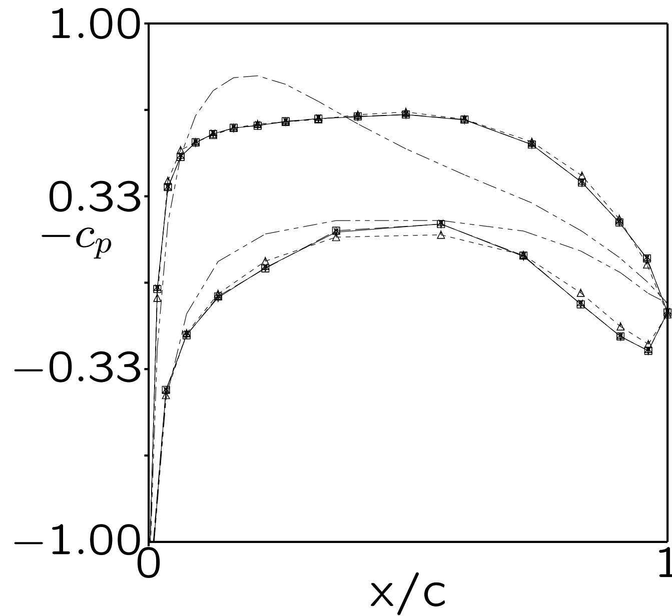
Pressure coefficient

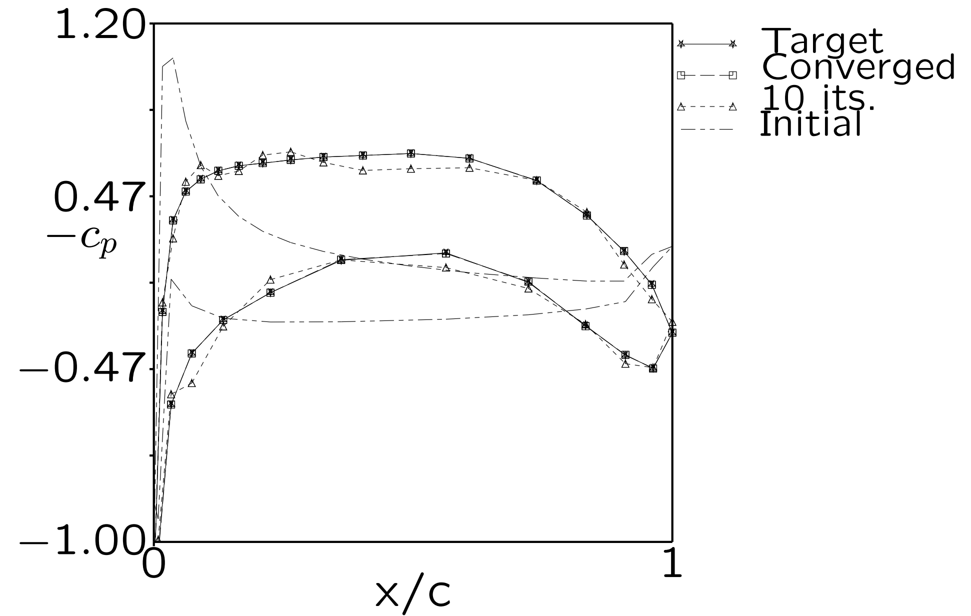


Inverse Design, two starting points

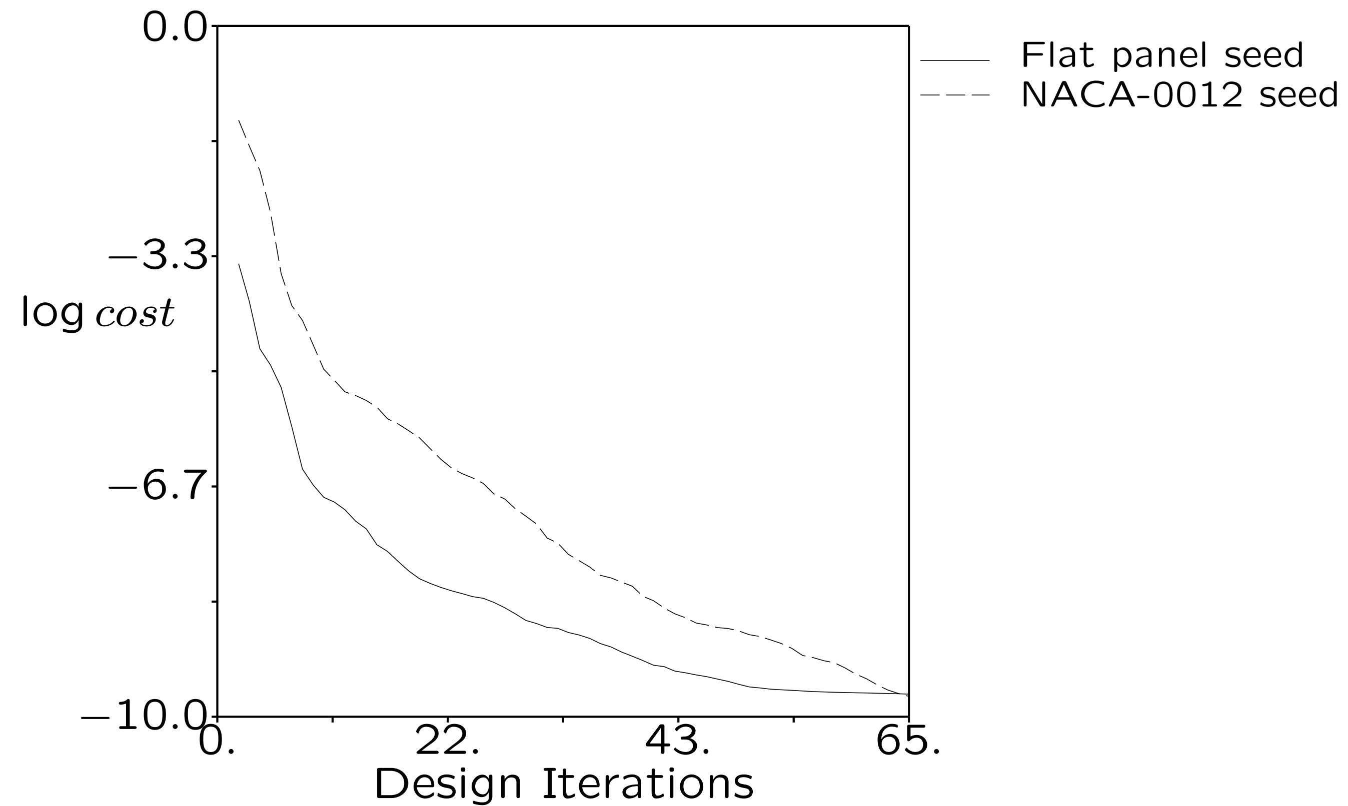








Convergence of the design



Lift-constrained Drag Minimisation

How to impose constraints using adjoints?

- Ideally, satisfy the constraint exactly: hard constraint.
- This requires knowledge of the sensitivity of the constraint w.r.t. to the design variables, hence an additional adjoint solve.
- Alternatively, satisfy the constraint approximately using a penalty function:

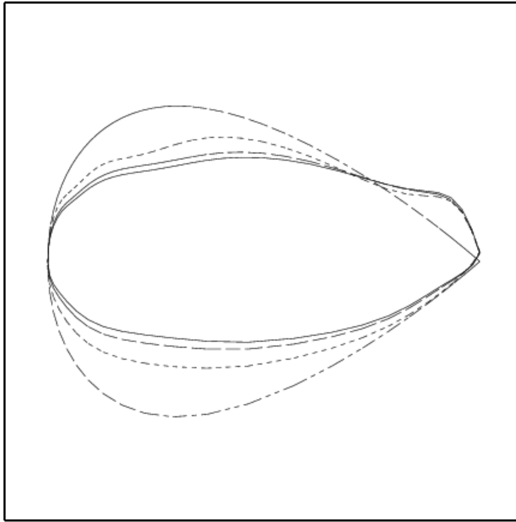
$$F = c_D^2 + \lambda(c_L - c_{L_T})^2$$

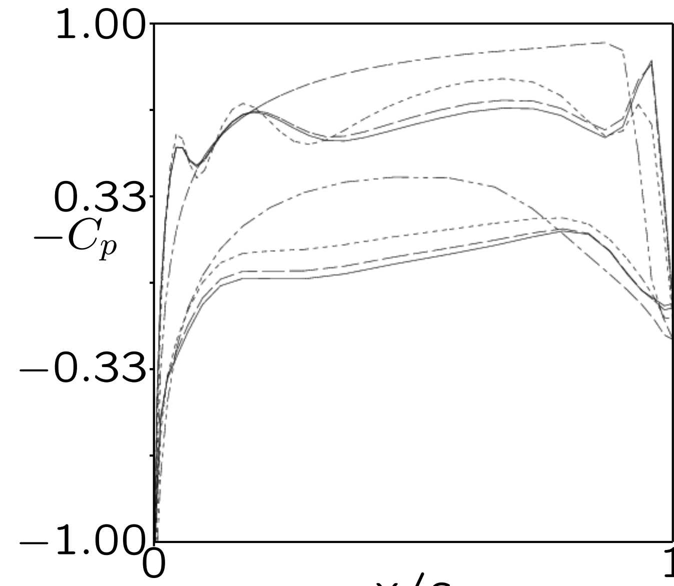
- It is not straightforward to choose a good value for λ.

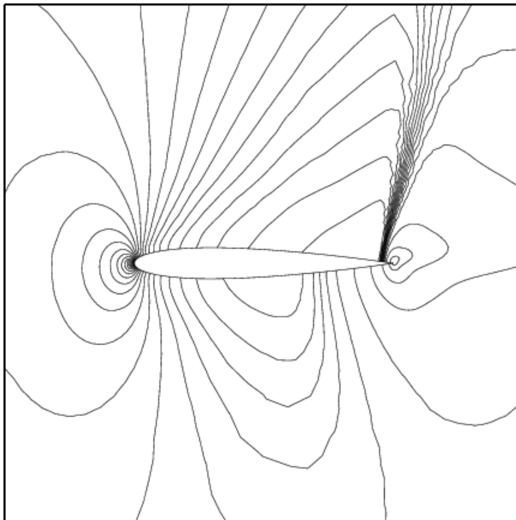
Effect of the penalty parameter

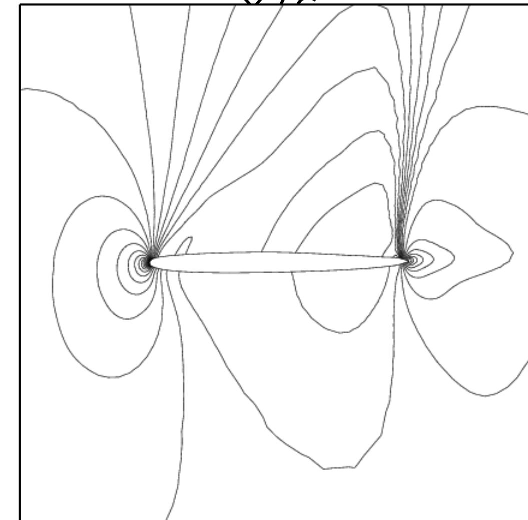
λ	$c_{L_T} = +5\%$ Its.	c_{L_D}	c_{D_D}	$c_{L_T} = \pm 0\%$ Its.	c_{L_D}	c_{D_D}	$c_{L_T} = -5\%$ Its.	c_{L_D}	c_{D_D}
1x10^{1}	DNC			DNC			DNC		
1x10^{0}	35	0.526	0.053	39	0.501	0.053	40	0.476	0.053
5x10^{-1}	31	0.526	0.053	27	0.501	0.053	36	0.476	0.053
1x10^{-1}	20	0.526	0.053	18	0.501	0.053	22	0.476	0.053
5x10^{-2}	17	0.526	0.053	19	0.501	0.053	17	0.476	0.053
1x10^{-2}	DNC			DNC			DNC		

Lift-constrained Drag Minimisation, Example







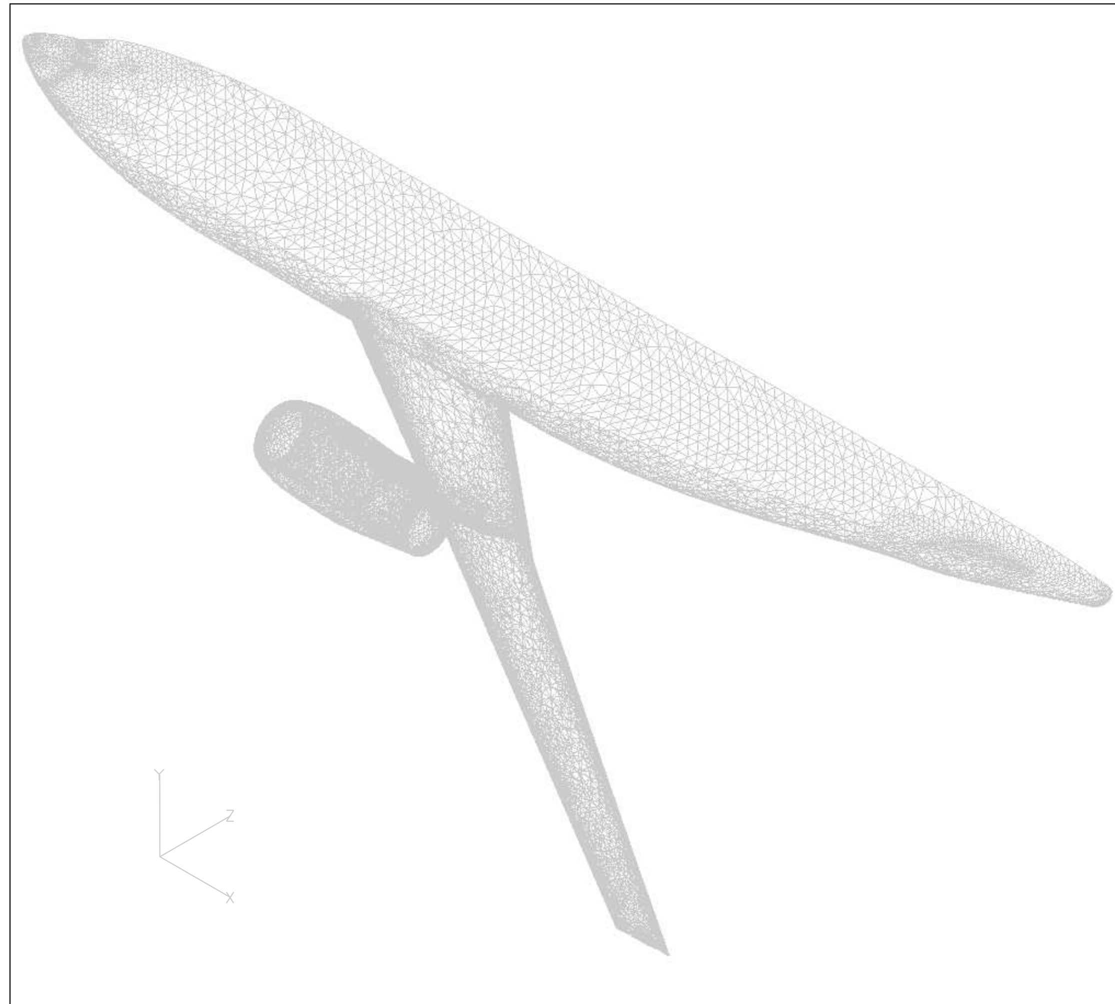


Comparison with finite differences

- 127 design points,
- L-BFGS, termination for proj. grad $< 10^{-7}$.

λ	Finite Difference Its.	Finite Difference CPU Time (s)	Adjoint Its.	Adjoint CPU Time (s)
$1\text{x}10^{1}$	DNC		DNC	
$1\text{x}10^{0}$	46	740,921.2	39	18,070.8
$5\text{x}10^{-1}$	18	349,879.5	27	12,755.8
$1\text{x}10^{-1}$	17	329,298.3	18	9,035.4
$5\text{x}10^{-2}$	18	349,881.4	19	9,566.9
$1\text{x}10^{-2}$	DNC		DNC	

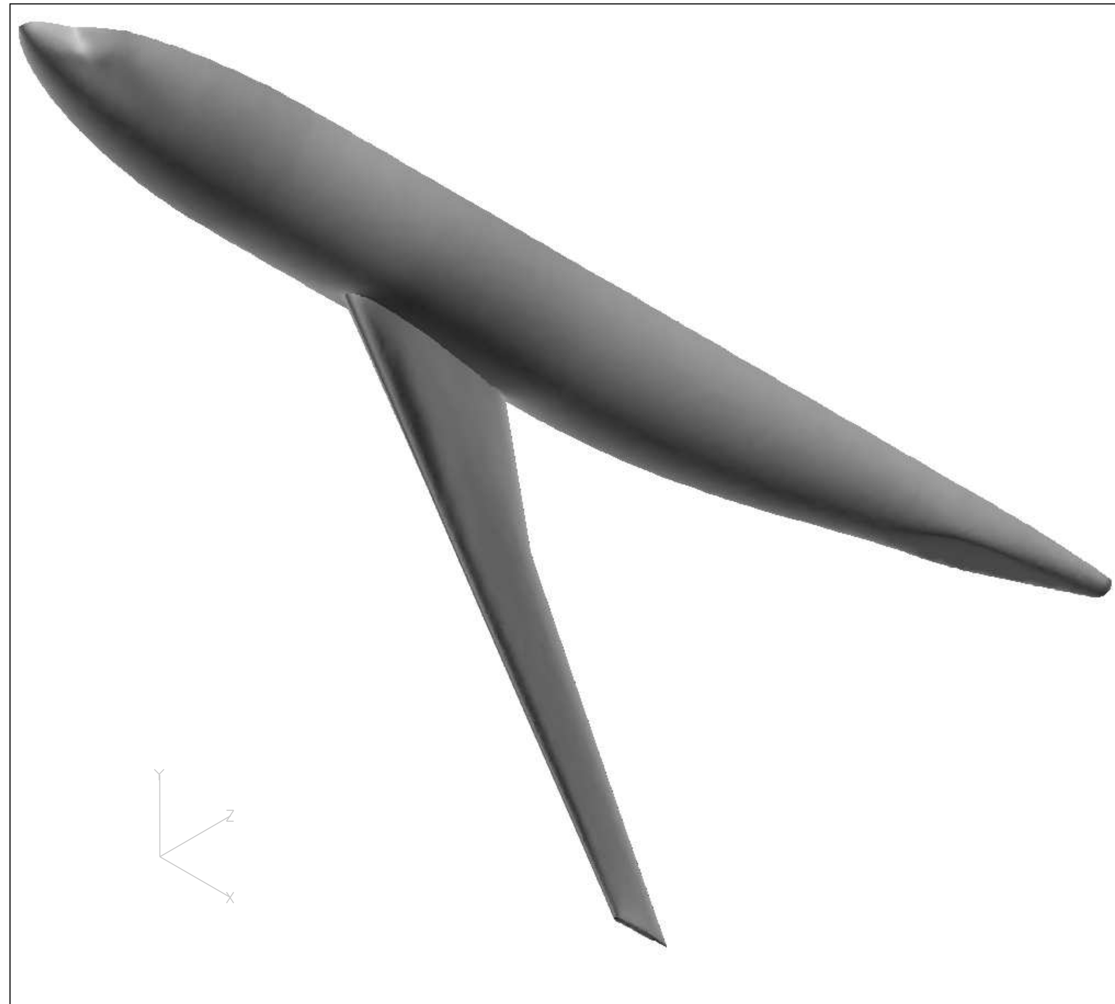
3D example: Optimising F6 Nacelle installation



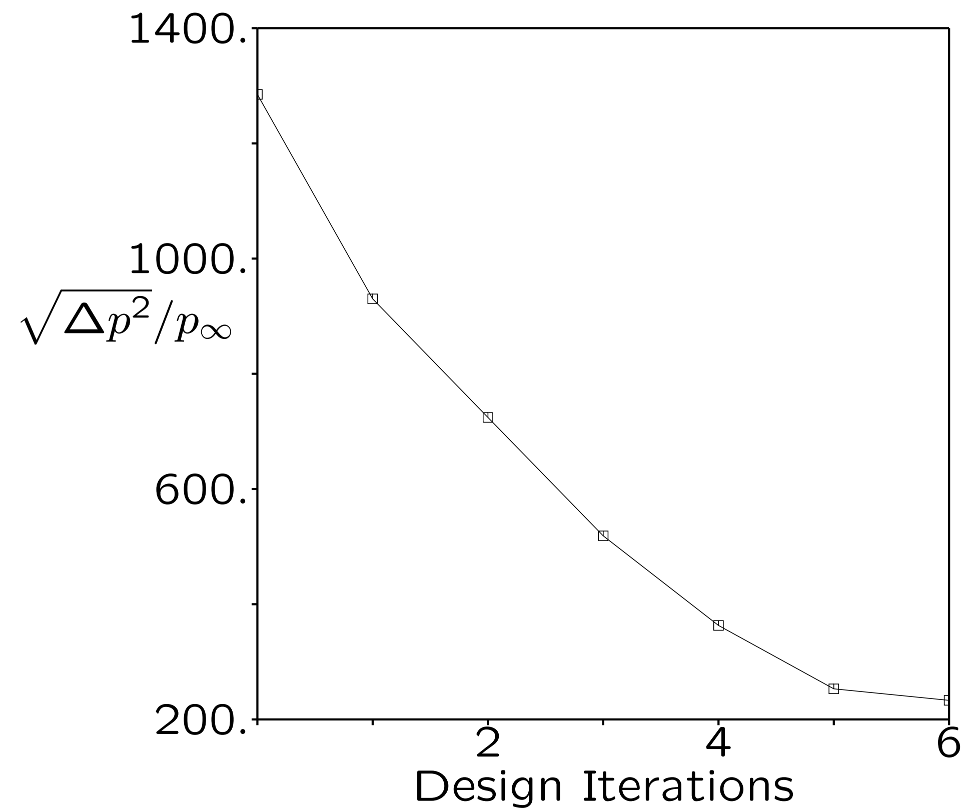
F6 Nacelle installation optimisation: Setup

- Impose pressure distribution of a clean wing as target for a wing with nacelle.
- Ma 0.7, Navier-Stokes (laminar), 2.1 M elements.
- Minimise the pressure difference by changing the wing profile.
- 360 design variables, Hicks-Henne bumps with 3 degrees of freedom in 120 control points, 2/3 of which adjust the wing thickness, 1/3 to adjust the chord length.

Surface Pressure on clean F6 wing



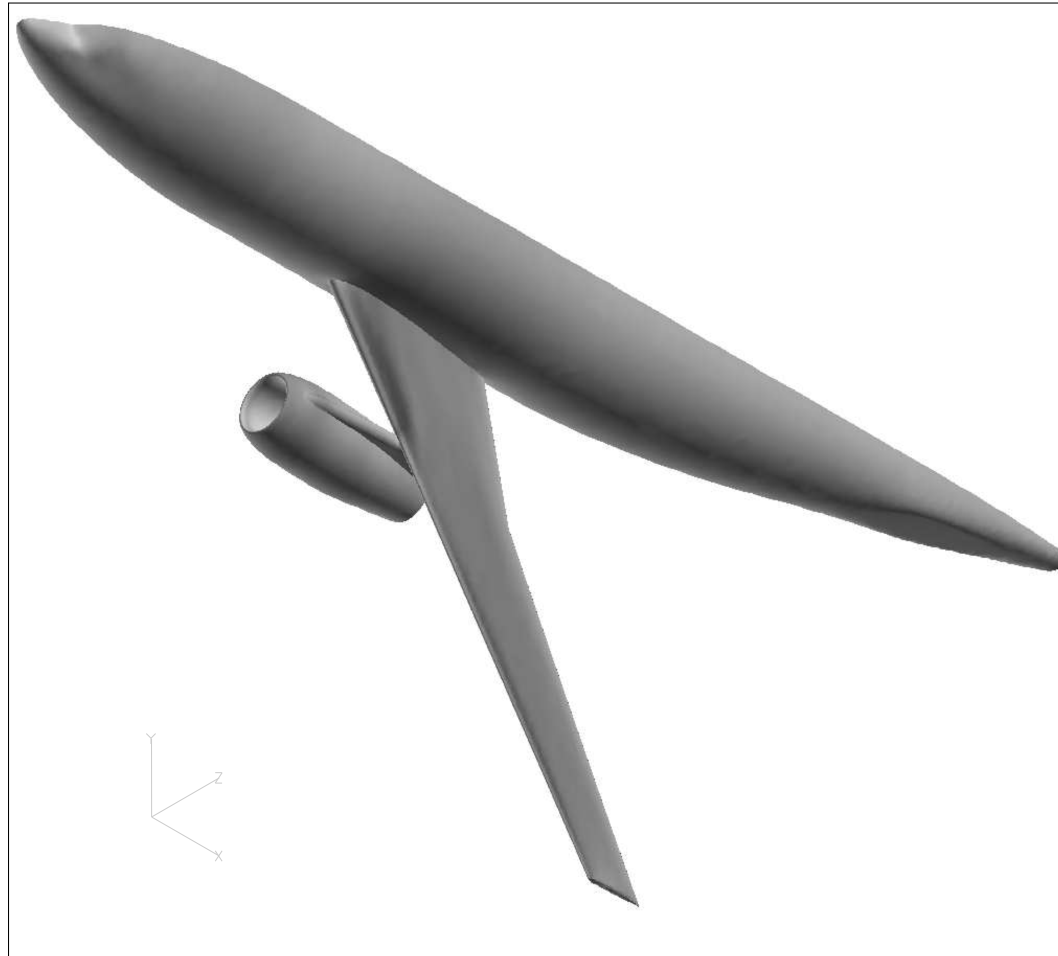
Convergence of the Optimisation



F6 Inverse Design Results

baseline target optimised

Pressure field on an optimised DLR F6



Section IV: Improving the Convergence of the Desig

- The previous examples used a very simple iterative scheme for the design:
  1. converge the primal (rather well, $10^{-}5$),
  2. converge the adjoint (still to, say, $10^{-}3$),
  3. perform a design step (quasi-Newton),
  4. stop if descent or curvature conditions on the cost function are met,
  5. otherwise recompute the primal.
- Needs, say, 50 design iterations.
- Each design iteration requires some 5-10 point evaluations of primal and adjoint.
- This results in a design costing 2-3 orders of magnitude more than a CFD calculation.

Multigrid to Improve Convergence

- Use Multigrid for primal and adjoint, cycle first on primal (Jameson).
- Converge primal and adjoints less.
- No authoritative experience, yet, whether the additional stability through multigrid sufficiently enhances robustness.
- Our implementation in 2D converges inverse design cases in around 5-10 times the cost of a CFD calculation.

Multigrid for the Design

- Use multiresolution to reduce the number of design variables on coarser meshes.
- Needs careful construction of the prolongation and restriction operators: converged fine grid solution must result in zero prolongation (Nash).
- Catch 1: Requires fully coupled iteration of primal, adjoint and design. Preconditioning?
- Catch 2: Problems with partially converged primals and adjoints. Use many small design steps with an inexpensive algorithm.
- Catch 3:
  - The cost of the source term in $v^T f$ is O(1 it), proportional to the number of design variables.
  - It needs to be recomputed at every design evaluation.
  - This could become very expensive.

Outlook

- Research in fully coupled algorithms for primal, adjoint and design is ongoing (Sachs, Gauger).
- Sufficiently converged adjoints for sufficiently accurate and smoothly varying gradients might indeed be provided with multigrid methods.
- There are a number of ways to reduce the cost of evaluating f, e.g.
  - considering only the dominant boundary terms (Mohammadi) and
  - limiting the mesh smoothing and hence the area where f is non-zero to a small part of the domain.