Differentiating a Time-dependent CFD Solver

Presented to *The AD Workshop, Nice, April 2005*

Mohamed Tadjouddine & Shaun Forth
Engineering Systems Department
Cranfield University (Shrivenham Campus)
Swindon SN6 8LA, UK

Ning Qin
Department of Mechanical Engineering
Sheffield University
Sheffield S1 3JD, UK

Work funded by EPSRC under Grant GR/R85358/01 AD2CompEng - Automatic Differentiation and Adjoints Applied to Computational Engineering

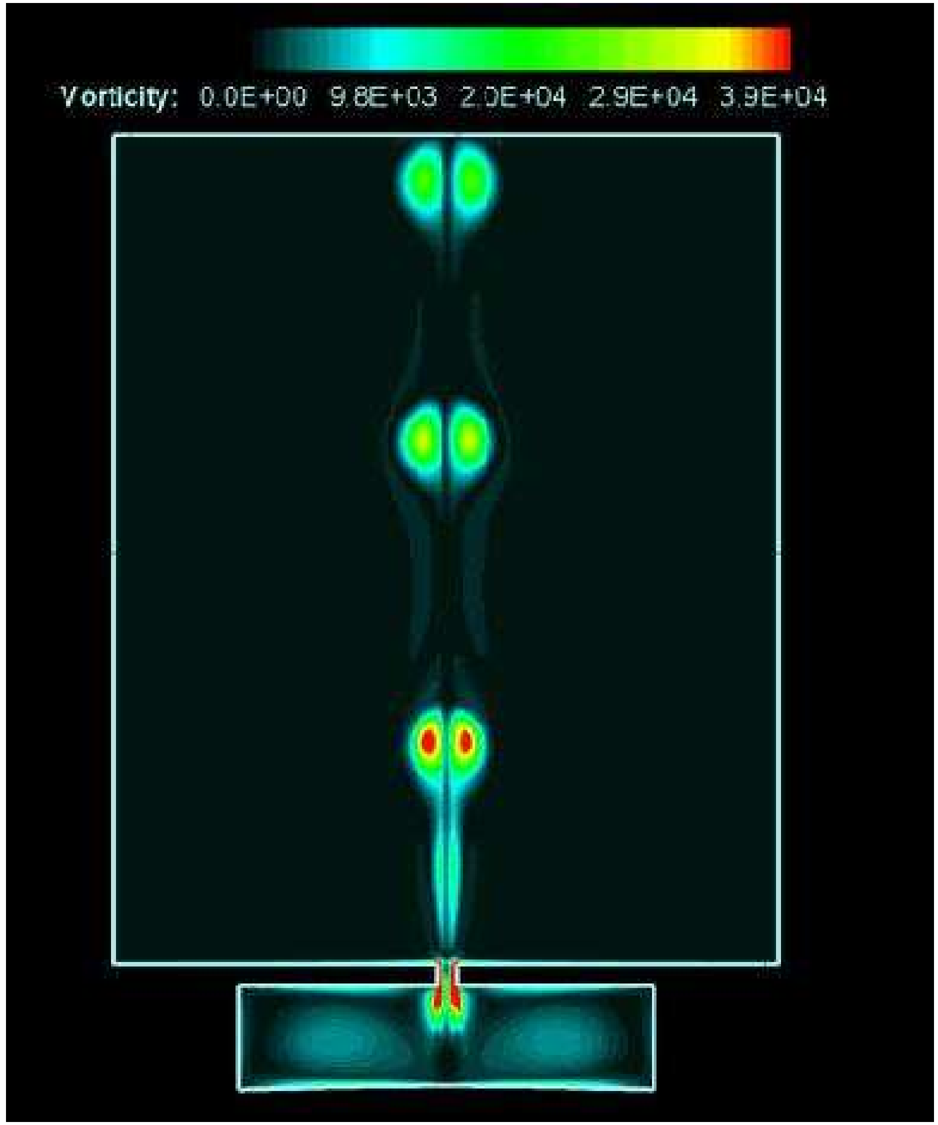
Introduction

- Aim: Tangent and Adjoint of a numerical code simulating a $2D$ model of a synthetic jet actuator.

- *Synthetic jet actuators* [GWP$^+$03]
  Synthetic jet actuators are small scale devices generating a jet-like motion by oscillating fluid in a chamber connected to the air flow via an orifice.

- *Possible Applications*
  - Embed into aircraft wing to control flow separation
  - Propulsion system for microfluid systems

Synthetic Jet



Mesh Movement Algorithm

- Time-dependent Navier-Stokes eqn.s on a moving mesh.
- Mesh movement for interior point $\mathbf{x}_i^{new} = \mathbf{x}_i + \Delta\mathbf{x}_i$ governed by forced boundary motion smoothed into interior mesh,

$$\Delta\mathbf{x}_i = \frac{1}{D_i} \sum_{j \in \text{Nbr}(i)} \frac{1}{|\mathbf{x}_i - \mathbf{x}_j|} \Delta\mathbf{x}_j, \text{ with } D_i = \sum_{j \in \text{Nbr}(i)} \frac{1}{|\mathbf{x}_i - \mathbf{x}_j|},$$

repeated 50 times - Gauss-Seidel smoothing/linear solve.

The Time-Dependent CFD Solver

- Finite-volume semi-discretisation of N-S equations for cell i,

$$\frac{\partial V_i \mathbf{q}_i}{\partial t} = \mathbf{R}_i(\mathbf{q}, \mathbf{x}).$$

- Backward Euler gives nonlinear system for $\mathbf{q}^{n+1}$.

$$\begin{aligned} \frac{V_i^{n+1}\mathbf{q}_i^{n+1} - V_i^n \mathbf{q}_i^n}{\Delta t} &= \mathbf{R}_i(\mathbf{q}_i^{n+1}, \mathbf{x}^n) \\ 0 &= \mathbf{R}_i(\mathbf{q}_i^{n+1}, \mathbf{x}^n) - \frac{V_i^{n+1}\mathbf{q}_i^{n+1} - V_i^n \mathbf{q}_i^n}{\Delta t} \end{aligned}$$

- Introduce pseudo-timestepping with pseudo-time τ,

$$V_i^{n+1}\frac{\partial \mathbf{q}_i^{n+1}}{\partial \tau} = \mathbf{R}_i(\mathbf{q}_i^{n+1}, \mathbf{x}^n) - \frac{V_i^{n+1}\mathbf{q}_i^{n+1} - V_i^n \mathbf{q}_i^n}{\Delta t}.$$

- Iterate to convergence using low-storage 4-stage Runge-Kutta scheme.

Schema of the Numerical Code

```
Read in mesh geometry X (nodes, cells, faces)
Initialise flow variables and boundary conditions
Read in design variables designvars : a, p
Read in number of time-steps N and celopt
Set F = 0
For i from 0 to N
 Move the mesh boundary using a sin scheme a sin(pX + φ)
 Update interior mesh then cell and face information
 While (not converged) !  FIXPOINT
  Converge the flow variables using a RK4 solve:
   Compute residual R = r(Q, j) for each cell j
   Update the flow variables Q using R
 End While
 Update F = F + Σ_{i=2,3} Q(i, celopt)^2
EndDo
```

Fortran 95 Features of the code

- The input code (4500 loc) uses dynamic allocation, modules, derived types and array operations
- A derived type example

```
type bound_type !define the boundary structure
 integer(2)::uns !unsteady flag
 integer(2)::dim !dimensional or not(1=Yes)
 integer(2)::var !
 real(8), allocatable::bQ(:,:)!primitive Q
 character::TP*80 !type,
 character*32::extra(2)!store extra information.
end type bound_type
```

Fortran 95 Features of the code

- A Module Example

```
module mesh_info
 use prop
 ...
 integer(4)::nodenum !  number of nodes
 integer(4)::cellnum !  number of cells
 integer(4)::facenum !  number of faces
 integer(2)::nthread !  number of threads
 type(node_type), allocatable::node(:)  !node set
 type(cell_type), allocatable::cell(:)  !cell set
 type(face_type), allocatable::face(:)  !face set
 type(thre_type), allocatable::thre(:)  !thread set
end module mesh_info
```

- To differentiate this CFD solver, we used the AD tools TAF and TAPENADE.

Code Preparation

- TAF forward worked and gave consistent results with FD.
- TAF generated adjoint used to blow up at runtime (?)
- To further investigate the adjoint, we cleaned up the original code by using a *sed* script:
  - real(8) $\longrightarrow$ double precision (Portability)
  - integer(4) $\longrightarrow$ integer
  - Dynamic Allocation $\longrightarrow$ Static Allocation
  - Module $\longrightarrow$ Common Block
  - Derived Type $\longrightarrow$ set of arrays

First Results

- Mesh size: 654 nodes, 582 cells, 1235 faces
- 2 independents (period & amplitude) and 1 dependent (kinetic energy per cell volume = 572.0564 for this run).

Method	Gradient		# Sig. F
TAF(fwd)	-3230806.74033	54833.2739837	11
TAF(rev)	-2990768.62366	51687.4932185	1
TAPENADE(fwd)	-3230806.74031	54833.2739834	11
TAPENADE(rev)	-3230806.74033	54833.2739835	11

- DEBUG: Can AD tools adopt *optimisation options* as compilers do?

Potential Debug Problem

- Consider an array partially overwritten [Ralf Giering]

```
subroutine incompletarray(bval,ff,nx,ny)
double precision::ff(nx,ny), ...  !declarations
ff = 2.d0
call boundary(bval,ff,nx,ny)!partially overwrites ff
end subroutine incompletarray
```

- By default, TAF adjoint code will look as follows:

```
subroutine adincompletarray(bval,adbval,ff,adff,nx,ny
!  declarations
call boundary (bval,ff,nx,ny)
!Recomputation of ff is wrong
call adboundary(adbval,adff,nx,ny)
end subroutine adincompletarray
```

- TAF assumes `boundary` completely overwrites `ff` so recomputation algorithm omits `ff=2.d0` line and `ff` has incorrect value.

Some Observations

- TAF adjoint, by default uses a recomputation strategy but provides a fair tradeoff off between storage/recomputation via directives.
- TAPENADE adjoint provides a recursive checkpointing strategy performed at subroutine levels to tradeoff off between storage and recomputation.
- TAPENADE's Stack may be insufficient in terms of memory requirement.
- We have coded Fortran 95 taping routines with RAM buffer (module array) and local disk files using direct access.

Exploiting Code Insights

- Fixed Point Iteration (Rule 22 of [Gri00, p. 299]): Fixed point iterations can and should be adjoined by recording only single steps.

Method	CPU(∇F) (s)	Tape_size	$\frac{\text{CPU}(\nabla F)}{\text{CPU}(F)}$
TAPENADE(fwd)	69	—	3.8
TAPENADE(rev)	7755	1.585 GB	484.6
TAPENADE(rev,FP)	5751	0.182 GB	319.5
TAPENADE(rev,FP,Par.)	5715	0.180 GB	317.5

- The gradient $\nabla F = \left[\begin{array}{cc} -3230835.9523 & 54833.2466 \end{array} \right]$ (using the same number of iterations as for the forward pass) is up to 6 significant digits as compared to the gradient using the mechanical adjoining used by TAPENADE.

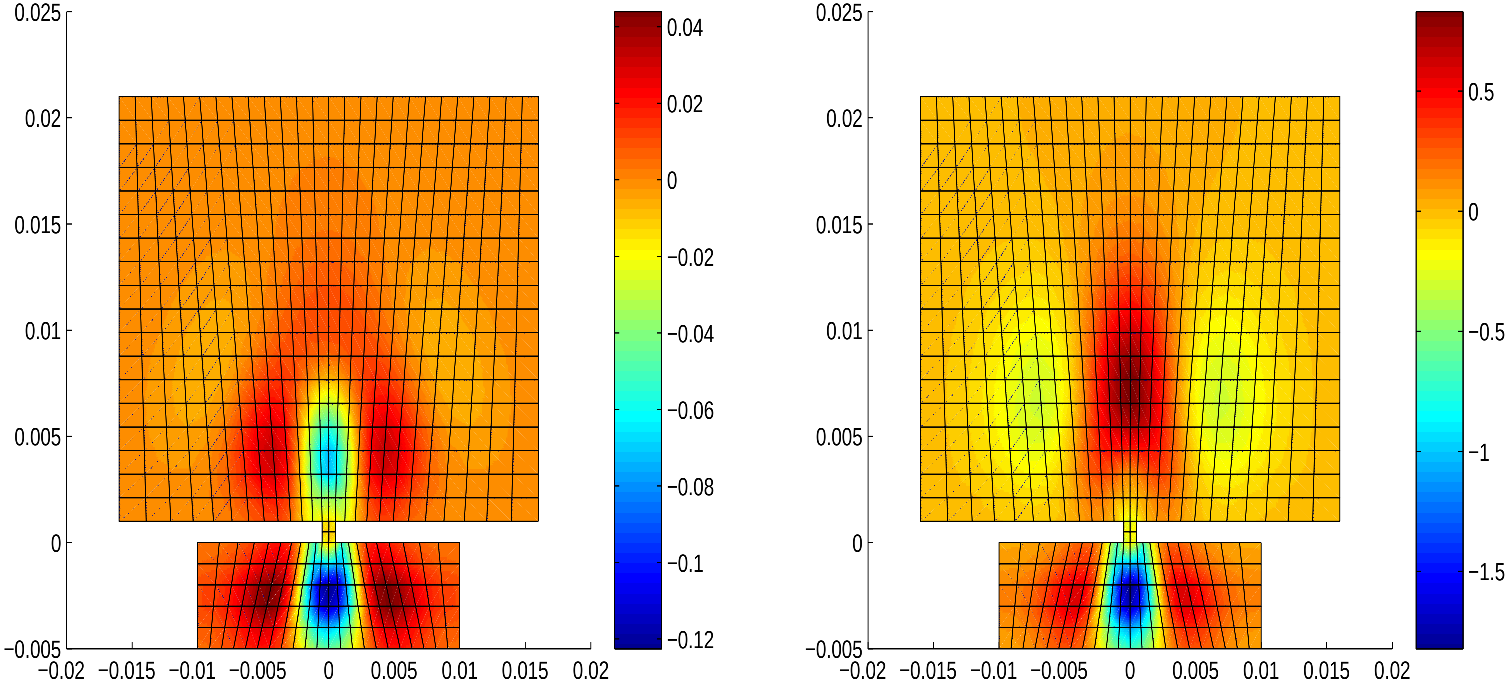
Optimisation

- Start Jet Animation
- Given a fixed amount of work $C(\mathbf{x})$ that the system cannot exceed, the objective is to maximise the kinetic energy $F(\mathbf{x})$ from a nominated cell in the upwards movement of the jet.

$$\max_{\mathbf{x}} F(\mathbf{x})$$
$$C(\mathbf{x}) \leq 1$$
$$10^{-3} \leq p \leq 10^{-1} \quad \texttt{!p is the period}$$
$$10^{-2} \leq a \leq 10^{-1} \quad \texttt{!a is the amplitude}$$

	Initial Guess		Optimun		
Method	Period	Ampl.	Period	Ampl.	Iter.
FMINCON	1.D-2	1.D-2	0.46D-2	0.45D-1	15
FMINCON(AD)	1.D-2	1.D-2	1.54D-2	1.05D-1	11

Velocity Profile at Convergence



On the Left [Right] is represented the upstream velocity profile of the jet before [after] the optimisation.

Concluding Remarks

- AD Adjoint mode performs better for applications with fairly large number of independents.
- For a given constraint, it is possible to choose the frequency and the amplitude of the fluid oscillation so as to maximise the kinetic energy of the jet's upwards movement.
- Improvements for the adjoint calculation (Fixpoint Iteration, Parallel Loops [HFH01]).
- Increase the number of the design variables in order to make the generated adjoint competitive.
- Run the adjoint for a finer mesh with (big number of nodes e.g., 20,000).
- It is safer to compile and run the original code on different platforms prior to differentiation!

References

References

[Gri00] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, Penn., 2000.

[GWP+03] Quentin Gallas, Guiquin Wang, Melih Papila, Mark Sheplak, and Louis Cattafesta. Optimization of synthetic jet actuators. *AIAA paper*, (0635), 2003.

[HFH01] Laurent Hascoët, Stefka Fidanova, and Christophe Held. Adjoining independent computations. In George Corliss, Christèle Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann, editors, *Automatic Differentiation: From Simulation to Optimization*, Computer and Information Science, chapter 35, pages 285–290. Springer, New York, 2001.