Use of AD in elsA CFD solver

Sébastien Bourasseau

elsA Team

24 January 2019

Journée DA - Algorithmic Differentiation Workshop

Institut de Physique du Globe, Paris, France

ONERA

THE FRENCH AEROSPACE LAB

Content

The elsA CFD solver [1]

Airbus/Safran/ONERA Cooperation Agreement for elsA development

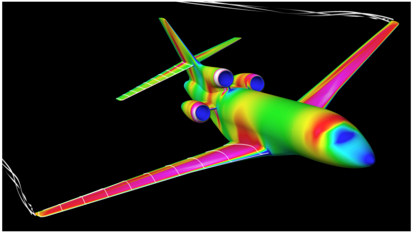
Multi-purpose CFD simulation platform

- Internal and external aerodynamics
- From low subsonic to high supersonic
- Perfect gases or real gases at equilibrium
- Compressible 3-D Navier-Stokes equations
- Moving deformable bodies
- Calculation of sensitivities
- Design optimization
- Aeroelasticity in elsA or with CFD/CSM coupling

1. L. Cambier, S. Heib, S. Plot. The ONERA elsA CFD software : input from research and feedback from industry, Mech.Ind (2013)

The elsA CFD solver
Design and implementation

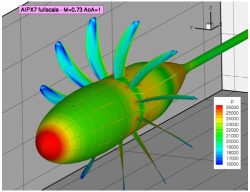
- Object-Oriented
- Kernel in C++/Fortran
- User interface in Python
- Python-CGNS interface for CGNS extraction, coupling with external software
- CPU and parallel efficiency on a large panel of computer platforms

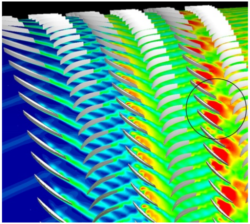


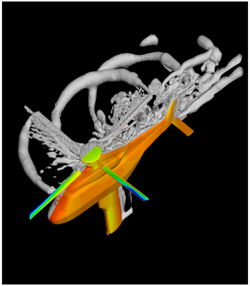
The elsA CFD solver

Applications

- Aircraft
- Helicopters
- Turbomachinery
- CROR
- Missiles
- Launchers
- Wind turbines
- Steam turbines
- ...







Content

Our specific problems
Reminder

- Fixed point of Navier-Stokes equations

$$R(W, X) = 0$$

- Implicit discrete Navier-Stokes equations

$$\left(\frac{\Omega}{\Delta t} + \frac{\partial R}{\partial W} \right) \delta W = -R(W)$$

- With :
  - W : aerodynamic field
  - X : mesh
  - Ω : volume
  - Δt : time step
  - R : explicit residual (flux balance + source terms) supposed C^1
- $n_{eq} \times n_{cells}$ sparse linear system

Our specific problems

CFD equations to resolve

- Exact implicit equations

$$\left(\frac{\Omega}{\Delta t} + \underbrace{\boxed{\frac{\partial \mathcal{R}}{\partial W}\Big).\boxed{\Delta W}}}_{\text{Matrix-vector produit}}\right. = \mathcal{R}(W, X)$$

Test vector

- Linearized equations

$$\mathcal{R}\big(W(\alpha), X(\alpha)\big) = 0 \rightarrow \boxed{\frac{\partial \mathcal{R}}{\partial W}\boxed{\frac{\partial W}{\partial \alpha}}} = -\boxed{\frac{\partial \mathcal{R}}{\partial X}\boxed{\frac{\partial X}{\partial \alpha}}}$$

Data

- Adjoint equations

$$\mathcal{R}\big(W(\alpha), X(\alpha)\big) = 0 \rightarrow \boxed{\left(\frac{\partial \mathcal{R}}{\partial W}\right)^T \boxed{\Lambda}} = -\boxed{\left(\frac{\partial J_n}{\partial W}\right)^T .\boxed{I}}$$

Our specific problems
CFD equations resolution

- GMRES[2] : iterative steps
  - Arnoldi basis of Krylov space computation
    - Arnoldi basis initialisation

      $g_0 = \boxed{A \times x_0} - b$

      $v_1 = \dfrac{1}{\|g_0\|} g_0$
    - $(j+1)^{th}$ vector of Arnoldi basis construction

      $w = \boxed{A \times v_j}$

      for $i = 1$ *to* j **do**

      $\alpha_i = (w.v_i)$

      $w = w - \alpha_i v_i$

      end for

      $v_{j+1} = \dfrac{1}{\|w\|} w$
  - QR factorization of Hessenberg matrix
  - Least squares method
- Block LU preconditioners

2. Y. Saad, Iterative methods for sparse linear systems. Vol. 82, SIAM (2003)

Our specific problems

Conclusions

- Need **matrix-vector** product with a Jacobian matrix
- Need Jacobian matrix in preconditioner

→ Need an AD tool adapted to our needs

Content

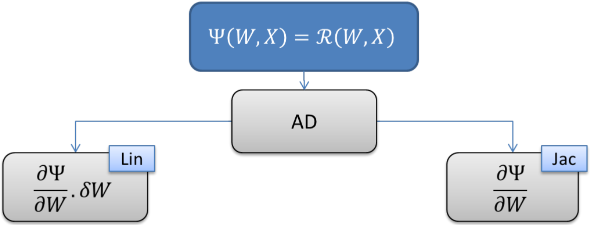
Use of AD in elsA

Vocabulary

- Let Ψ be a $\mathbb{R} \to \mathbb{R}$ application
- Direct computation : $x \to \Psi(x)$
- Linearized computation : $(x, \delta x) \to \left(\frac{\partial \Psi}{\partial x}\right) \delta x$
- Adjoint computation : $(x, \delta x) \to \left(\frac{\partial \Psi}{\partial x}\right)^T \delta x$

Use of AD in elsA

Direct computation



- GMRES

Linear system: $A.x = b$

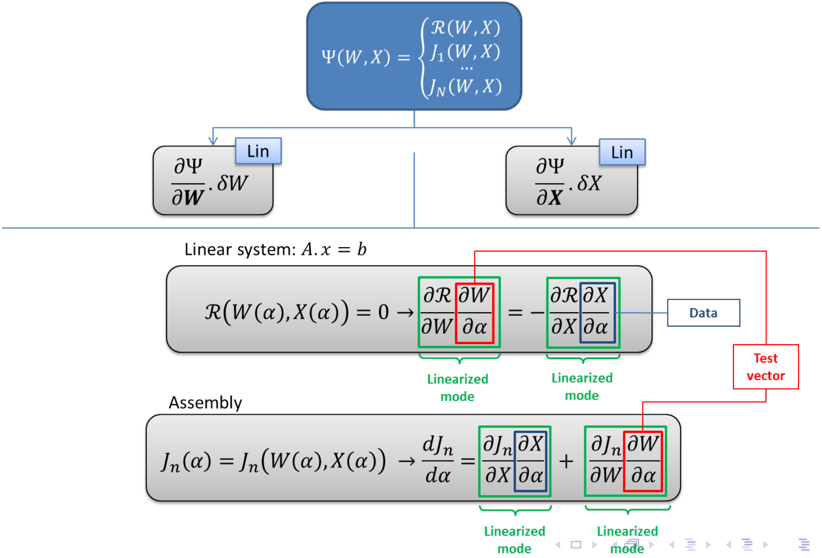
$$\underbrace{\left(\frac{\Omega}{\Delta t} + \frac{\partial \mathcal{R}}{\partial W}\right).}_{\text{Matrix-vector product}} \underbrace{\boxed{\Delta W}}_{\text{Test vector}} = \mathcal{R}(W,X)$$

- Block LU preconditioner

Linear system: $A.x = b$

$$\underbrace{\left(\frac{\Omega}{\Delta t} + \frac{\partial \tilde{\mathcal{R}}}{\partial W}\right)}_{\text{« Multi » linearized mode} + \tilde{\mathcal{R}} \text{ written in BSR format o(1)}} . \Delta W = \mathcal{R}(W,X)$$

Use of AD in elsA
Linearized computation



Use of AD in elsA
Adjoint computation

$$\Psi(W,X) = \begin{cases} \mathcal{R}(W,X) \\ J_1(W,X) \\ \cdots \\ J_N(W,X) \end{cases}$$

Adj

$$\left(\frac{\partial \Psi}{\partial \boldsymbol{W}}\right)^T.\Lambda$$

Adj

$$\left(\frac{\partial \Psi}{\partial \boldsymbol{X}}\right)^T.\Lambda$$

Linear system: $A.x = b$

$$\mathcal{R}(W(\alpha),X(\alpha)) = 0 \rightarrow \underbrace{\boxed{\left(\frac{\partial \mathcal{R}}{\partial W}\right)^T \boxed{\Lambda}}}_{\text{Adjoint mode}} = -\underbrace{\boxed{\left(\frac{\partial J_n}{\partial W}\right)^T.\boxed{I}}}_{\text{Adjoint mode}}$$

Test vector

Assembly

$$J_n(\alpha) = J_n(W(\alpha),X(\alpha)) \rightarrow \begin{cases} \dfrac{dJ_n}{dX} = \overbrace{\boxed{\boxed{I}.\dfrac{\partial J_n}{\partial X}}}^{\text{Adjoint mode}} + \overbrace{\boxed{\boxed{\Lambda^T}\dfrac{\partial \mathcal{R}}{\partial X}}}^{\text{Adjoint mode}} \\ \dfrac{dJ_n}{d\alpha} = \underbrace{\boxed{\dfrac{\partial J_n}{\partial X}\boxed{\dfrac{\partial X}{\partial \alpha}}}}_{\text{Linearized mode}} \end{cases}$$

Data

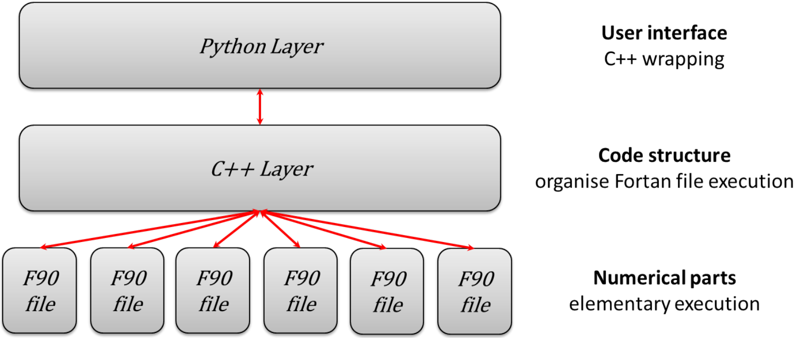
Content

Choice of an AD tool

Context

- Thinking about a new software architecture for elsA
- Needs in elsA
  - Linearized mode for :
    - Implicit resolution
    - Shape optimization
    - Stability
  - Adjoint mode for :
    - Shape optimization
    - Goal-oriented mesh adaptation
    - Sensibility
- New software architecture must be adapted to new hardware architecture
  - Preserve HPC layer
    - Domain decomposition
    - Preserve directive or pragma for vectorisation for example

Choice of an AD tool
The elsA software architecture



- Remark : F90 files could be replaced by C++ files if needed

Question

→ So, which AD tool to use ?

Choice of an AD tool

Pure symbolic differentiation (SD) on discrete equations

- Having the mathematical functional allows straightforward differentiation
- But functionals not expressed in a purely mathematical way ! (loops, conditional, stencil operation, ...)

→ Need to reconstruct the entire graph structure (AST = mathematics + loops + conditional + stencil operation + ...)

→ SD can't be a solution for our applications

Operators overloading (OO)

- Works quickly if the code is in C++ template
- Automatic
- Need a tape machine for adjoint mode

→ OO could be a solution for our applications

Choice of an AD tool

Source code transformation (SCT)

- Applicable on Fortran or C
- Not fully automatic
- Readable generated code
- Preserve directive and pragma

→ SCT could be a solution for our applications

Conclusion

→ We have to test OO and SCT for our applications

Tools retained for comparison

- CoDiPack[3] (OO) and Tapenade[4] (SCT)

3. https ://www.scicomp.uni-kl.de/codi/
4. L. Hascoët, and V. Pascual. The Tapenade automatic differentiation tool : Principles, model, and specification, ACM Trans. Math. Softw. 39(3) : 20 (2013)

Choice of an AD tool

AD tools comparison

- Two test-cases
  - Test 1 : finite differences (order 2)
  - Test 2 : Roe flux
  - Mesh : $256 \times 256 \times 24$
  - Number of iterations : 200
- Criteria
  - CPU cost
  - Memory cost
- Computations
  - Direct : $x \to \Psi(x)$
  - Linearized : $(x, \delta x) \to \left(\frac{\partial \Psi}{\partial x}\right) \delta x$
  - Adjoint : $(x, \delta x) \to \left(\frac{\partial \Psi}{\partial x}\right)^T \delta x$

  $\to$ Strictly indentical numerical results (between Tapenade and CoDiPack)

Choice of an AD tool

AD tools comparison

- Comparison on direct computation

	Direct (C++)		Direct (Fortran)	
	Time (s)	Mem (MB)	Time (s)	Mem (MB)
FDO2	0.41	16	0.39	16
Roe	5.38	120	5.37	120

- Comparison on linearized computation

	Linearized (C++)		Linearized (Fortran)	
	Time (s)	Mem (MB)	Time (s)	Mem (MB)
FDO2	0.49	32	0.50	32
Roe	5.20	184	6.20	184

- Partial conclusion
  - Similar CPU and memory costs

  → Can not choose between SCT and OO for those applications

Choice of an AD tool

AD tools comparison

- Comparison on adjoint computation

	Adjoint (C++)		Adjoint (Fortran)		CoDiPack vs Tapenade	
	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time	Mem
FDO2	3.09	108	0.58	32	4x	3x
Roe	39.90	1930	9.42	184	4x	10.5x

- Partial conclusion
  - Memory overhead from direct to adjoint :
    - CoDiPack : 16x
    - Tapenade : 1.5x
  - CPU and memory costs are in favor of SCT

→ Need to verify those conclusions on a representative CFD case

Choice of an AD tool

Representative CFD test-case

- SU2[5]/CoDiPack on ONERA M6 wing (582 752 nodes)
  - Direct computation : 3.3 GB
  - Adjoint computation : 21.7 GB
  - → Factor about 7 between direct and adjoint in terms of memory
  - In agreement with N. Gauger communication[6]
- elsA/Tapenade on Taylor Green Vortex (7 189 057 nodes)
  - Direct computation : 10.6 GB
  - Adjoint computation : 15.6 GB
  - → Factor about 1.5 between direct and adjoint in terms of memory

5. https ://su2code.github.io
6. T. Albring, N. Gauger, M. Sagebaum, B. Zhou, AD-based Discrete Adjoints in SU2, 1st SU2 Developers'Meeting, TU Delft (5-6 September 2016)-
https :su2code.github.io/documents/su2_dev_gauger.pdf

Choice of an AD tool

Conclusion

- For our CFD adjoint applications, OO is too expensive in memory
- Explanation : big sparse problem (over 1 billion points)
- Problem of OO : Tape machine management
  - Must store each operation
  - Must store each variable (global and local)
  - Loose loops structure
- Vectorization is an advantage of SCT

→ We choose Tapenade for AD in elsA

Content

Collaboration with Tapenade

Preservation of compiler directives

- Compiler directives preserved in differentiated modes

→ Preservation of HPC layer

→ Equivalent CPU performances in elsA between direct, linearized and adjoint modes

Dynamic to static trajectory for reverse mode

- Replace stack by temporary variables

→ Vectorization possible

→ Performance gain

	CPU Elapsed/NbCells/NbIterations
Direct	1.12 μs
Linearized	1.61 μs
Adjoint with static trajectory	2.35 μs
Adjoint with dynamic trajectory	3.38 μs

Content

Some results with elsA and Tapenade

Duality between linearized and adjoint modes

- Duality test : principle

$$\forall(\lambda, \delta W) : \underbrace{\left(\lambda \frac{\partial R}{\partial W}\right)}_{\text{AdjCompute}} \underbrace{\delta W}_{\text{TestVector}} = \underbrace{\lambda}_{\text{TestVector}} \underbrace{\left(\frac{\partial R}{\partial W} \delta W\right)}_{\text{LinCompute}}$$

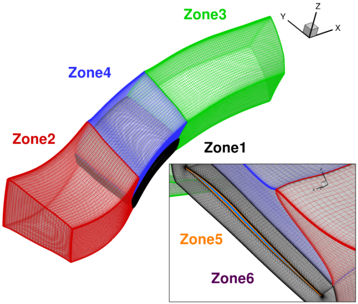
with :
$\frac{\partial R}{\partial W}$: *jacobian matrix* $(n_{eq} \times n_{cells}, n_{eq} \times n_{cells})$
λ : *adjoint vector* $(1, n_{eq} \times n_{cells})$
δW : *linearized vector* $(n_{eq} \times n_{cells}, 1)$

- Duality verification
  - Euler NACA0012 (unstructured mesh)
  - Order 1 : $\epsilon_{error} = 1.32977e^{-15}$
  - Order 2 : $\epsilon_{error} = 4.67519e^{-15}$
  - Same results in parallel

Some results with elsA and Tapenade

CFD computation : NASA rotor 37

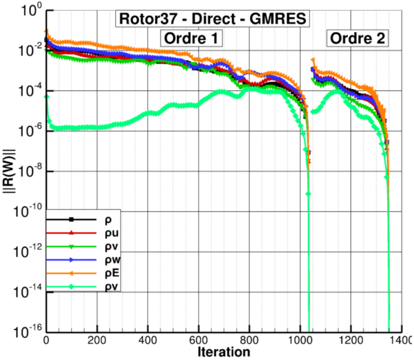
- Unstructured mesh
- Roe flux with order 1 (**null** limiter) and order 2 (**valbada** limiter)
- GMRES implicit resolution
  - Based on linearized mode
  - ILU(0) preconditioner built by AD
  - FGMRES([60, 10^{-3}], [60, 10^{-3}])
  - Adaptative CFL with $CFL_{Init} = 1$
- Linearisable Spalart-Allmaras
- Adjoint computation with objective function $J(W) = F(entropy)$
- GMRES for adjoint resolution



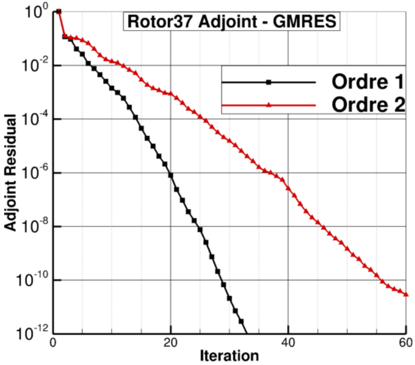
Some results with elsA and Tapenade

CFD computation : NASA rotor 37

- Stationary convergence

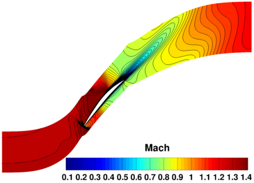


- Adjoint convergence

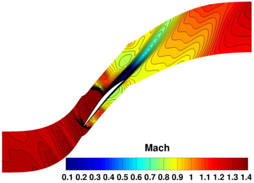


Some results with elsA and Tapenade
CFD computation : NASA rotor 37

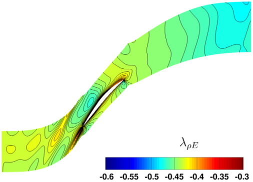
- Mach (order 1)



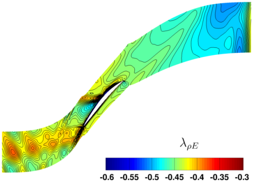
- Mach (order 2)



- $\lambda_{\rho E}$ (order 1)



- $\lambda_{\rho E}$ (order 2)



Some results with elsA and Tapenade

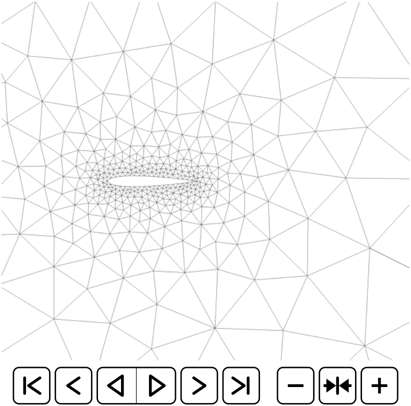
Other computed configurations

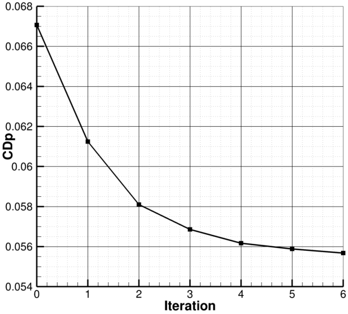
- Naca0012
- Axi Trans Bump
- M6 wing
- OAT15A
- CREATE compressor
- High Lift CRM (coarse, medium and fine)
- An helicopter compressor
- ...

Some results with elsA and Tapenade

Goal-oriented mesh adaptation

- Example on NASA NACA0012 (M=0.95 and AoA=0.) with Pointwise





Content

Conclusion

- AD is very useful
  - Less development time
  - Less debug time
- With Tapenade, all developments have been validated
  - Duality
  - Comparison with finite differences
- Good results on industrial test-cases

Thank you for your attention

Questions ?