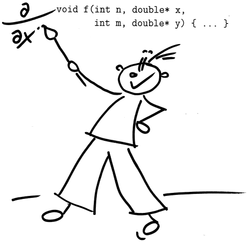
Toward First- and Higher-Order Discrete Adjoint [Flow Solvers]

Uwe Naumann



LuFG Informatik 12
Software and **T**ools for **C**omputational **E**ngineering
RWTH Aachen University, Germany
naumann@stce.rwth-aachen.de
www.stce.rwth-aachen.de

and

The **N**umerical **A**lgorithms **G**roup Ltd.
Oxford, United Kingdom
Uwe.Naumann@nag.co.uk
www.nag.co.uk

Acknowledgment

Various aspects of this presentation have seen contributions from members of the STCE team, in particular,

- Markus Beckers
- Klaus Leppkes
- Johannes Lotz
- Viktor Mosenkis
- Jan Riehme
- Niloofar Safiran
- Markus Towara.

A significant fraction of ongoing research and development at STCE (and in close collaboration with NAG) is inspired by/related to the following.

Outline

- Motivation: Algorithmic Differentiation (AD)[1][2] OF OpenFOAM
- Recall: Black-Box AD FOR numerical algorithms
- Support: dco/c++, dco/fortran
- Approach: White-Box AD OF numerical algorithms (linear solvers, nonlinear solvers, NLP solvers)
- Case Study: NAG AD Library

[1] A. Griewank and A. Walther: *Evaluating Derivatives*. SIAM, 2008.
[2] U.N.: *The Art of Differentiating Computer Programs*. SIAM, 2012.

Motivation: AD OF OpenFOAM

Given: Continuous adjoint model for ducted flows in OpenFOAM[3] allows efficient evaluation of sensitivities of dissipation / pressure loss with respect to porosity / flow resistance.

Wanted: Discrete adjoint model of OpenFOAM allowing efficient evaluation of sensitivities of any objective with respect to any set of parameters, e.g. the above.

TODO:

1. apply AD to OpenFOAM ($\rightarrow$ M. Towara)
2. compare numerical results of both approaches
3. draw conclusions

[3] C. Othmer: *A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows.* Intern. J. f. Num. Meth. in Fluids. p. 861–877, 2008.

The usual suspects ...

- $F(\mathbf{x}) = 0,\ F : \mathbb{R}^n \to \mathbb{R}^n$
  - Newton requires ∇F
  - (matrix-free) Newton-Krylov requires $< \nabla F, \mathbf{x}^{(1)} >$
- $f(\mathbf{x}) \to \min,\ f : \mathbb{R}^n \to \mathbb{R}$
  - quasi-Newton requires ∇f
  - (matrix-free) Newton-Krylov requires $< \nabla^2 f, \mathbf{x}^{(2)} >$
- $f(\mathbf{x}) \to \min$ s.t. $c(\mathbf{x}) = 0,\ f : \mathbb{R}^n \to \mathbb{R},\ c : \mathbb{R}^n \to \mathbb{R}^m$
  - Newton-Lagrange requires ∇f, $< \nabla^2 f, \mathbf{x}^{(2)} >$, $< \nabla c, \mathbf{x}^{(1)} >$, and $< \lambda, \nabla^2 c, \mathbf{x}^{(2)} >$, where λ denotes the vector of Lagrange multipliers
- ...

- Tangent-linear model (forward mode AD)

$$\mathbb{R} \ni y^{(1)} = f^{(1)}(\mathbf{x}, \mathbf{x}^{(1)}) \equiv < \underbrace{\nabla f(\mathbf{x})}_{\in \mathbb{R}^n}, \underbrace{\mathbf{x}^{(1)}}_{\in \mathbb{R}^n} > \quad \Rightarrow \quad \nabla f \text{ at } O(n)$$

- Adjoint model (reverse mode AD)

$$\mathbb{R}^n \ni \mathbf{x}_{(1)} = f_{(1)}(\mathbf{x}, y_{(1)}) \equiv < \underbrace{y_{(1)}}_{\in \mathbb{R}}, \nabla f(\mathbf{x}) > = y_{(1)} \cdot \nabla f(\mathbf{x})$$

$$\Rightarrow \quad \nabla f \text{ at } O(1)$$

- Tangent-linear/adjoint code

```
void d1_f(int n, double *x, double *d1_x,
              double &y, double &d1_y);
```

Higher-Order Algorithmic Differentiation $y = f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$

- Second-order tangent-linear model[4]

$$\mathbb{R} \ni y^{(1,2)} = f^{(1,2)}(\mathbf{x}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \equiv < \underbrace{\nabla^2 f(\mathbf{x})}_{\in \mathbb{R}^{n \times n}}, \underbrace{\mathbf{x}^{(1)}}_{\in \mathbb{R}^n}, \underbrace{\mathbf{x}^{(2)}}_{\in \mathbb{R}^n} >$$

$$\Rightarrow \quad \nabla^2 f \text{ at } O(n^2)$$

- Second-order adjoint model[5]

$$\mathbb{R}^n \ni \mathbf{x}^{(2)}_{(1)} = f^{(2)}_{(1)}(\mathbf{x}, \mathbf{x}^{(2)}, y_{(1)}) \equiv < y_{(1)}, \nabla^2 f(\mathbf{x}), \mathbf{x}^{(2)} >$$

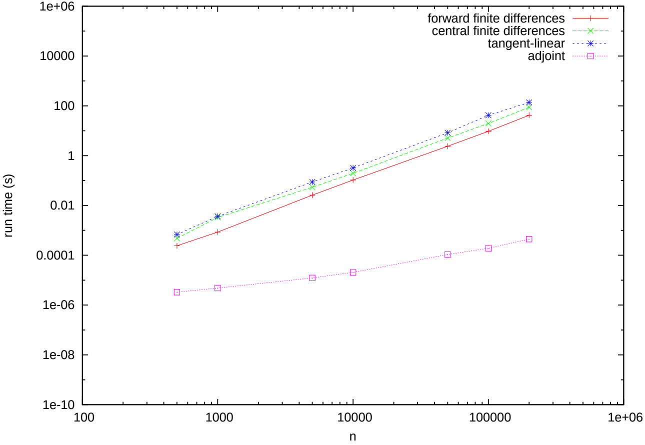
$$\Rightarrow \quad \nabla^2 f \cdot \mathbf{x}^{(2)} \text{ at } O(1) \text{ resp. } \nabla^2 f \text{ at } O(n)$$

- Higher-order tangent-linear (fofo...fof) and adjoint (fofo...for) models are derived recursively

[4] fof

[5] for(=rof=ror → symmetry, associativity))

Performance



- Source transformation (dcc, NAG Fortran compiler)

```
void a1_f(int n, double *x, double *a1_x, double &y, double &a1_y) {
  y=0; for (int i=0;i<n;i++) y=y+x[i]*x[i];
  double rd=y; y=y*y; double rcp=y; y=rd;
  a1_y=2*y*a1_y;
  for (int i=n-1;i>=0;i--)
    a1_x[i]+=2*x[i]*a1_y;
  a1_y=0; y=rcp;
}
```

- Overloading (dco/c++, dco/fortran)

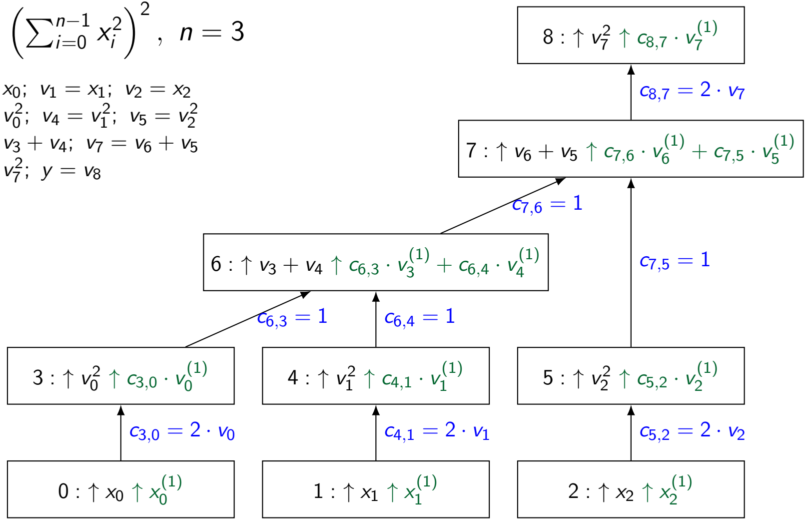
```
template<class DType>
void f(int n, DType *x, DType &y) {
  y=0;
  for (int i=0;i<n;i++) y=y+x[i]*x[i];
  y=y*y;
}
```

- Reality → Hybrid White-Box

Tangent-Linear/Forward Mode $\Rightarrow$ No Tape

$$y = \left(\sum_{i=0}^{n-1} x_i^2\right)^2, \quad n = 3$$

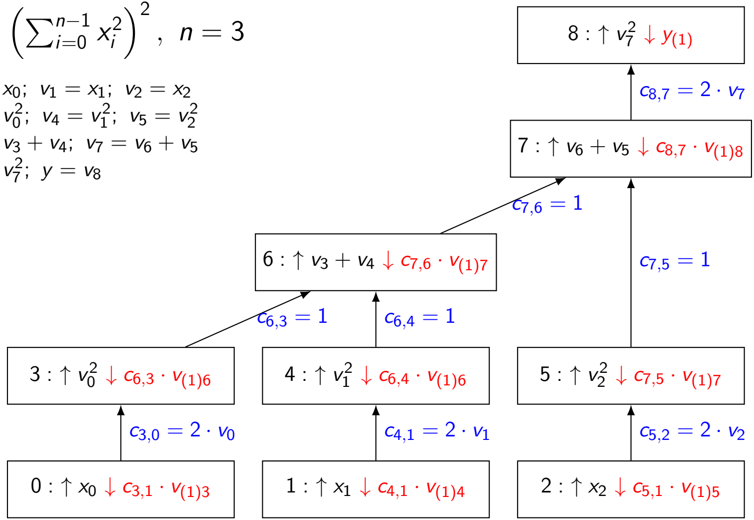
$v_0 = x_0;\ v_1 = x_1;\ v_2 = x_2$
$v_3 = v_0^2;\ v_4 = v_1^2;\ v_5 = v_2^2$
$v_6 = v_3 + v_4;\ v_7 = v_6 + v_5$
$v_8 = v_7^2;\ y = v_8$



Adjoint/Reverse Mode $\Rightarrow$ Tape

$$y = \left(\sum_{i=0}^{n-1} x_i^2\right)^2, \quad n = 3$$

$v_0 = x_0;\ v_1 = x_1;\ v_2 = x_2$
$v_3 = v_0^2;\ v_4 = v_1^2;\ v_5 = v_2^2$
$v_6 = v_3 + v_4;\ v_7 = v_6 + v_5$
$v_8 = v_7^2;\ y = v_8$

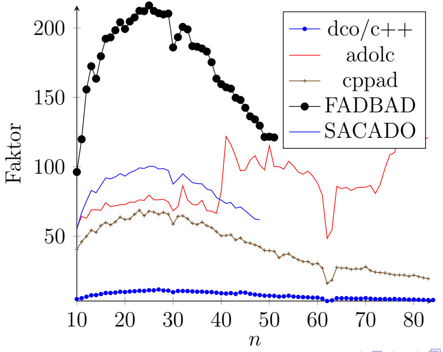


dco/c++

- expression templates for compile-time code optimization
- highly optimized internal representation
- intuitive user interface
- support for
  - first- and higher-order tangents and adjoints
  - checkpointing
  - external functions
  - parallelism (AMPI v1.0)[6]
- used for discrete adjoint of OpenFOAM ($\rightarrow$ M Towara's talk)

[6] M.Schanen, U.N., L. Hascoët, J. Utke: *Interpretative adjoints for numerical simulation codes using MPI*. ICCS 2010.

Reference problem: M. Matyka: Hydro Dynamica 3d, University of Wroclaw (3-D Navier-Stokes solver using SIMPLE scheme, single execution of black-box adjoint, 2GB memory)



Conclusion I: The Bad News

Black-box AD will probably fail on your code[7] because

- it assumes differentiability of the function and data-flow continuity of its implementation; It will fail on, e.g.,

$$y = \begin{cases} 3 \cdot x & x = 0 \\ 2 \cdot x & x \neq 0 \end{cases}$$

- it delivers first and higher derivatives with machine accuracy; Is this what you want? ($\rightarrow y = x^2 + 0.1 \cdot \sin(100 * x)$)
- it delivers (sub-)derivatives of the given implementation; Is this what you want? ($\rightarrow y = |x|$)
- it assumes availability of a sufficient amount of memory to store the variables that are required for the data flow reversal (e.g., the tape) in adjoint mode.

[7] no matter which tool you use!

Considering

- $A(\mathbf{z}) \cdot \mathbf{x} = \mathbf{b}(\mathbf{z}), \quad A \in \mathbb{R}^{n \times n}, \mathbf{x}, \mathbf{b} \in \mathbb{R}^n, \mathbf{z} \in \mathbb{R}^m,$
  (U.N. and J.Lotz: *Algorithmic Differentiation of Direct Solvers for Systems of Linear Equations*, RWTH Aachen 2012.)
- $F(\mathbf{x}, \lambda(\mathbf{z})) = 0, \quad F : \mathbb{R}^n \times \mathbb{R}^{n_\lambda} \to \mathbb{R}^n, \mathbf{z} \in \mathbb{R}^m,$
  (U.N. and J.Lotz and M.Towara: *Algorithmic Differentiation of Solvers for Systems of Nonlinear Equations*, RWTH Aachen 2012.)
- $\text{argmin}_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}, \lambda(\mathbf{z})), \quad f : \mathbb{R}^n \times \mathbb{R}^{n_\lambda} \to \mathbb{R}, \mathbf{z} \in \mathbb{R}^m,$
  (U.N. and J.Lotz and M.Towara: *Algorithmic Differentiation of Solvers for Unconstrained Convex Nonlinear Programming*, RWTH Aachen 2012.)

for example, in the context of $\min_{\mathbf{z} \in \mathbb{R}^m} g(\mathbf{z})$ requiring $\nabla g(\mathbf{z})$.
For example, $g(\mathbf{z}) \equiv \sum_{i=0}^{n-1} (x_i(\mathbf{z}) - o_i)^2$ and, hence,
$\nabla g(\mathbf{z}) = 2 \cdot \nabla_{\mathbf{z}} S(\mathbf{x}(\mathbf{z}))^T \cdot \mathbf{x}$ at the solution $\mathbf{x}$.

Black-Box AD of $A \cdot \mathbf{x} = \mathbf{b}$

- Discrete Tangent-Linear Mode

$$\mathbf{x} := S(A, \mathbf{b})$$
$$\mathbf{x}^{(1)} := S^{(1)}(A, A^{(1)}, \mathbf{b}, \mathbf{b}^{(1)}) = < \frac{\partial \mathbf{x}}{\partial A}, A^{(1)} > + < \frac{\partial \mathbf{x}}{\partial \mathbf{b}}, \mathbf{b}^{(1)} >$$

E.g., Gauss: $\text{MEM}(S^{(1)}) \sim O(n^2)$, $\text{OPS}(S^{(1)}) \sim O(n^3)$

- Discrete Adjoint Mode

$$(\mathbf{x}, \tau) := S_{\downarrow}(A, \mathbf{b})$$
$$\begin{pmatrix} A_{(1)} \\ \mathbf{b}_{(1)} \end{pmatrix} := S_{\uparrow(1)}(\tau, \mathbf{x}_{(1)}) = \begin{pmatrix} < \mathbf{x}_{(1)}, \frac{\partial \mathbf{x}}{\partial A} > \\ < \mathbf{x}_{(1)}, \frac{\partial \mathbf{x}}{\partial \mathbf{b}} > \end{pmatrix}$$

E.g., Gauss: $\text{MEM}(S_{(1)}) \sim O(n^3)$, $\text{OPS}(S_{(1)}) \sim O(n^3)$

White-Box Tangent-Linear $A \cdot \mathbf{x} = \mathbf{b}$

E.g., Continuous Tangent-Linear Gauss

$$(\mathbf{x}, L, U) := S(A, \mathbf{b})$$

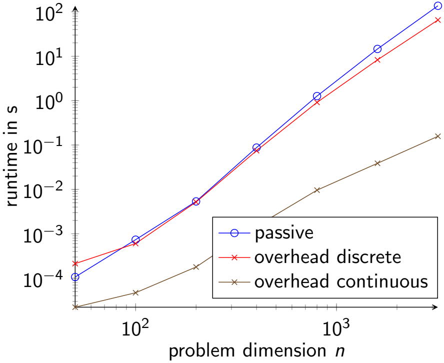
$$\mathbf{x}^{(1)} := S^{(1)}(L, U, A^{(1)}, \mathbf{b}, \mathbf{b}^{(1)}) =< \frac{\partial \mathbf{x}}{\partial A}, A^{(1)} > + < \frac{\partial \mathbf{x}}{\partial \mathbf{b}}, \mathbf{b}^{(1)} >$$

Partial differentiation of $A \cdot \mathbf{x} = \mathbf{b}$ at the solution $\mathbf{x}$ with respect to ...

- ... $\mathbf{b}$ yields $L \cdot U \cdot < \frac{\partial \mathbf{x}}{\partial \mathbf{b}}, \mathbf{b}^{(1)} >= \mathbf{b}^{(1)}$
- ... A yields $< \frac{\partial \mathbf{x}}{\partial A}, A^{(1)} >= -\mathbf{x}^T \cdot A^{(1)}$

$\text{MEM}(S^{(1)}) \sim O(n^2)$, $\text{OPS}(S^{(1)}) \sim O(n^2)$

White-Box Adjoint $A \cdot \mathbf{x} = \mathbf{b}$ Performance



White-Box Adjoint $A \cdot \mathbf{x} = \mathbf{b}$

E.g., Continuous Adjoint Gauss

$$(\mathbf{x}, L, U) = S(A, \mathbf{b}):$$

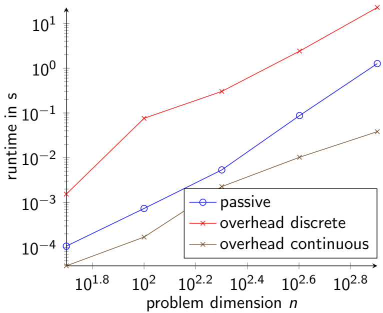
$$\begin{pmatrix} A_{(1)} \\ \mathbf{b}_{(1)} \end{pmatrix} := S_{(1)}(L, U, \mathbf{x}_{(1)}) = \begin{pmatrix} < \mathbf{x}_{(1)}, \frac{\partial \mathbf{x}}{\partial A} > \\ < \mathbf{x}_{(1)}, \frac{\partial \mathbf{x}}{\partial \mathbf{b}} > \end{pmatrix}$$

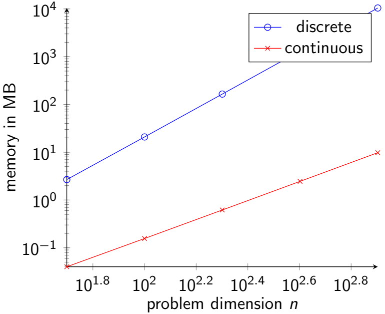
Partial differentiation of $A \cdot \mathbf{x} = \mathbf{b}$ at the solution $\mathbf{x}$ with respect to ...

- ... $\mathbf{b}$ yields $U^T \cdot L^T \cdot < \mathbf{x}_{(1)}, \frac{\partial \mathbf{x}}{\partial \mathbf{b}} >= \mathbf{x}_{(1)}$
- ... A yields $< \mathbf{x}_{(1)}, \frac{\partial \mathbf{x}}{\partial A} >= -\mathbf{b}_{(1)} \cdot \mathbf{x}^T$

$\mathrm{MEM}(S_{(1)}) \sim O(n^2)$, $\mathrm{OPS}(S_{(1)}) \sim O(n^2)$

White-Box Adjoint $A \cdot \mathbf{x} = \mathbf{b}$ Performance





Options / Issues

- supported by dco via user-defined intrinsics
- similar results apply to other (sparse) direct linear solvers, such as QR, LL^T, SuperLU, ...
- dense (rank-1) adjoint of A even in sparse case
- preliminary tests with iterative (Krylov-subspace) solvers exhibit good convergence of the continuous adjoint depending on the accuracy of the approximation of the primal solution;[8] divergence of the discrete adjoint was observed by colleagues at Argonne Ntl. Lab $\rightarrow$ work in progress[9]

[8] T. Lajewski: *Analysing the coupling of discrete and continuous adjoints for an iterative linear solver.* RWTH 2012.

[9] B. Christianson, J. Utke, S. Wild: *When AD derivatives diverge.* Unpublished draft.

Black-Box AD of $F(\mathbf{x},\lambda)=0$

- Discrete Tangent-Linear Mode

$$\mathbf{x} := S(\mathbf{x}^0,\lambda)$$
$$\mathbf{x}^{(1)} := S^{(1)}(\mathbf{x}^0,\lambda,\lambda^{(1)}) = <\frac{\partial \mathbf{x}}{\partial \lambda},\lambda^{(1)}>$$

E.g., k Newton steps + Gauss:
$\text{MEM}(S^{(1)}) \sim O(n^2)$, $\text{OPS}(S^{(1)}) \sim O(k\cdot n^3)$

- Discrete Adjoint Mode

$$(\mathbf{x},\tau) = S_{\downarrow}(\mathbf{x}^0,\lambda)\vdots$$
$$\lambda_{(1)} := S_{\uparrow(1)}(\tau,\mathbf{x}_{(1)}) = <\mathbf{x}_{(1)},\frac{\partial \mathbf{x}}{\partial \lambda}>$$

E.g., $\text{MEM}(S_{(1)}) \sim O(k\cdot n^3)$, $\text{OPS}(S_{(1)}) \sim O(k\cdot n^3)$

Partial differentiation of $F(\mathbf{x}, \lambda) = 0$ at the solution $\mathbf{x}$ wrt. λ yields

$$\mathbf{x}^{(1)} = < \frac{\partial \mathbf{x}}{\partial \lambda}, \lambda^{(1)} > = \frac{\partial \mathbf{x}}{\partial \lambda} \cdot \lambda^{(1)} = -\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}, \lambda)^{-1} \cdot \frac{\partial F}{\partial \lambda}(\mathbf{x}, \lambda) \cdot \lambda^{(1)}$$

and, hence, the solution of the linear system

$$\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}, \lambda) \cdot \mathbf{x}^{(1)} = -\frac{\partial F}{\partial \lambda}(\mathbf{x}, \lambda) \cdot \lambda^{(1)}$$

whose right-hand side is obtained by a single call of the tangent-linear version F.

E.g., $\text{MEM}(S^{(1)}) \sim O(n^2)$, $\text{OPS}(S^{(1)}) \sim O(n^3)$

White-Box (Continuous) Adjoint

$$F(\mathbf{x},\lambda)=0$$

Partial differentiation of $F(\mathbf{x},\lambda)=0$ at the solution $\mathbf{x}$ wrt. λ yields

$$\lambda_{(1)} = < \mathbf{x}_{(1)}, \frac{\partial \mathbf{x}}{\partial \lambda} > = \left(\frac{\partial \mathbf{x}}{\partial \lambda}\right)^T \cdot \mathbf{x}_{(1)} = -\frac{\partial F}{\partial \lambda}(\mathbf{x},\lambda)^T \cdot \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\lambda)^{-T} \cdot \mathbf{x}_{(1)}$$

and, hence, the solution of the linear system

$$\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\lambda)^T \cdot \mathbf{z} = -\mathbf{x}_{(1)}$$

followed by a single call of the adjoint version of F to obtain

$$\lambda_{(1)} = \frac{\partial F}{\partial \lambda}(\mathbf{x},\lambda)^T \cdot \mathbf{z}.$$

E.g., $\text{MEM}(S_{(1)}) \sim O(n^2 + \text{MEM}(F_{(1)})$, $\text{OPS}(S_{(1)}) \sim O(n^3)$

Options

- checkpointing in discrete adjoint mode ($i_{\max}$ iterations)
  - store $\mathbf{x}^i$; adjoin iterations individually in reverse order (reduces memory requirement by factor $i_{\max}$; doubles operations count)
  - use `revolve`[10] for optimal reversal scheme[11]
- semi-discrete tangent-linear / adjoint modes
  - continuous tangent-linear / adjoint linear solver, discrete remainder; see also M. Towara's talk
  - checkpointing in semi-discrete adjoint mode

---

[10] A. Griewank and A. Walther: *Algorithm 799: Revolve: An Implementation of Checkpoint for the Reverse or Adjoint Mode of Computational Differentiation*," ACM TOMS 26(1), p. 19–45, 2000.

[11] U.N. and O.Schenk, eds.: Combinatorial Scientific Computing. CRC Press 2012.

- Discrete Tangent-Linear Mode

$$\mathbf{x} := S(\mathbf{x}^0,\lambda)$$

$$\mathbf{x}^{(1)} := S^{(1)}(\mathbf{x}^0,\lambda,\lambda^{(1)}) =< \frac{\partial \mathbf{x}}{\partial \lambda},\lambda^{(1)} >$$

E.g., k Newton steps + Cholesky:
MEM$(S^{(1)}) \sim O(n^2)$, OPS$(S^{(1)}) \sim O(k \cdot n^3)$

- Discrete Adjoint Mode

$$(\mathbf{x},\tau) = S_{\downarrow}(\mathbf{x}^0,\lambda)\vdots$$

$$\lambda_{(1)} := S_{\uparrow(1)}(\tau,\mathbf{x}_{(1)}) =< \mathbf{x}_{(1)}, \frac{\partial \mathbf{x}}{\partial \lambda} >$$

E.g., MEM$(S_{(1)}) \sim O(k \cdot n^3)$, OPS$(S_{(1)}) \sim O(k \cdot n^3)$

White-Box (Continuous) Tangent-Linear $\arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x},\lambda(\mathbf{z}))$

Partial differentiation of the first-order optimality condition $\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x},\lambda)=0$ at the solution $\mathbf{x}$ wrt. λ yields

$$\mathbf{x}^{(1)} =< \frac{\partial \mathbf{x}}{\partial \lambda}, \lambda^{(1)} >= \frac{\partial \mathbf{x}}{\partial \lambda} \cdot \lambda^{(1)} = -\frac{\partial^2 f}{\partial \mathbf{x}^2}^{-1} \cdot \frac{\partial^2 f}{\partial \mathbf{x} \partial \lambda} \cdot \lambda^{(1)}$$

and, hence, the solution of the linear system

$$\frac{\partial^2 f}{\partial \mathbf{x}^2} \cdot \mathbf{x}^{(1)} = -\frac{\partial^2 f}{\partial \mathbf{x} \partial \lambda} \cdot \lambda^{(1)}$$

whose right-hand side can be computed efficiently by a single call of the second-order adjoint version of f.

E.g., $\text{MEM}(S^{(1)}) \sim O(n^2 + \text{MEM}(f_{(1)}^{(2)})$, $\text{OPS}(S^{(1)}) \sim O(n^3)$

Partial differentiation of the first-order optimality condition $\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x},\lambda)=0$ at the solution $\mathbf{x}$ wrt. λ yields

$$\lambda_{(1)} = <\mathbf{x}_{(1)}, \frac{\partial \mathbf{x}}{\partial \lambda}> = \frac{\partial \mathbf{x}}{\partial \lambda}^T \cdot \mathbf{x}_{(1)} = -\frac{\partial^2 f}{\partial \mathbf{x} \partial \lambda}^T \cdot \frac{\partial^2 f}{\partial \mathbf{x}^2}^{-T} \cdot \mathbf{x}_{(1)}$$

and, hence, the solution of the linear system

$$\frac{\partial^2 f}{\partial \mathbf{x}^2} \cdot \mathbf{z} = \mathbf{x}_{(1)}$$

followed by a single call of the second-order adjoint version of f to compute $-\frac{\partial^2 f}{\partial \mathbf{x} \partial \lambda}^T \cdot \mathbf{z}$ efficiently.

E.g., $\text{MEM}(S_{(1)}) \sim O(n^2 + \text{MEM}(f_{(1)}^{(2)}))$, $\text{OPS}(S_{(1)}) \sim O(n^3)$

E.g.,

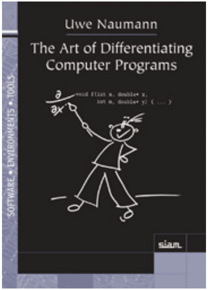
- linear solver (f04ba)
  - nag_real_gen_lin_solve (A,b ,...)
  - nag_real_gen_lin_solve_t1s (A,A_t1s,b, b_t1s ,...)
  - nag_real_gen_lin_solve_a1s (A,A_a1s,b, b_a1s ,...)
- nonlinear least-squares solver (e04gb)
  - nag_opt_lsq_deriv (F,x, user ,...)
  - nag_opt_lsq_deriv_t1s (F,x, x_t1s , user , user_t1s ,...)
  - nag_opt_lsq_deriv_a1s (F,x, x_a1s , user , user_a1s ,...)
- ... will keep us busy for a while ...

Conclusion II: The Good News

White-box AD has the potential to produce robust, efficient, and sustainable first- and higher-order tangent-linear and/or adjoint versions of your flow solver if

- you are willing to learn AD;
- you are willing to invest the required development time;
- your AD tool is flexible enough to comply with the requirements of your tailored AD solution;
- your AD tool produces efficient first-order adjoint code ($\rightarrow$ relative run time);
- your AD tool helps you to detect and exploit special structure and/or sparsity within your problem;
- the code generated by your AD tool is able to handle/exploit parallelism (OpenMP, MPI, accelerators).

To get started ...



U. Naumann:
The Art of Differentiating Computer Programs.
SIAM, 2012.

naumann@stce.rwth-aachen.de

Uwe.Naumann@nag.co.uk

Ongoing Projects

- Jurassic
- ICON
- Telemac/Sisyphe
- McCormick
- OpenFOAM
- JADE
- Computational Finance

```
#include "dco.hpp"
void f(int n, dco::t1s::type *x, dco::t1s::type &y);

void t1s_driver(int n, double *x, double &y, double *g) {
  dco::t1s::type *t1s_x=new dco::t1s::type[n], t1s_y;
  for (int i=0;i<n;i++) t1s_x[i]=x[i];
  for (int i=0;i<n;i++) {
    dco::t1s::set(t1s_x[i],1.0,1);
    f(n,t1s_x,t1s_y);
    dco::t1s::set(t1s_x[i],0.0,1);
    dco::t1s::get(t1s_y,g[i],1);
  }
  dco::t1s::get(t1s_y,y);
  delete [] t1s_x;
}
```

Adjoint Mode with dco/c++


```
#include "dco.hpp"
void f(int n, dco::a1s::type *x, dco::a1s::type &y);

void a1s_driver(int ts, int n, double *x,
                double &y, double *g) {
  dco::a1s::global_tape=dco::a1s::tape::create(ts);
  dco::a1s::type *a1s_x=new dco::a1s::type[n], a1s_y;
  for (int i=0;i<n;i++) {    a1s_x[i]=x[i];
    dco::a1s::global_tape->register_variable(a1s_x[i]);
  }
  f(n,a1s_x,a1s_y);
  dco::a1s::get(a1s_y,y);
  dco::a1s::set(a1s_y,1.0,-1);
  dco::a1s::global_tape->interpret_adjoint();
  for (int i=0;i<n;i++) dco::a1s::get(a1s_x[i],g[i],-1);
  delete [] a1s_x;
  dco::a1s::tape::remove(dco::a1s::global_tape);
}
```

Second-Order Tangent-Linear Mode with dco/c++

```
...
void t2s_t1s_driver(int n, double *x,
                    double &y, double *g, double **H) {
  dco::t2s_t1s::type *t2s_t1s_x=new dco::t2s_t1s::type[n];
  dco::t2s_t1s::type t2s_t1s_y;
  for (int i=0;i<n;i++) {
    for (int j=0;j<=i;j++) {
      for (int k=0;k<n;k++) t2s_t1s_x[k]=x[k];
      dco::t2s_t1s::set(t2s_t1s_x[i],1.0,1,0);
      dco::t2s_t1s::set(t2s_t1s_x[j],1.0,0,2);
      f(n,t2s_t1s_x,t2s_t1s_y);
      dco::t2s_t1s::get(t2s_t1s_y,H[i][j],1,2);
      dco::t2s_t1s::get(t2s_t1s_y,g[i],1,0);
    }
  }
  dco::t2s_t1s::get(t2s_t1s_y,y);
  delete [] t2s_t1s_x;
}
```

```
void t2s_a1s_driver(int ts, int n, double *x,
    double &y, double *g, double **H) { ...
  for (int i=0;i<n;i++) { t2s_a1s_x[i]=x[i];
    dco::t2s_a1s::global_tape->register_variable(t2s_a1s_x[i]);
  }

  for (int i=0;i<n;i++) {
    if (i!=0) dco::t2s_a1s::global_tape->zero_adjoints();
    dco::t2s_a1s::set(t2s_a1s_x[i],1.0,0,2);
    f(n,t2s_a1s_x,t2s_a1s_y);
    dco::t2s_a1s::get(t2s_a1s_y,g[i],0,2);
    dco::t2s_a1s::get(t2s_a1s_y,y);
    dco::t2s_a1s::set(t2s_a1s_y,1.0,-1);
    dco::t2s_a1s::global_tape->interpret_adjoint();
    dco::t2s_a1s::set(t2s_a1s_x[i],0.0,0,2);
    for (int j=0;j<=i;j++)
      dco::t2s_a1s::get(t2s_a1s_x[j],H[j][i],-1,2);
  } ...
```