Differentiation of a finite element solver for stationary Navier-Stokes

René Schneider[1] Peter Jimack[2] Andrea Walther[3]

[1]Mathematik in Industrie und Technik
Fakultät für Mathematik
TU Chemnitz

[2]School of Computing
University of Leeds

[3]Institut für Mathematik
Universität Paderborn

4. June 2010

Department of Mathematics CHEMNITZ UNIVERSITY OF TECHNOLOGY

Outline

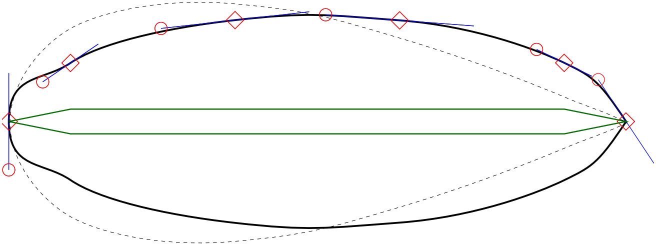
Motivation

- We are interested in optimisation problems where the performance function I is a functional of the solution of a PDE and the design variables influence the geometric shape of the PDE domain.
  $\Rightarrow$ **shape optimisation**
- Numerical methods for the PDE lead to very large systems of equations whose solution is expensive.
  $\Rightarrow$ Efficient methods are mandatory.
  $\Rightarrow$ Gradient based optimisation algorithms.
- Here special case of shape optimisation for stationary Navier-Stokes.

Motivation example 1: Shape optimisation ($Re = 10$)

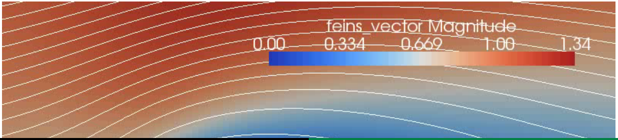
optimisation result

$$f_0 = 1.4056 \qquad f_* = 1.3714$$



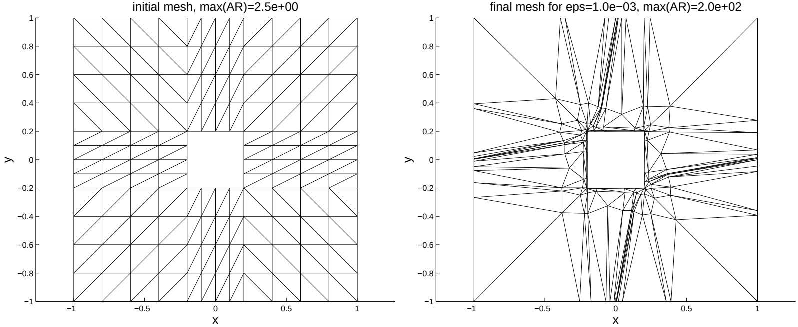
optimisation movie

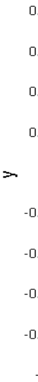
$$f_0 = 1.4056 \qquad f_* = 1.3714$$



Motivation example 2: mesh optimisation ($\varepsilon = 10^{-3}$)

initial and optimised coarse mesh





opt-adapt

Introduction to FEINS

- Name `FEINS`
  - originally "Finite Elements for Incompressible Navier-Stokes"
  - grown to general purpose FEM-library + collection of solvers
- primary interest: shape optimisation
- written entirely in C, currently $\approx 43,329$ lines of code
- free software, GPL-license
- only 2D, but 3D extension prepared
- triangular elements (P_1, P_2), extensions prepared
- modern (optimal) solvers, based on iterative solvers and multigrid preconditioning
- adaptive discretisation

FEINS: Equations

(only stationary problems)

- Poisson-equation:
  - as testbed for components for other problems
  - PCG solver with BPX or multigrid preconditioning
  - no shape gradient available
- Lamé-equations (linear elasticity):
  - PCG solver with BPX or multigrid preconditioning
  - shape gradient available
  - adaptivity
- incompressible Navier-Stokes equations (fluid dynamics):
  - linearisation with Newton's methods or Picard iteration
  - GMRES-solver with F_p preconditioner (Schur complement preconditioner)
  - Taylor-Hood elements ($\inf - \sup$ stable)
  - shape gradient available

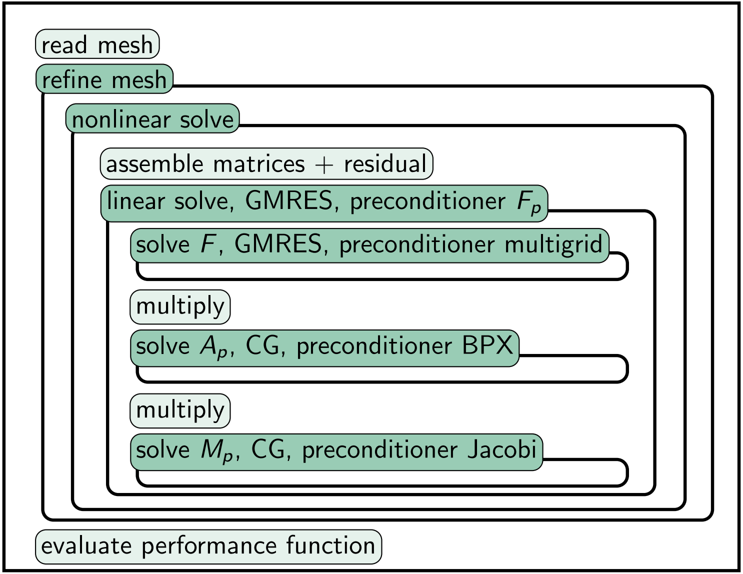
FEINS: Efficient solvers

problem	# unknowns	time solve (s)	time shape gradient (s)
Lamé	35,419,650	733	*
Navier-Stokes	37,769,219	31,000	37,000

System: on single core of

- 2x Intel Xeon Dual Core CPU 3.0 GHz
- 64 GB RAM PC2-5300 DDR2-667ECC
- openSUSE Linux 11.1 (x86_64)

FEINS: Navier-Stokes solver outer structure



Example problem

FE code `FEINS`:

- Domain where part of the boundary is defined by Bezier splines.
- Stationary incompressible Navier Stokes equations.
- Various performance functionals.
- Differentiation wrt. to parameters of the Bezier splines (control points).

Here tests for:

- Performance functionals:
    - Energy dissipation in whole domain.

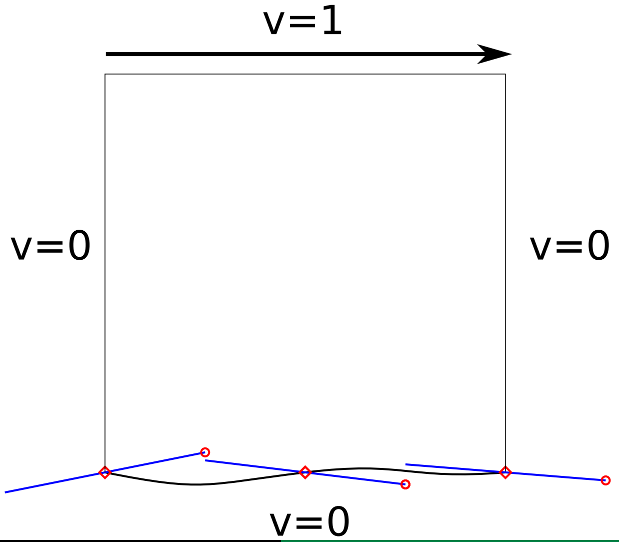
$$I_1 = \int_\Omega \frac{\mu}{2}(\mathrm{grad}\, u + \mathrm{grad}\, u^T) : (\mathrm{grad}\, u + \mathrm{grad}\, u^T)\, \mathrm{d}\Omega$$

    - Surface force in x direction, on bottom of cavity.

$$I_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}^T \int_{\Gamma_b} \Big[\mu(\mathrm{grad}\, u + \mathrm{grad}\, u^T) - \underbrace{\frac{2}{3}\mu \nabla \cdot u}_{=0} \Big] \cdot n\, \mathrm{d}\Gamma - \int_{\Gamma_b} p \cdot n\, \mathrm{d}\Gamma$$

Example problem

- Lid driven cavity flow, cavity with curved bottom, 12 Spline parameters



Adjoint Equations / Adjoint mode AD

Assumptions:

Number of independent and intermediate variables “large”, but only one dependent variable (result/output).

Sensitivity Analysis

- Consider scalar quantity

$$\begin{aligned} I(s) &= \widetilde{I}(u(s), s), \quad \text{where vector } u(s) \text{ defined by} \\ 0 &= R(u(s), s). \end{aligned} \tag{1}$$

$\Rightarrow$ require ∇I

- Consider small perturbation δs in s,

$$\delta I = \frac{\partial \widetilde{I}}{\partial u}\delta u + \frac{\partial \widetilde{I}}{\partial s}\delta s$$

$$0 = \delta R = \frac{\partial R}{\partial u}\delta u + \frac{\partial R}{\partial s}\delta s.$$

$\Rightarrow$ sensitivity equation

Adjoint Technique

$$
\begin{aligned}
\delta I &= \left(\frac{\partial\widetilde{I}}{\partial u}\delta u + \frac{\partial\widetilde{I}}{\partial s}\delta s\right) & & - \Psi^T\left(\frac{\partial R}{\partial u}\delta u + \frac{\partial R}{\partial s}\delta s\right)\\
&= \left(\frac{\partial\widetilde{I}}{\partial u} - \Psi^T\frac{\partial R}{\partial u}\right)\delta u & & + \left(\frac{\partial\widetilde{I}}{\partial s} - \Psi^T\frac{\partial R}{\partial s}\right)\delta s
\end{aligned}
$$

Thus

$$
\frac{DI}{Ds} = \frac{\partial\widetilde{I}}{\partial s} - \Psi^T\frac{\partial R}{\partial s}
$$

$$
\text{if}\quad \left[\frac{\partial R}{\partial u}\right]^T\Psi = \left[\frac{\partial\widetilde{I}}{\partial u}\right]^T.
$$

Example: Discrete Adjoint for FEM

- FEM $\Rightarrow$ linear system

$$K(s)u = b(s) \qquad\qquad R(u,s) := K(s)u - b(s)$$

- Functional $I := [g(s)]^T u$
- discrete-adjoint equation

$$K(s)^T \Psi = g(s)$$

evaluation of the gradient

$$\frac{DI}{Ds} = \frac{\partial I}{\partial s} - \Psi^T \frac{\partial R}{\partial s}$$

- Just **one** additional solve to get whole gradient, independent of $\dim(s)$.

Example for differentiating FE code

- Differentiate performance functional in FEM code wrt. domain geometry

solve

$$K(s)^T \Psi = g(s)$$

then evaluate gradient

$$\frac{DI}{Ds} = \frac{\partial I}{\partial s} - \Psi^T \frac{\partial R}{\partial s}$$

- difficulty:
  require $\partial R/\partial s$, $\partial I/\partial s$, i.e. derivatives of the FE discretisation wrt. node positions

Example for $\partial R/\partial s$ for FE code

- model problem:

$$F(s) := \int\limits_{T_\ell} \nabla\varphi_j(x) \cdot \nabla\varphi_i(x) \, \mathrm{d}\Omega \tag{2}$$

- unstructured mesh, isoparametric elements
- (2) is calculated by quadrature-formula on reference element

Example for $\partial R/\partial s$ for FE code (2)

$$F(s) = \sum_{k=1}^{m} \nabla_T \varphi_j(M(\hat{x}_k)) \cdot \nabla_T \varphi_i(M(\hat{x}_k)) |\det(J(\hat{x}_k))| w_k$$

$$\begin{aligned} x =& M(\hat{x}) = \sum_i s_i \hat{\varphi}_i(\hat{x}) \\ J :=& \left[\frac{\partial x}{\partial \hat{x}}\right] = \sum_i s_i \left[\hat{\nabla}\hat{\varphi}_i(\hat{x})\right]^T \\ \varphi(x) =& \hat{\varphi}(M^{-1}(x)) \\ \nabla_T \varphi_i(M(\hat{x}_k)) =& J^{-1}\hat{\nabla}\hat{\varphi}_i(\hat{x}_k) \end{aligned}$$

$\Rightarrow$ if derivatives of highlighted terms are calculated, only have to use product rule for rest

Example for $\partial R/\partial s$ for FE code (3)

Proposition 1

$$\frac{\partial[\nabla_T \varphi^{(i)}]_u}{\partial[s_k]_t} = -\left[\nabla_T \psi^{(k)}\right]_u \left[\nabla_T \varphi^{(i)}\right]_t$$

$$\frac{\partial|\det(J_T)|}{\partial s_{g_T(k)}} = |\det(J_T)| J_T^{-T} \hat{\nabla} \widehat{\psi}^{(k)}(\hat{x}_\ell).$$

Proof:
[S. PhD Thesis], [S./Jimack 2008]

- first part:
  implicit function theorem, re-organising terms
- second part:
  utilise adjoint representation of inverse of J_T

History and Background

- Wanted to apply discrete adjoint technique for shape optimisation in CFD
  s are node positions in FE mesh
  $\Rightarrow$ differentiate wrt. to these
- in 2003 started to implement FEM flow solver FEINS as testbed
- adjoint requires $\partial R/\partial u$, $\partial I/\partial u$, $\partial R/\partial s$, $\partial I/\partial s$
- $\partial R/\partial u$, $\partial I/\partial u$ simple
- $\partial R/\partial s$ and $\partial I/\partial s$ more tricky,
  $\Rightarrow$ tried to use AD (ADIC, ADOL-C)
- spent two weeks with little success
- used differentiation by hand
  [S. PhD Thesis], [S./Jimack 2008]
- Referee not happy about our opinion of AD.
  $\Rightarrow$ reconsidered

Applying ADOL-C to FEINS

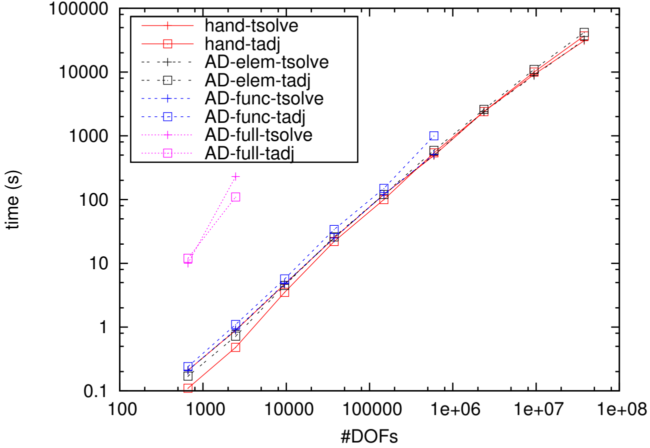
[Tijskens et. al. 2002]
Level at which applied

1. Full code.
2. Routines for $I(u, s)$ and $(\Psi^T R(u, s))$.
3. Element level of $I(u, s)$ and $R(u, s)$.

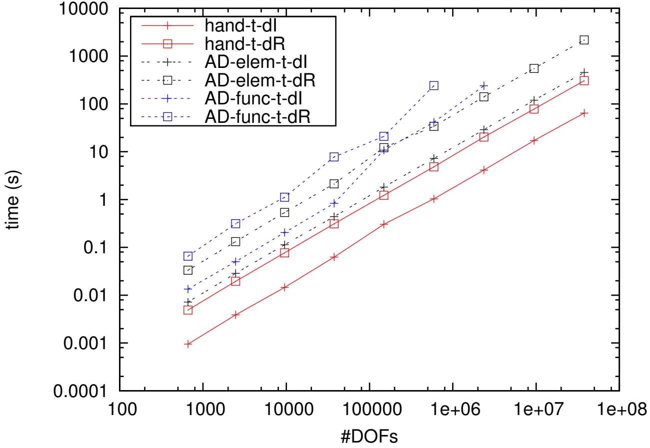
What we had to change

- Started out with hand differentiated code:
  - $\approx$ 3130 out of $\approx$ 40943 lines of code dedicated to $I(u, s)$ and the derivatives of $R(., .)$ and $I(., .)$.
- For ADOL-C on full code:
  - g++ instead of `gcc` compiler.
  - Dropped mesh generator `triangle`. (Definitely lost.)
  - Dropped all used `LAPACK` and `BLAS` routines.
    $\Rightarrow$ No direct coarse grid solvers for multigrid.
  - Changed $\approx$ 4240 out of $\approx$ 40943 lines of code.
- For ADOL-C on individual routines:
  - g++ instead of `gcc` compiler.
  - Dropped mesh generator `triangle`. (May be possible again.)
  - Changed $\approx$ 1859 out of $\approx$ 40943 lines of code.
- For ADOL-C on element level:
  - same restrictions as for "on individual routines"
  - Changed $\approx$ 2082 out of $\approx$ 40943 lines of code.

Results


hand-tsolve
hand-tadj
AD-elem-tsolve
AD-elem-tadj
AD-func-tsolve
AD-func-tadj
AD-full-tsolve
AD-full-tadj
time (s)
#DOFs
100000
10000
1000
100
10
1
0.1
100
1000
10000
100000
1e+06
1e+07
1e+08

Results



hand-t-dI
hand-t-dR
AD-elem-t-dI
AD-elem-t-dR
AD-func-t-dI
AD-func-t-dR
time (s)
#DOFs
10000
1000
100
10
1
0.1
0.01
0.001
0.0001
100
1000
10000
100000
1e+06
1e+07
1e+08

Results

	tape size in MByte			
# DOFs	AD-full	AD-function	AD-element	hand-coded
659	3,259	13	0	0
2,467	13,562	53	0	0
9,539	34,898	212	0	0
37,507		849	0	0
148,739		3,396	0	0
592,387		13,586	0	0
2,364,419		39,531	0	0
9,447,427		34,237	0	0
37,769,219		88,538	0	0

- biggest single file 64GByte
- bug or restriction in ADOL-C or in Linux?

Results

criteria (659 DOFs)

hand-coded/AD-function	AD-full
+9.4737171891857397e-01	+9.4737171819690480e-01
-3.2834420199797765e-02	-3.2834420248391054e-02

Results

grad (659 DOFs)

	relative error to hand-coded	
para	AD-function	AD-full
1	2.8e-12	2.9e+00
2	1.2e-12	1.3e+00
3	6.2e-12	6.2e-01
4	5.8e-13	2.3e+00
5	1.6e-12	2.8e+00
6	1.0e-12	1.2e+00
7	3.4e-12	1.0e-01
8	5.4e-13	2.4e-05
9	4.5e-12	1.6e-01
10	3.5e-12	6.0e-05
11	6.5e-13	1.0e-01
12	1.2e-12	2.4e-07

Results

criterion convergence (hand-coded), value

# DOFs	I_1	I_2
659	+9.47371719e-01	-3.28344202e-02
2,467	+1.23049527e+00	-3.34322002e-02
9,539	+1.52044541e+00	-3.37011495e-02
37,507	+1.81382551e+00	-3.38392446e-02
148,739	+1.66728516e+00	-3.37266467e-02
592,387	+1.66437167e+00	-3.37215419e-02
2,364,419	+1.66350100e+00	-3.37199409e-02
9,447,427	+1.66327072e+00	-3.37193837e-02
37,769,219	+1.66321284e+00	-3.37191661e-02

Results

criterion convergence (hand-coded), abs-error

# DOFs	l_1	l_2
659	-7.2e-01	+8.8e-04
2,467	-4.3e-01	+2.9e-04
9,539	-1.4e-01	+1.8e-05
37,507	+1.5e-01	-1.2e-04
148,739	+4.1e-03	-7.5e-06
592,387	+1.2e-03	-2.4e-06
2,364,419	+2.9e-04	-7.7e-07
9,447,427	+5.8e-05	-2.2e-07
37,769,219	+0.0e+00	+0.0e+00

Results

gradient convergence (hand-coded), abs-error

# DOFs	l_1	l_2
659	+2.8e-02	+2.2e-02
2,467	+1.9e-02	+1.2e-02
9,539	+1.4e-02	+4.0e-03
37,507	+1.0e-02	+1.3e-03
148,739	+6.4e-04	+3.1e-04
592,387	+2.0e-04	+4.9e-04
2,364,419	+5.7e-05	+5.8e-04
9,447,427	+1.2e-05	+1.3e-04
37,769,219	+0.0e+00	+0.0e+00

$\Rightarrow$ good news:
gradient converges with same order as $I(u)$.

Why does full AD not work?

- Andrea suggested there might be problems differentiating Krylov subspace solvers (GMRES, CG).
- Tested this with example problem $Ax = b$:

$$\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}$$

- system is solved with GMRES
- differentiate solution x_1 wrt. the nonzero entries of A

Differentiating GMRES

Results for $n = 20$

i	j	∇_{exact}	∇_{AD}	difference
1	1	-9.524e+00	-9.500e+00	2.4e-02
1	2	-1.810e+01	-1.800e+01	9.5e-02
2	2	-1.719e+01	-1.700e+01	1.9e-01
2	1	-9.048e+00	-9.000e+00	4.8e-02
9	10	-3.143e+01	-2.800e+01	3.4e+00
10	10	-2.881e+01	-2.750e+01	1.3e+00
10	9	-2.829e+01	-2.700e+01	1.3e+00
19	20	-9.524e-01	-1.000e+00	-4.8e-02
20	20	-4.762e-01	-5.000e-01	-2.4e-02

Conclusions/Outlook

- Algorithmic Differentiation with ADOL-C is valuable alternative to hand-coded derivatives, if applied at right level of code.
- “Automatic” is relative.
- Tape sizes appear to be one major problem with ADOL-C applied to this type of problem.
- AD not simple to apply to external libraries (`LAPACK`, `BLAS`, `triangle`).
- AD coded derivatives are significantly slower than (non-optimised) hand coded ones, but overhead small compared to the scale of the PDE code.
- It would be nice to compare with a source transformation tool.

Thank you!