Differentiation of a finite element solver for stationary Navier-Stokes

René Schneider\textsuperscript{1} \quad Peter Jimack\textsuperscript{2} \quad Andrea Walther\textsuperscript{3}

\textsuperscript{1}Mathematik in Industrie und Technik
Fakultät für Mathematik
TU Chemnitz

\textsuperscript{2}School of Computing
University of Leeds

\textsuperscript{3}Institut für Mathematik
Universität Paderborn

4. June 2010
1. Motivation
2. Structure of the FE solver
3. Differentiation/Adjoint by hand
4. AD at different abstraction levels
5. Conclusions
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.

\[ \Rightarrow \text{shape optimisation} \]

Numerical methods for the PDE lead to very large systems of equations whose solution is expensive.

\[ \Rightarrow \text{Efficient methods are mandatory.} \]

\[ \Rightarrow \text{Gradient based optimisation algorithms.} \]

Here special case of shape optimisation for stationary Navier-Stokes.
Motivation example 1: Shape optimisation \((Re = 10)\)

**Optimisation result**

\[
\begin{align*}
    f_0 &= 1.4056 \\
    f_* &= 1.3714
\end{align*}
\]

**Optimisation movie**

\[
\begin{align*}
    f_0 &= 1.4056 \\
    f_* &= 1.3714
\end{align*}
\]
Motivation example 2: mesh optimisation ($\varepsilon = 10^{-3}$)

initial and optimised coarse mesh
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

<table>
<thead>
<tr>
<th>problem</th>
<th># unknowns</th>
<th>time solve (s)</th>
<th>time shape gradient (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lamé</td>
<td>35,419,650</td>
<td>733</td>
<td>*</td>
</tr>
<tr>
<td>Navier-Stokes</td>
<td>37,769,219</td>
<td>31,000</td>
<td>37,000</td>
</tr>
</tbody>
</table>

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

- read mesh
- refine mesh
- nonlinear solve
  - assemble matrices + residual
  - linear solve, GMRES, preconditioner $F_p$
    - solve $F$, GMRES, preconditioner multigrid
      - multiply
        - solve $A_p$, CG, preconditioner BPX
          - multiply
            - solve $M_p$, CG, preconditioner Jacobi
  - evaluate performance function
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.
    \[ I_1 = \int_{\Omega} \frac{\mu}{2} (\text{grad} \ u + \text{grad} \ u^T) : (\text{grad} \ u + \text{grad} \ u^T) \ d\Omega \]
  - Surface force in x direction, on bottom of cavity.
    \[ I_2 = \left[ \begin{array}{c} 1 \\ 0 \end{array} \right]^T \left( \int_{\Gamma_b} \left[ \mu (\text{grad} \ u + \text{grad} \ u^T) - \frac{2}{3} \mu \nabla \cdot u \right] \cdot n \ d\Gamma - \int_{\Gamma_b} p \cdot n \ d\Gamma \right) \]
Example problem

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

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

\[ l(s) = \tilde{l}(u(s), s), \quad \text{where vector } u(s) \text{ defined by} \]
\[ 0 = R(u(s), s). \tag{1} \]

⇒ require \( \nabla l \)

Consider small perturbation \( \delta s \) in \( s \),

\[ \delta l = \frac{\partial \tilde{l}}{\partial u} \delta u + \frac{\partial \tilde{l}}{\partial s} \delta s \]

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

⇒ sensitivity equation

\[ \Rightarrow \text{sensitivity equation} \]
Adjoint Technique

\[
\delta l = \left( \frac{\partial l}{\partial u} \delta u + \frac{\partial l}{\partial s} \delta s \right) - \psi^T \left( \frac{\partial R}{\partial u} \delta u + \frac{\partial R}{\partial s} \delta s \right)
\]

\[
= \left( \frac{\partial l}{\partial u} - \psi^T \frac{\partial R}{\partial u} \right) \delta u + \left( \frac{\partial l}{\partial s} - \psi^T \frac{\partial R}{\partial s} \right) \delta s
\]

Thus

\[
\frac{DL}{Ds} = \frac{\partial l}{\partial s} - \psi^T \frac{\partial R}{\partial s}
\]

if

\[
\begin{bmatrix} \frac{\partial R}{\partial u} \end{bmatrix}^T \psi = \left[ \frac{\partial l}{\partial u} \right]^T
\]
Example: Discrete Adjoint for FEM

- FEM $\Rightarrow$ linear system

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

- Functional $l := [g(s)]^T u$

- discrete-adjoint equation

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

evaluation of the gradient

$$\frac{DI}{Ds} = \frac{\partial l}{\partial s} - \psi^T \frac{\partial R}{\partial s}$$

- Just one additional solve to get whole gradient, independent of $\text{dim}(s)$. 
Example for differentiating FE code

- Differentiate performance functional in FEM code wrt. domain geometry

\[ K(s)^T \Psi = g(s) \]

solve

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 \( \frac{\partial R}{\partial s} \) for FE code

- model problem:

\[
F(s) := \int_{T_\ell} \nabla \varphi_j(x) \cdot \nabla \varphi_i(x) \, d\Omega
\]

- unstructured mesh, isoparametric elements
- (2) is calculated by quadrature-formula on reference element
Example for $\partial R/\partial s$ for FE code (2)

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

$$x = M(\hat{x}) = \sum_i s_i \hat{\varphi}_i(\hat{x})$$

$$J := \left[ \frac{\partial x}{\partial \hat{x}} \right] = \sum_i s_i \left[ \hat{\nabla} \hat{\varphi}_i(\hat{x}) \right]^T$$

$$\varphi(x) = \hat{\varphi}(M^{-1}(x))$$

$$\nabla_T \varphi_i(M(\hat{x}_k)) = J^{-1} \hat{\nabla} \hat{\varphi}_i(\hat{x}_k)$$

⇒ 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**

\[
\frac{\partial [\nabla_T \varphi(i)]_u}{\partial [s_k]_t} = - \left[ \nabla_T \psi(k) \right]_u \left[ \nabla_T \varphi(i) \right]_t \\
\frac{\partial |\det(J_T)|}{\partial s_{gT}(k)} = |\det(J_T)| J_T^{-T} \hat{\nabla} \hat{\psi}(k)(\hat{x}_\ell).
\]

**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

- Full code.
- Routines for $I(u, s)$ and $(\Psi^T R(u, s))$.
- Element level of $I(u, s)$ and $R(u, s)$.
What we had to change

- Started out with hand differentiated code:
  - ≈ 3130 out of ≈ 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.
    - No direct coarse grid solvers for multigrid.
  - Changed ≈ 4240 out of ≈ 40943 lines of code.

- For ADOL-C on individual routines:
  - g++ instead of gcc compiler.
  - Dropped mesh generator triangle. (May be possible again.)
  - Changed ≈ 1859 out of ≈ 40943 lines of code.

- For ADOL-C on element level:
  - same restrictions as for “on individual routines”
  - Changed ≈ 2082 out of ≈ 40943 lines of code.
Results

![Graph showing the relationship between time (s) and #DOFs for different methods: hand-tsolve, hand-tadj, AD-elem-tsolve, AD-elem-tadj, AD-func-tsolve, AD-func-tadj, AD-full-tsolve, and AD-full-tadj. The graph demonstrates the scalability of these methods as the number of DOFs increases.](image-url)
Results

![Graph showing the comparison of time (s) vs. #DOFs for different differentiation methods.]

- Red line with crosses: hand-t-dI
- Red square: hand-t-dR
- Grey plus: AD-elem-t-dI
- Grey square: AD-elem-t-dR
- Blue square: AD-func-t-dI
- Blue plus: AD-func-t-dR

The graph indicates that the time required increases as the number of degrees of freedom (#DOFs) increases, with the automatic differentiation methods (AD) generally showing improved performance compared to the hand-differentiation (hand-t) methods.
## Results

### Tape Size in MByte

<table>
<thead>
<tr>
<th># DOFs</th>
<th>AD-full</th>
<th>AD-function</th>
<th>AD-element</th>
<th>hand-coded</th>
</tr>
</thead>
<tbody>
<tr>
<td>659</td>
<td>3,259</td>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2,467</td>
<td>13,562</td>
<td>53</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9,539</td>
<td>34,898</td>
<td>212</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>37,507</td>
<td>849</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>148,739</td>
<td>3,396</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>592,387</td>
<td>13,586</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2,364,419</td>
<td>39,531</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9,447,427</td>
<td>34,237</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>37,769,219</td>
<td>88,538</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>criteria (659 DOFs)</th>
<th>hand-coded/AD-function</th>
<th>AD-full</th>
</tr>
</thead>
<tbody>
<tr>
<td>+9.4737171891857397e-01</td>
<td>+9.4737171819690480e-01</td>
<td></td>
</tr>
<tr>
<td>-3.2834420199797765e-02</td>
<td>-3.2834420248391054e-02</td>
<td></td>
</tr>
</tbody>
</table>
## Results

### grad (659 DOFs)

<table>
<thead>
<tr>
<th>para</th>
<th>relative error to hand-coded</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AD-function</td>
</tr>
<tr>
<td>1</td>
<td>2.8e-12</td>
</tr>
<tr>
<td>2</td>
<td>1.2e-12</td>
</tr>
<tr>
<td>3</td>
<td>6.2e-12</td>
</tr>
<tr>
<td>4</td>
<td>5.8e-13</td>
</tr>
<tr>
<td>5</td>
<td>1.6e-12</td>
</tr>
<tr>
<td>6</td>
<td>1.0e-12</td>
</tr>
<tr>
<td>7</td>
<td>3.4e-12</td>
</tr>
<tr>
<td>8</td>
<td>5.4e-13</td>
</tr>
<tr>
<td>9</td>
<td>4.5e-12</td>
</tr>
<tr>
<td>10</td>
<td>3.5e-12</td>
</tr>
<tr>
<td>11</td>
<td>6.5e-13</td>
</tr>
<tr>
<td>12</td>
<td>1.2e-12</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>DOFs</th>
<th>$l_1$</th>
<th>$l_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>659</td>
<td>+9.47371719e-01</td>
<td>-3.28344202e-02</td>
</tr>
<tr>
<td>2,467</td>
<td>+1.23049527e+00</td>
<td>-3.34322002e-02</td>
</tr>
<tr>
<td>9,539</td>
<td>+1.52044541e+00</td>
<td>-3.37011495e-02</td>
</tr>
<tr>
<td>37,507</td>
<td>+1.81382551e+00</td>
<td>-3.38392446e-02</td>
</tr>
<tr>
<td>148,739</td>
<td>+1.66728516e+00</td>
<td>-3.37266467e-02</td>
</tr>
<tr>
<td>592,387</td>
<td>+1.66437167e+00</td>
<td>-3.37215419e-02</td>
</tr>
<tr>
<td>2,364,419</td>
<td>+1.66350100e+00</td>
<td>-3.37199409e-02</td>
</tr>
<tr>
<td>9,447,427</td>
<td>+1.66327072e+00</td>
<td>-3.37193837e-02</td>
</tr>
<tr>
<td>37,769,219</td>
<td>+1.66321284e+00</td>
<td>-3.37191661e-02</td>
</tr>
</tbody>
</table>
### Results

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

**gradient convergence (hand-coded), abs-error**

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

⇒ good news:

gradient converges with same order as $l(u)$. 

Differentiation of a finite element solver for stationary Navier-Stokes
Andrea suggested there might be problems differentiating Krylov subspace solvers (GMRES, CG).

Tested this with example problem $Ax = b$:

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

System is solved with GMRES

differentiate solution $x_1$ wrt. the nonzero entries of $A$
Differentiating GMRES

Results for $n = 20$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
<th>$\nabla_{\text{exact}}$</th>
<th>$\nabla_{\text{AD}}$</th>
<th>difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-9.524e+00</td>
<td>-9.500e+00</td>
<td>2.4e-02</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-1.810e+01</td>
<td>-1.800e+01</td>
<td>9.5e-02</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-1.719e+01</td>
<td>-1.700e+01</td>
<td>1.9e-01</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-9.048e+00</td>
<td>-9.000e+00</td>
<td>4.8e-02</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>-3.143e+01</td>
<td>-2.800e+01</td>
<td>3.4e+00</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>-2.881e+01</td>
<td>-2.750e+01</td>
<td>1.3e+00</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>-2.829e+01</td>
<td>-2.700e+01</td>
<td>1.3e+00</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>-9.524e-01</td>
<td>-1.000e+00</td>
<td>-4.8e-02</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>-4.762e-01</td>
<td>-5.000e-01</td>
<td>-2.4e-02</td>
</tr>
</tbody>
</table>
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!