Optimisation of triple-ring-electrodes on piezoceramic transducers using algorithmic differentiation

Benjamin Jurgelucks ¹  Leander Claes ²

¹Research group Mathematics and its Applications, Paderborn University
²Measurement Engineering Group, Paderborn University

7th International Conference on Algorithmic Differentiation
Oxford, United Kingdom
15.09.2016
Outline

1 Motivation
2 Physical model & Sensitivity Evaluation
3 Applying ADOL-C to CFS++
4 Finite Differences & Algorithmic Differentiation for Sensitivity Evaluation
5 Optimisation
6 Conclusion
Motivation

- Piezoelectric transducers are used as hardware components (i.e. accelerometer in smartphones), many are ceramic-based.
- For simulation and development of piezoceramics precise knowledge about material parameters is indispensable.
- Correct material parameter data of piezoelectric ceramics can differ up to 20% from data given by manufacturer.
- Existing parameter estimation methods may be inaccurate or expensive.
Motivation

• Piezoelectric transducers are used as hardware components (i.e. accelerometer in smartphones), many are ceramic-based
• For simulation and development of piezoceramics precise knowledge about material parameters is indispensable
• Correct material parameter data of piezoelectric ceramics can differ up to 20% from data given by manufacturer
• Existing parameter estimation methods may be inaccurate or expensive
• Idea: formulate parameter estimation problem as inverse problem with impedance as measurable quantity and solve (Lahmer, 2008)
• Problem: for some parameters sensitivity of impedance with respect to material parameters is low → Cannot solve inverse problem → Increase sensitivity
• Usually, electrode plates completely covering top and bottom of piezoceramic are used to excite the piezoceramic
• Studies have shown that sensitivity with respect to material parameters can be increased by attaching triple-ring electrodes
• Sensitivity of impedance with respect to parameters of ceramic depends on (ring) geometry
• Many configurations show low to zero sensitivity
• Identify configurations with high sensitivity
  → optimisation problem
• Finite differences → a lot of locally optimal points
  → investigate with AD
Motivation

Triple-ring electrodes in real
CFS++

Coupled Field Simulation in C++

- Mainly developed at
  - Vienna University of Technology
  - Friedrich-Alexander-Universität Erlangen-Nürnberg
  - Alpen-Adria-Universität Klagenfurt
- About 30 developers (many Ph.D. students)
- Approx. 750 source files (.cc .hh), 310k lines (230k .cc & 80k .hh)
- “Code-accessible” for academic purposes but not quite open source
- Heavy use of templating, macros and generally use of object oriented programming style
ADCFS

Three implementations of ADOL-C into CFS++

- Prototype for heat equation based on “type insertion”
- Implementation for branch “08/10 Optimization Master” → Full electrode inverse problems by Tom Lahmer with “typedef” approach
- Implementation for current CFS++ version (“typedef” approach), handles triple-ring electrodes

Main challenges:

- Combination of templates, template metaprogramming and inheritance → complexity of source changes needed → see 14th EuroAD workshop talk (also in Oxford)
- String arithmetic, OO programming and MuParser (aka MathParser) → (indirectly) limited to use of ADOL-C’s traceless (forward) mode
- Linear algebra externalisation with ADOL-C traceless mode
Strings and MuParser

- Legitimate idea: Let the user prescribe more general relations by providing formulae as input via string.
- Example:
  
  ```xml
  <charge name="electrode1" value="1.0*spike(2e-7,t-1e-7)"/>
  ```

- Unmodified ADOL-C leads to the error message:
  
  MathParser reports:
  
  -------------------
  Message: Misplaced colon at position 10
  Formula:
  
  `((((Value:5.31482e-09ADValues(1):0(a))*(Value:6.12323e-17ADValues(1):0(a))))+[...])`

  ⇒ MuParser is trying to evaluate strings of adoubles 
  ⇒ MuParser is using “string arithmetic” (concatenation of strings)

  Derivative calculation is being bypassed!
String arithmetic and OO programming

Main timestepping loop:

```cpp
for (actTimeStep_ = startStep_; actTimeStep_ <= endStep_; actTimeStep_ += 1, count++)
{
    mathParser_→SetValue( Parser::GLOB_HANDLER, "t", actTime_ );
    mathParser_→SetValue( Parser::GLOB_HANDLER, "dt", dt );
    mathParser_→SetValue( Parser::GLOB_HANDLER, "step", actTimeStep_ );
    // [...]
    ptPDE_→GetSolveStep()→PreStepTrans();
    ptPDE_→GetSolveStep()→SolveStepTrans();
    ptPDE_→GetSolveStep()→PostStepTrans();
    // [...]
}
```

- String arithmetic interferes again (possible to deal with)
- Main methods (and submethods) are pure virtual and take no arguments
  → How to reuse trace here?
- Tracing whole procedure is too expensive
  → Cancel traced mode run after ≈10 hours and 400 Gb trace after only a few timesteps finished
Solutions:

- Use ADOL-C’s traceless mode instead of traced mode
  - Only use externalisation where it is advantageous (deep memory copy in traceless mode)
  - Parallelisation inside CFS++?

- Dealing with strings:
  - Heavy string arithmetic usage in at least one folder full of core classes + MuParser calls
    → Rewriting is unlikely (newly introduced feature)
  - Formulae are not long. Use analytical derivatives?

- Overall solution: Change code to make exactly my case work
Run times

<table>
<thead>
<tr>
<th># timesteps</th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>primal</td>
<td>0.71</td>
<td>0.71</td>
<td>0.741</td>
<td>0.763</td>
<td>0.834</td>
<td>1.09</td>
<td>1.27</td>
<td>1.933</td>
<td>3.427</td>
<td>6.363</td>
</tr>
<tr>
<td>FD (= 2 × primal)</td>
<td>1.42</td>
<td>1.42</td>
<td>1.482</td>
<td>1.526</td>
<td>1.668</td>
<td>2.18</td>
<td>2.54</td>
<td>3.866</td>
<td>6.854</td>
<td>12.726</td>
</tr>
<tr>
<td>AD w/o ext.</td>
<td>1.50</td>
<td>1.56</td>
<td>1.71</td>
<td>1.95</td>
<td>2.53</td>
<td>4.22</td>
<td>6.90</td>
<td>12.38</td>
<td>28.78</td>
<td>56.22</td>
</tr>
<tr>
<td>AD with ext.</td>
<td>1.538</td>
<td>1.565</td>
<td>1.644</td>
<td>1.801</td>
<td>2.22</td>
<td>3.441</td>
<td>5.422</td>
<td>9.413</td>
<td>21.53</td>
<td>41.818</td>
</tr>
<tr>
<td>Ratio AD/FD</td>
<td>1.08</td>
<td>1.10</td>
<td>1.11</td>
<td>1.18</td>
<td>1.33</td>
<td>1.58</td>
<td>2.13</td>
<td>2.43</td>
<td>3.14</td>
<td>3.29</td>
</tr>
</tbody>
</table>

**Table**: Runtime (in sec) of single FEM simulation for CFS configurations vs. number of timesteps.

<table>
<thead>
<tr>
<th>Mesh element size</th>
<th>0.00006</th>
<th>0.00005</th>
<th>0.00004</th>
<th>0.00003</th>
<th>0.00002</th>
<th>0.00001</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD (6 processors)</td>
<td>8.857</td>
<td>14.06</td>
<td>22.6</td>
<td>40.38</td>
<td>110.23</td>
<td>606.757</td>
</tr>
<tr>
<td>AD (3 processors)</td>
<td>45.72</td>
<td>67.59</td>
<td>108.96</td>
<td>208.02</td>
<td>510.91</td>
<td>3457.95</td>
</tr>
<tr>
<td>Ratio (3-AD)/(6-FD)</td>
<td>2.58</td>
<td>2.40</td>
<td>2.41</td>
<td>2.58</td>
<td>2.32</td>
<td>2.85</td>
</tr>
</tbody>
</table>

**Table**: Runtime (in sec) for full sensitivity evaluation vs. mesh size.
Finite Differences

- Forward differences:
  \[ f'(x) \approx \frac{f(x + h) - f(x)}{h} \text{ for small } h \]

- Theory of finite differences states (i.e. (Griewank, Walther 2008))
  \( \exists h_{opt} \) optimal stepwidth depending on given \( x \).

- \( h_{opt} \) is known neither a-priori nor a-posteriori (without reference value given analytically or by AD)

- AD usually precise up to machine accuracy (Griewank, Walther 2008)
\[ \gamma = \frac{h}{x} \quad \text{("normed stepwidth")} \]
FD vs. AD

$\gamma = h/x$ "normed stepwidth"

Relative difference over stepwidth

- Default config
- r16
- r33
- r4

B. Jurgelucks & L. Claes
• AD and FD match very well for a “good” stepwidth

• Even in non-extremal cases: accuracy of FD is unclear and can vary for different electrode configurations

• Goal: Eliminate FD from the simulation completely by substituting AD for FD
Optimisation

FD optimisation using commercial tool (more details in Kulshreshtha et. al, 2015):
Use derivative-free optimisation method LINCOA (Powell 2014)
  - Extension to Powell’s trust-region optimiser BOBYQA
  - Incorporates linear constraints

Final optimisation problem:
\[
\begin{align*}
\min_{\vec{r} \in \mathbb{R}^4} & \quad J(\vec{r}) \\
\text{s.t.} & \quad A\vec{r} \leq b
\end{align*}
\]

with
\[
J(\vec{r}) := \frac{1}{\|WS\nabla z(\vec{r})\|_2^2} \quad \text{[now } J(\vec{r}) := -\|WS\nabla z(\vec{r})\|_2 \text{]}
\]

with diagonal matrices $S, W \in \mathbb{R}^{10 \times 10}$ for scaling and adding weight to parameters.
Initial configurations

Use barely feasible configurations as initial points:
## Initial configurations

<table>
<thead>
<tr>
<th>param.</th>
<th>gain ratio case1</th>
<th>gain ratio case2</th>
<th>gain ratio case3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_{11}$</td>
<td>1.2434</td>
<td>0.9293</td>
<td>1.1830</td>
</tr>
<tr>
<td>$c_{33}$</td>
<td>1.4205</td>
<td>0.8531</td>
<td>2.3508</td>
</tr>
<tr>
<td>$c_{44}$</td>
<td>5.3202</td>
<td>1.1749</td>
<td>4.3539</td>
</tr>
<tr>
<td>$c_{12}$</td>
<td>1.0125</td>
<td>0.5684</td>
<td>2.6384</td>
</tr>
<tr>
<td>$c_{13}$</td>
<td>1.0978</td>
<td>0.8933</td>
<td>1.3414</td>
</tr>
<tr>
<td>$\epsilon_{33}$</td>
<td>1.3629</td>
<td>0.6659</td>
<td>2.5290</td>
</tr>
<tr>
<td>$e_{31}$</td>
<td>1.1556</td>
<td>1.0677</td>
<td>1.2714</td>
</tr>
<tr>
<td>$e_{33}$</td>
<td>1.8522</td>
<td>0.9212</td>
<td>2.4383</td>
</tr>
<tr>
<td>$\epsilon_{11}$</td>
<td>2.8385</td>
<td>1.2864</td>
<td>2.0128</td>
</tr>
<tr>
<td>$e_{15}$</td>
<td>3.2481</td>
<td>1.4586</td>
<td>1.8070</td>
</tr>
</tbody>
</table>

- Initial and optimised sensitivities are still relatively small
- every optimisation converges to a different end configuration
  → many locally optimal points
Optimisation using CFS++ with AD:
Points evaluated

Consider optimisation \texttt{fmincon} with only one material parameter:

⇒ Different starting points (blue) now run into the same end points (red)
Optimisation using CFS++ with AD: Function iterations

Note: Optimisation configured to stop at $\Delta \vec{r} \leq 10^{-5}$
Optimisation using CFS++ with FD: Points evaluated

⇒ Different starting points (blue) do not run into the same end points (red)
Optimisation using CFS++ with FD: Function iterations

Note: Optimisation configured to stop at $\Delta \vec{r} \leq 10^{-5}$
Conclusions:

- Optimisation with FD feasible, many locally optimal points
- Cost function using AD is easier to optimise
- Finally possible optimisation of triple-ring case with AD

Challenges:

- Computationally expensive simulation
- Find globally optimal point if possible
- Further fine tuning of weights is necessary → Pareto optimality?

Ongoing work:

- Optimisation for all material parameters
- Eliminate FD from simulation and optimisation
- ADOL-C’s traceless higher order vector mode make derivative-based optimisation possible. Final implementation ongoing


Thank you for your attention.