Computing Second Order Derivatives with ADiMat
Facilitating Optimal Experimental Design by Automatic Differentiation

Johannes Willkomm
Institute for Scientific Computing
Technische Universität Darmstadt

May 14, 2013 / Colloquium of the Interdisciplinary Center for Scientific Computing (IWR) of Heidelberg University
Introduction

Second Order Derivatives

ADiMat

Second Order Derivatives with ADiMat

Full Second Order Derivatives: Hessians

Nested Application of ADiMat

Performance Results
Second Order Derivatives

Second order derivatives are often required in software for Optimal Experimental Design (OED), for example in VPLAN [Körkel, 2002].

We consider functions of the form
\[ z = F(x, p, q) : (\mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_q}) \to \mathbb{R}^m \]

Costs: time \( T_F \) and memory \( M_F \)

Needed derivatives: \( \frac{d^2F}{dx^2}, \frac{d^2F}{dxdq}, \text{ and } \frac{d^2F}{dpdq} \)

Abbreviations: \( X = [x, p, q], n = n_x + n_p + n_q \)

Using Automatic Differentiation (AD) for 2\(^{nd}\) order derivatives is attractive for performance reasons

- Precise derivatives help in optimization
- AD is often more efficient than numerical methods
- AD is more broadly applicable (in the mathematical sense)
Example Function

```matlab
function z = F(x, p, q)
    t1 = 1;
    for i = 1:length(x)
        t1 = t1 .* sin(x(i));
    end
    t2 = 1;
    for i = 1:length(p)
        t2 = t2 .* sin(x(i) .* q(i));
    end
    t3 = 1;
    for i = 1:length(q)
        t3 = t3 .* cos(p(i) .* q(i));
    end
    z = [t1, t2, t3];
```

Second Order Derivatives

2\textsuperscript{nd} Order Derivatives of Example Function

Figure: \texttt{spy} plots of the Hessians of the $m = 3$ output components of $F$ for $n_x = n_p = n_q = 10$. 
Automatic Differentiation for Matlab (ADiMat) is an AD tool for MATLAB (http://www.adimat.de)

Uses source transformation, but combines it with operator overloading [Bischof & Bücker et al., 2002]

Supports both forward mode (FM) and reverse mode (RM)

Capitalizes on the high-level mathematical functions and operators in MATLAB, like *, \, eig, svd, expm, cross, interp1, roots, ...

ADiMat features
- Comfortable user interface [Willkomm & Bischof & Bücker, 2012]
- Higher order derivatives (univariate and mixed)
ADiMat Internals

- ADiMat transforms function $F$ to a new function

\[
\text{function } z = F(a, b) \\
z = a \ast b;
\]

- Evaluation of derivatives of $F$ at certain arguments $a, b$ by running the generated functions

- Derivative inputs have to be properly initialized (seeding)
Scalar and Vector Mode

- Derivative variables have the **same shape** as the originals
  
  \[ g_{a} \times b \]

  This only allows for a single directional derivative in \( g_{x} \): **scalar mode**

- For **vector mode** use **derivative class** objects, with **overloaded operators**, as containers for \( n_{dd} > 1 \) directional derivatives

  \[ g_{a} \times b \]

  - Overloaded operator dispatch happens at run time, since MATLAB is **weakly typed**
  - Performance is quite **bad**
Alternative: Vectorized Code

- Alternative: “vectorize” the code explicitly
  - Replace objects by opaque data type
  - Replace overloaded operators by function calls

```matlab
function z = F(a, b)
    z = a * b;
end
```

```matlab
function [d_z z] = d_F(d_a, a, d_b, b)
    d_z = opdiff_mult(d_a, a, d_b, b);
    z = a * b;
end
```

- Resolution of function calls now at compile time
  - Often very good performance, especially with “scalar” or “F77-style” codes, for small to medium $n_{dd}$. 
Main driver for second order derivatives is `admHessian`
Computes the full Hessian matrix $H$
or (multiple) products $H \cdot v$ thereof
- We can pick out our desired derivatives from $H$,
- or compute only the suitable linear combinations $H \cdot v$
Returns the Hessians of all function results $H_k$, $1 \leq k \leq m$
Two evaluation strategies:
- Forward over reverse mode (default)
- Linear combination of second order univariate Taylor coefficients
Forward over Reverse Mode

- Differentiate function $F$ in RM
- Run RM evaluation with a typical first order FM OO class
  - Obtain first order derivatives of function result by the FM
  - and the derivatives of those w.r.t. all inputs by the RM
- Costs:
  - Time $O(m) \cdot T_F$ for one $H \cdot v$ product
  - Time $O(n \cdot m) \cdot T_F$ for full $H$
  - Space $O(T_F)$ for the stack required by the RM

```matlab
adopts = admOptions('i', [1 2 3]);
adopts.functionResults = {z};
H = admHessian(@F, 1, x, p, q, adopts);
```

- Caveat: FM OO class supports very few builtins as of yet
Driver for Hessians

Forward over Reverse Mode

- We don’t need the full Hessian $H$, in particular not $\frac{d^2 F}{dq^2}$
- Mask out the columns corresp. to $q$ with a seed matrix $S$

$$S = \begin{pmatrix} I_{n_x} & 0_{np} \\ 0_{np} & I_{np} \\ 0_{nq} & 0_{nq} \end{pmatrix} \in \mathbb{R}^{n \times (n_x + n_p)}$$

- With the example function $F$ we could even use compression (adding together the $x$– and $p$–columns)

- Costs:
  - Time $O((n_x + n_p) \cdot m) \cdot T_F$ for the desired sub blocks of $H$

```matlab
S = [eye(numel(x)), zeros(numel(x)), zeros(numel(p)), eye(numel(p)), zeros(numel(q)), zeros(numel(q))];

H = admHessian(@F, S, x, p, q, adopts);
```
2nd Order Taylor Coefficients

- Propagate 2nd order univariate Taylor coefficients in FM
- Compute the off-diagonal Hessian entries as
  \[ H_{i,j} = \frac{1}{2} \left(D^2_{e_i+e_j}F(X) - D^2_{e_i}F(X) - D^2_{e_j}F(X)\right), \quad i \neq j \]
  [Griewank & Walther, 2008]
  - For full H need \( n + \frac{n(n+1)}{2} \) derivative directions
- Costs:
  - Time \( O(n^2) \cdot T_F \) for full H
  - Space \( O(n^2) \cdot M_F \)

```matlab
adopts.hessianStrategy = 't2for';
% Alternatives: use FD, vectorized Taylor mode
% adopts.admDiffFunction = @admDiffFD;
% adopts.admDiffFunction = @admTaylorVFor;
H = admHessian(@F, 1, x, p, q, adopts);
```
Nested Application of ADiMat

Mixed 2\textsuperscript{nd} Order Directional Derivatives

- Generate three functions by twice applying the FM
  - Diff. $F$ in FM w.r.t. $x$, then $dx_F$ w.r.t. both $x$ and $g_x$
  - Also differentiate $dx_F$ w.r.t. $q$
  - Differentiate $F$ in FM w.r.t. $p$, then $dp_F$ w.r.t. $q$

\texttt{alias } \texttt{ac='adimat-client-F'}
\texttt{ac -ix -d1 -odx_F.m F.m}
\texttt{ac -ig_x , x -d1 -sgradprefix=h_ -odx_dx_F.m dx_F.m}
\texttt{ac -iq -d1 -sgradprefix=h_ -odq_dx_F.m dx_F.m}
\texttt{ac -ip -d1 -odp_F.m F.m}
\texttt{ac -iq -d1 -sgradprefix=h_ -odq_dp_F.m dp_F.m}

- Costs:
  - Time $O(1) \cdot T_F$ and space $O(1) \cdot M_F$ for one entry $H_{i,j}$
  - Time $O(n_x^2/2 + n_x n_q + n_p n_q) \cdot T_F$ for the desired sub blocks
  - Caveat: ADiMat may not be able to reprocess its code
Mixed 2\textsuperscript{nd} Order Directional Derivatives

\[
\begin{align*}
\text{h}_g\text{x} &= \text{zeros(size}(x)) ; \quad \text{h}_x = \text{h}_g\text{x} ; \quad \text{g}_x = \text{h}_x ; \\
\text{for } i &= 1:\text{numel}(x) , \quad \text{for } j = 1:i \\
& \quad \text{h}_x(i) = 1 ; \quad \text{g}_x(j) = 1 ; \\
& \quad \text{h}_g\text{f} = \text{dx}dx_F(\text{h}_g\text{x} , \text{g}_x , \text{h}_x , x , p , q) ; \\
& \quad \text{dF}_dx\text{dx}(:,i,j) = \text{h}_g\text{f}(:) ; \\
& \quad \text{dF}_dx\text{dx}(:,j,i) = \text{h}_g\text{f}(:) ; \\
& \quad \text{h}_x(i) = 0 ; \quad \text{g}_x(j) = 0 ; \\
\text{end} \quad \text{end} \\
\text{h}_q &= \text{zeros(size}(q)) ; \quad \text{g}_x = \text{zeros(size}(x)) ; \\
\text{for } i &= 1:\text{numel}(x) , \quad \text{for } j = 1:\text{numel}(q) \\
& \quad \text{g}_x(i) = 1 ; \quad \text{h}_q(j) = 1 ; \\
& \quad \text{h}_g\text{f} = \text{dq}dx_F(\text{g}_x , x , p , \text{h}_q , q) ; \\
& \quad \text{dF}_dq\text{dx}(:,i,j) = \text{h}_g\text{f}(:) ; \\
& \quad \text{g}_x(i) = 0 ; \quad \text{h}_q(j) = 0 ; \\
\text{end} \quad \text{end} \ % \text{otherwise for } dF_dqdp
\end{align*}
\]
Nested Application of ADiMat

Complex Variable Method over FM

- First order FM to compute Jacobian
- Apply complex variable (CV) method on top of that
  - Only applicable if $F$ is real analytic
  - Very precise and efficient approximation to derivatives

```matlab
adopts2 = admOptions('i', [1 2 3] + 2, 'd', 1);
adopts2.nargout = 1;
H = admDiffComplex(@admDiffVFor, S, ...
  @F, 1, x, p, q, adopts, adopts2);
H = reshape(H, [numel(z) size(S)]);
```

- Costs:
  - Time $O(n) \cdot T_F$ for one $H \cdot v$ product
  - Time $O(n \cdot (n_x + n_p)) \cdot T_F$ for the desired sub blocks of $H$
  - Space $O(n) \cdot M_F$
Performance Test

- Six methods to compute the three Hessian sub blocks:
Summary

- Presented six different methods for evaluation of 2\textsuperscript{nd} order derivatives with ADiMat
  - There are more
  - Certain room to manoeuver w.r.t. performance and language support

- ToDo items
  - Broaden language support of the 2\textsuperscript{nd} higher derivative methods in ADiMat
  - And also enhance performance of them

- Outreach
  - Visit ADiMat on the web at www.adimat.de
  - Subscribe to the ADiMat Users mailing list
References I

Stefan Körkel
Das Softwarepaket VPLAN
Dissertation, 2002

Andreas Griewank & Andrea Walther
Evaluating Derivatives
SIAM, 2008

C. Bischof, M. Bücker, B. Lang, A. Rasch & A. Vehreschild
Combining Source Transformation and Operator Overloading Techniques to Compute Derivatives for MATLAB Programs
J. Willkomm, C. Bischof & M. Bücker
A New User Interface for ADiMat: Toward Accurate and Efficient Derivatives of Matlab Programs with Ease of Use
Int. J. Computational Science and Engineering, *to appear*