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

TECHNISCHE
UNIVERSITÄT
DARMSTADT

SCIENTIFIC
COMPUTING

Outline

1. Introduction
   - Second Order Derivatives
   - ADiMat
2. Second Order Derivatives with ADiMat
   - Full Second Order Derivatives: Hessians
   - Nested Application of ADiMat
3. Performance Results

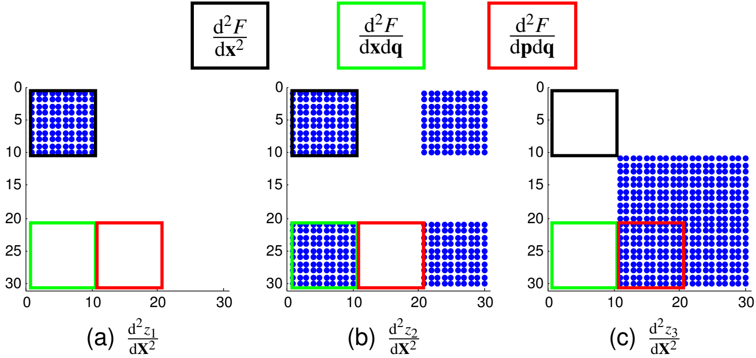
Second Order Derivatives

- 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 $\mathbf{z} = F(\mathbf{x}, \mathbf{p}, \mathbf{q}) : (\mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_q}) \to \mathbb{R}^m$
    - Costs: time T_F and memory M_F
    - Needed derivatives: $\frac{\mathrm{d}^2 F}{\mathrm{d}\mathbf{x}^2}$, $\frac{\mathrm{d}^2 F}{\mathrm{d}\mathbf{x}\mathrm{d}\mathbf{q}}$, and $\frac{\mathrm{d}^2 F}{\mathrm{d}\mathbf{p}\mathrm{d}\mathbf{q}}$
    - Abbreviations: $\mathbf{X} = [\mathbf{x}, \mathbf{p}, \mathbf{q}]$, $n = n_x + n_p + n_q$
- Using Automatic Differentiation (AD) for 2$^{\text{nd}}$ 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

```
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];
```

2^{nd} Order Derivatives of Example Function



(a) $\frac{d^2z_1}{d\mathbf{X}^2}$ (b) $\frac{d^2z_2}{d\mathbf{X}^2}$ (c) $\frac{d^2z_3}{d\mathbf{X}^2}$

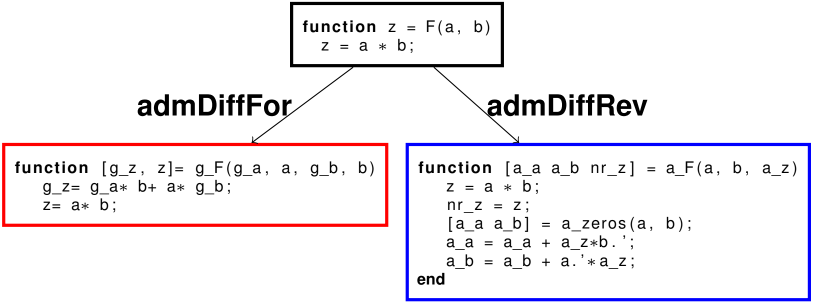
Figure: `spy` plots of the Hessians of the $m = 3$ output components of F for $n_x = n_p = n_q = 10$.

ADiMat

- **A**utomatic **Di**fferentiation for **Mat**lab (**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



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

admDiffFor

admDiffRev

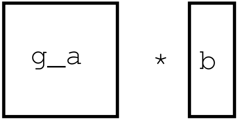
```
function [g_z, z]= g_F(g_a, a, g_b, b)
   g_z= g_a* b+ a* g_b;
   z= a* b;
```

```
function [a_a a_b nr_z] = a_F(a, b, a_z)
   z = a * b;
   nr_z = z;
   [a_a a_b] = a_zeros(a, b);
   a_a = a_a + a_z*b.';
   a_b = a_b + a.'*a_z;
end
```

- Evaluation of derivatives of `F` at certain arguments `a, b` by running the generated functions
- Derivative inputs have to be properly initialized (seeding)

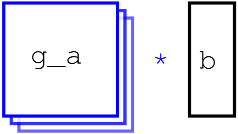
Scalar and Vector Mode

- Derivative variables have the same shape as the originals



This only allows for a single directional derivative in `g_x`: scalar mode

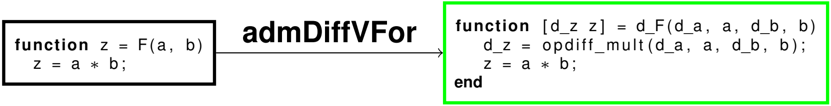
- For vector mode use derivative class objects, with overloaded operators, as containers for $n_{\mathrm{dd}} > 1$ directional derivatives



- 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



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

admDiffVFor

```
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}.

Full 2$^{\text{nd}}$ Order Derivatives: Hessians

- Main driver for second order derivatives is **admHessian**
- Computes the full Hessian matrix H
- or (multiple) products $\mathrm{H} \cdot \mathbf{v}$ thereof
  - We can pick out our desired derivatives from H,
  - or compute only the suitable linear combinations $\mathrm{H} \cdot \mathbf{v}$
- 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 $\mathrm{H} \cdot \mathbf{v}$ product
  - Time $O(n \cdot m) \cdot T_F$ for full H
  - Space $O(T_F)$ for the stack required by the RM

```
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

Forward over Reverse Mode

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

$$\mathrm{S} = \begin{pmatrix} \mathrm{I}_{n_x} & 0_{n_p} \\ 0_{n_p} & \mathrm{I}_{n_p} \\ 0_{n_q} & 0_{n_q} \end{pmatrix} \in \mathbb{R}^{n \times (n_x + n_p)}$$

  - With the example function F we could even use compression (adding together the $\mathbf{x}$– and $\mathbf{p}$–columns)
- Costs:
  - Time $O((n_x + n_p) \cdot m) \cdot T_F$ for the desired sub blocks of H

```
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
  $H_{i,j} = \frac{1}{2}\left(D^2_{\mathbf{e}_i+\mathbf{e}_j}F(\mathbf{X}) - D^2_{\mathbf{e}_i}F(X) - D^2_{\mathbf{e}_j}F(X)\right), \quad i \neq j$
  [Griewank & Walther, 2008]
    - For full H need $n + \frac{n\cdot(n+1)}{2}$ derivative directions
- Costs:
    - Time $O(n^2)\cdot T_F$ for full H
    - Space $O(n^2)\cdot M_F$

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

Mixed 2nd 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`

```
alias ac='adimat-client -F'
ac -ix      -d1                  -odx_F.m    F.m
ac -ig_x,x  -d1 -sgradprefix=h_  -odx_dx_F.m dx_F.m
ac -iq      -d1 -sgradprefix=h_  -odq_dx_F.m dx_F.m
ac -ip      -d1                  -odp_F.m    F.m
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^{nd} Order Directional Derivatives

```
h_g_x = zeros(size(x)); h_x = h_g_x; g_x = h_x;
for i=1:numel(x), for j=1:i
    h_x(i) = 1; g_x(j) = 1;
    h_g_f = dx_dx_F(h_g_x, g_x, h_x, x, p, q);
    dF_dxdx(:,i,j) = h_g_f(:);
    dF_dxdx(:,j,i) = h_g_f(:);
    h_x(i) = 0; g_x(j) = 0;
  end end
h_q = zeros(size(q)); g_x = zeros(size(x));
for i=1:numel(x), for j=1:numel(q)
    g_x(i) = 1; h_q(j) = 1;
    h_g_f = dq_dx_F(g_x, x, p, h_q, q);
    dF_dqdx(:,i,j) = h_g_f(:);
    g_x(i) = 0; h_q(j) = 0;
  end end % likewise for dF_dqdp
```

Complex Variable Method over FM

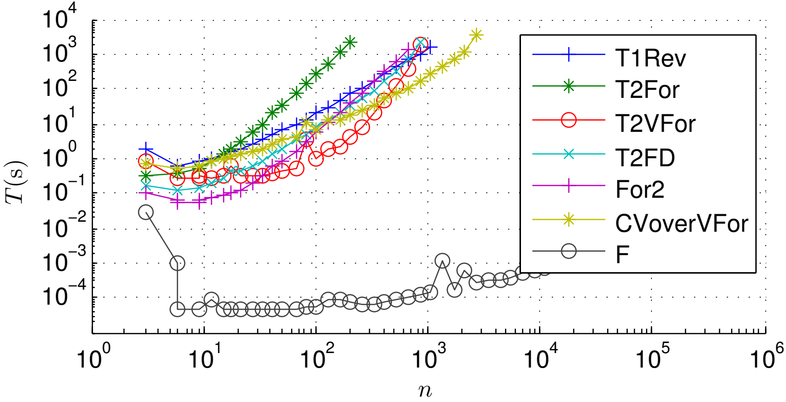
- First order FM to compute Jacobian
- Apply complex variable (CV) method on top of that
  - Only applicable if F is real analytic
  - Very precise and efficient approximation to derivatives

```
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 $\mathrm{H} \cdot \mathbf{v}$ 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 2nd 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 2nd 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
Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation (SCAM), 2002

References II

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*