Higher Order Forward and Reverse Mode on Matrices

EuroAD 2008, Oxford

Sebastian F. Walter
sebastian.walter@gmail.com

HU Berlin

Monday, 23’th November 2008
Context of this Talk

Theory

Griewank & Walther, Evaluating Derivatives
Giles, Collected Matrix Derivative Results for Forward and Reverse Mode AD

Application/Problem

Padulo, Robust Aircraft Conceptual Design Using AD in Matlab
Koerker, Optimal Design of Experiments

This talk:
Higher Order Derivatives of Matrix Operations

Implementation

ALGOPY

python powered
Goal of this Talk:

- ODOE objective function $\Phi$
  (parameters unconstrained):

  \[
  \Phi : \mathbb{R}^{N_q} \rightarrow \mathbb{R} \\
  q \mapsto \Phi(q) = \text{tr} \left( (J^T(q) J(q))^{-1} \right) \\
  \in \mathbb{R}^{N_M \times N_p}
  \]

  $q \in \mathbb{R}^{N_q}$: control variables, $J$: sensitivities of measurement function

Goal of this talk:

- Show how to compute $\nabla_q \Phi(q)$, $\nabla^2_q \Phi$, $\nabla^3_q$, etc. by algorithmic differentiation
- need to differentiate matrix operations!
Motivation for Higher Derivatives of Matrix Operations

Figure: Right local minimum is favorable when $q$ varies!

$q$-robust objective function

$$\min_{\tilde{q} \in \mathbb{R}^{Nq}} \mathbb{E}_{\tilde{q}}[\Phi(q)] = \min_{q \in \mathbb{R}^{Nq}} \Phi(q) + \text{tr}(H\Sigma) + \mathbb{E}_q[\mathcal{O}(\|q - \bar{q}\|^4)],$$

where $q \sim \mathcal{N}(\bar{q}, \Sigma^2)$, $H = \nabla^2_q \Phi$
Evaluating Derivatives of Scalar Operations
Forward Mode

Univariate Taylor Propagation

\[ [f] = \sum_{d=0}^{D} f_d t^d = \sum_{d=0}^{D} \frac{1}{d!} \frac{d^d}{dt^d} f(\sum_{c=0}^{D} x_c t^c) \bigg|_{t=0} t^d \]

= \ f([x])

■ generalization from functions \( f : \mathbb{R} \rightarrow \mathbb{R} \) to functions \( f : \mathbb{P}_D \rightarrow \mathbb{P}_D \) acting on the ring of truncated Taylor polynomials \( \mathbb{P}_D \).

Define operator \( P_D \)

\[ P_D(f(x)) \ := \ f([x]) \]
Reverse Mode

partial evaluation

\[ d\bar{f}f = \bar{f}df(x) = \bar{f} \frac{\partial f}{\partial x} dx \]

\[ :\equiv \bar{x} \]

**Example:** Gradient of \( f(g(x), y) = g(x)y = x^2y \):

\[ df(g, y) = \frac{\partial f}{\partial z}(z, y) \bigg|_{z=g(x)} \text{dg} + \frac{\partial f}{\partial y} \text{dy} \]

\[ = y \underbrace{\text{dg}}_{\equiv \bar{g}} + \underbrace{g \text{dy}}_{\equiv \bar{y}} \]

\[ = \bar{g}2x \text{dx} + \bar{y} \text{dy} \equiv \bar{x} \]

With \( \bar{f} = 1 \) we obtain the gradient

\[ \nabla f = (\bar{x}, \bar{y})^T = (2yx, x^2)^T \]
Combining Forward and Reverse

operators interchange:

\[
d P_D f \quad \overset{(*)}{=} \quad P_D df\]

forward then reverse

\[
P_D : C^D(\mathbb{R}^N, \mathbb{R}^M) \to C^1(\mathbb{P}_D^N, \mathbb{P}_D^M),
\]

\[
\mathbb{P} \text{ ring of truncated Taylor polynomials } [x] = [x_0, x_1, \ldots, x_D] = \sum_{d=0}^D x_d t^d.
\]

\[
df(x) : T_x \mathbb{R}^N \to T_{f(x)} \mathbb{R}^M \text{ differential, i.e. mapping between tangent spaces.}
\]

example: \[
\frac{\partial^2 \sin(x_0)}{\partial x^2} \times x_1, \quad [x] = [x_0, x_1] = x_0 + x_1 t
\]

\[
d \sin([x]) \quad \overset{(*)}{=} \quad \cos([x])
\]

\[
= [\cos(x_0), -\sin(x_0)x_1]
\]
Evaluating Derivatives of Matrix Operations
Back to ODOE problem

\[ \mathbb{R}^{Nq} \ni q \mapsto \Phi(q) = \text{tr} \left( (J^T(q) J(q))^{-1} \right) \]

Possibility 1: Matrices of Taylor Polynomials

\[
[A] = \begin{bmatrix}
[A_{11}] & \ldots & [A_{1N}] \\
\vdots & \ddots & \vdots \\
[A_{M1}] & \ldots & [A_{MN}]
\end{bmatrix}, \quad [A_{nm}] \in \mathbb{P}
\]

Possibility 2: Taylor Polynomials of Matrices

\[
\mathbb{P}^{M \times N} \ni [A] = [A_0, A_1, \ldots, A_D] = \sum_{d=0}^{D} A_d t^d
\]
Reverse on Matrices

**Objective function** \( \phi : \mathbb{R}^{Nq} \rightarrow \mathbb{R} \)

\[
\tilde{\Phi} \in \mathbb{R} \quad \text{d} \Phi( \begin{bmatrix}  \mathbf{X} \end{bmatrix} ) = \sum_{n,m} \tilde{\Phi} \frac{\partial \Phi}{\partial X_{nm}} \text{d}X_{nm}
\]

\[
= \text{tr} \left( \tilde{\Phi} \begin{bmatrix} \frac{\partial \Phi}{\partial X_{11}} & \cdots & \frac{\partial \Phi}{\partial X_{1N}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \Phi}{\partial X_{M1}} & \cdots & \frac{\partial \Phi}{\partial X_{MN}} \end{bmatrix} \begin{bmatrix} \text{d}X_{11} & \cdots & \text{d}X_{1M} \\ \vdots & \ddots & \vdots \\ \text{d}X_{N1} & \cdots & \text{d}X_{NM} \end{bmatrix} \right)
\]

\[
= \text{tr}(\tilde{\mathbf{X}}\text{d}X)
\]

**Interpretation:** \( \bar{X}_{nm} = \frac{\partial \Phi}{\partial X_{nm}} \)
Example: Higher Order Derivatives of the Matrix Inversion
Forward: \( Y = X^{-1} \)

\[
[Y] = [Y_0, Y_1, \ldots, Y_D] = [X]^{-1}
\]

\[\Leftrightarrow \quad I = [X][Y] \]

\( t^0 : \)

\( Y_0 = X_0^{-1} \)

\( t^d : \)

\[ Y_d = -Y_0 \left( \sum_{e=1}^{d} X_e Y_{d-e} \right) \quad d = 0, \ldots, D \]

Reverse: \( Y = X^{-1} \Leftrightarrow XY = I \)

\[
0 = dI = d(XY)
\]

\[= (dX)Y + XdY \]

\[\Leftrightarrow dY = -YdXY \]

\[\text{tr}(\bar{Y}dY) = \text{tr}(-\bar{Y}YdXY) \]

\[= \text{tr}(-Y\bar{Y}YdX) \]

\[= :\bar{X}^T \]
Combination: Forward + Reverse for $Y = X^{-1}$

\[
\begin{align*}
\text{tr}([\bar{Y}]d[Y]) &= \text{tr}(-[Y][\bar{Y}][Y]d[X]) \\
&= :[,X]^T \\
&= \text{tr}([\bar{X}]dX)
\end{align*}
\]

- Only typical matrix operations: $\Rightarrow$ Linear Algebra Packages
- No reevaluating of the computational graph necessary!
Problem with Matrices of Taylor Polynomials

- Operator Overloading: (CppAD, ADOLC)
  1. differentiate existing algos: Retaping necessary (very slow)
  2. my self-made algorithms: order 100 slower than ATLAS and likely to be buggy.
- Source Trafo: no differentiated LAPACK code available (e.g. with Tapenade).

Advantage of Taylor polynomials of Matrices

1. natural for Matlab
2. implicit checkpointing
3. can use highly **optimized linear algebra packages** (Atlas, etc.)
4. No problem with algorithms that use **pivoting** (matrix inversion)!
All the theory presented here is implemented in ALGOPY, a Python module to differentiate complex algorithms written in Python.

- Uses operator overloading on scalars (arbitrary order) and matrices (currently second order), forward and reverse modes.
- Big unit test: over 1500 lines of code, some examples (Newton’s method on matrices, ODOE example).
- Support for numpy functions, in particular: \texttt{dot}, \texttt{trace}, \texttt{inv}, \texttt{prod}, \texttt{sum}.
- Early alpha version is available at http://github.com/b45ch1/algopy.
\[
\begin{align*}
\text{cg} &= \text{CGraph()} \\
\text{FA} &= \text{Function (Mtc(A, Adot))} \\
\text{FB} &= \text{Function (Mtc(B, Bdot))} \\
\text{FA} &= \text{FA} \ast \text{FB}; \quad \text{FA} = \text{dot (FA, FB)} + \text{FA} \cdot \text{T} \\
\text{FA} &= \text{FB} + \text{FA} \ast \text{FB}; \quad \text{FB} = \text{inv (FA)}; \quad \text{FB} = \text{FB} \cdot \text{T} \\
\text{FC} &= \text{Function ([FA, FB], [FB, FA])} \\
\text{FTR} &= \text{trace (FC)} \\
\text{cg.plot(filename = 'computational_graph_circo.svg', method = 'circo')} \\
\text{g} &= \text{gradient (cg, [A, B])}
\end{align*}
\]
Applied to ODOE example

**Parameter Estimation**

- Red dots: measurements
- Blue line: initial guess
- Blue dots: true
- Green line: estimated
- Green dots: measurement model

**Optimal Design of Experiments**

- Black line: \( \Phi(q) \)
- Green dot: computed optimal solution

\[
\begin{align*}
\dot{x}(t) &= p_2 + qx(t) \\
x(0) &= p_1 \\
F &= (x(t_1), \ldots, x(t_{Nm})) \\
J &= \frac{dF}{dp} \\
\min_q \Phi(q) &= \min_q \text{tr}(J^T(q) J(q))^{-1}
\end{align*}
\]
Summary and Outlook:

- Implement arbitrary order derivatives on matrices
- Implement seamless connection between scalar and matrix mode
- Improve performance (goal: factor 20 slower than ADOLC on scalars)
Collected Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation, Mike B. Giles, Advances in Automatic Differentiation, Lecture Notes in Computational Science and Engineering, 2008


ALGOPY, a Python module to differentiate complex algorithms on scalars and matrices, http://github.com/b45ch1/algopy