Higher Order Forward and Reverse Mode on Matrices

EuroAD 2008, Oxford

Sebastian F. Walter
sebastian.walter@gmail.com

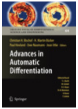
HU Berlin

Monday, 23’th November 2008

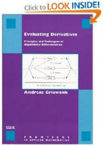
Context of this Talk

Theory

Giles, Collected Matrix Derivative Results for Forward and Reverse Mode AD



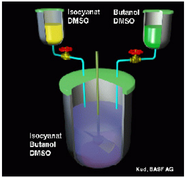
Griewank & Walther, Evaluating Derivatives



Application/Problem

Padulo, Robust Aircraft Conceptual Design Using AD in Matlab

Koerkel, Optimal Design of Experiments



This talk:

Higher Order Derivatives of Matrix Operations

python powered

ALGOPY

Implementation

Goal of this Talk:

- **ODOE objective function** Φ
  (parameters unconstrained):

$$\begin{aligned} \Phi : \mathbb{R}^{N_q} &\rightarrow \mathbb{R} \\ q &\mapsto \Phi(q) = \operatorname{tr}\left(\left(J^T(q) \underbrace{J(q)}_{\in \mathbb{R}^{N_M \times N_p}} \right)^{-1} \right) \end{aligned}$$

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

- **Goal of this talk**:
  - Show how to compute $\nabla_q \Phi(q)$, $\nabla_q^2 \Phi$, ∇_q^3, etc. by ***algorithmic differentiation***
  - need to **differentiate matrix operations**!

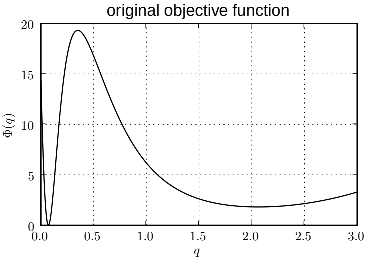


Figure: Right local minimum is favorable when q varies!

q-robust objective function

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

where $q \sim \mathcal{N}(\bar{q}, \Sigma^2)$, $H = \nabla_q^2 \Phi$

Evaluating Derivatives of Scalar Operations

Forward Mode

Univariate Taylor Propagation

$$
\begin{aligned}
[f] = \sum_{d=0}^{D} f_d t^d &= \sum_{d=0}^{D} \frac{1}{d!} \frac{\mathrm{d}^d}{\mathrm{d}t^d} f\Big(\sum_{c=0}^{D} x_c t^c\Big)\Big|_{t=0} t^d \\
&= f([x])
\end{aligned}
$$

- generalization from functions $f : \mathbb{R} \to \mathbb{R}$ to functions $f : \mathbb{P}_D \to \mathbb{P}_D$ 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

$$\mathrm{d}\bar{f} f = \bar{f}\mathrm{d}f(x) \quad = \quad \underbrace{\bar{f}\frac{\partial f}{\partial x}}_{:=\bar{x}} \mathrm{d}x$$

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

$$\begin{aligned}
\mathrm{d}f(g,y) &= \frac{\partial f}{\partial z}(z,y)\Big|_{z=g(x)} \mathrm{d}g + \frac{\partial f}{\partial y}\mathrm{d}y \\
&= \underbrace{y}_{=:\bar{g}} \mathrm{d}g + \underbrace{g}_{\bar{y}} \mathrm{d}y \\
&= \underbrace{\bar{g}2x}_{=:\bar{x}} \mathrm{d}x + \bar{y}\mathrm{d}y
\end{aligned}$$

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

$$\nabla f \quad = \quad (\bar{x}, \bar{y})^T = (2yx, x^2)^T$$

Combining Forward and Reverse

- operators interchange:

$$\underbrace{\mathrm{d}P_D f}_{\text{forward then reverse}} \stackrel{(*)}{=} P_D \mathrm{d}f$$

$P_D : C^D(\mathbb{R}^N, \mathbb{R}^M) \to C^1(\mathbb{P}_D^N, \mathbb{P}_D^M)$,
$\mathbb{P}$ ring of truncated Taylor polynomials $[x] = [x_0, x_1, \dots, x_D] = \sum_{d=0}^{D} x_d t^d$.
$\mathrm{d}f(x) : T_x\mathbb{R}^N \to T_{f(x)}\mathbb{R}^M$ differential, i.e. mapping between tangent spaces.

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

$$\begin{aligned} \mathrm{d}\sin([x]) &\stackrel{(*)}{=} \cos([x]) \\ &= [\cos(x_0), -\sin(x_0)x_1] \end{aligned}$$

Evaluating Derivatives of Matrix Operations

Back to ODOE problem

$$\mathbb{R}^{N_q} \ni q \quad \mapsto \quad \Phi(q) = \mathrm{tr}\left(\left(J^T(q) \underbrace{J(q)}_{\in \mathbb{R}^{N_M \times N_p}} \right)^{-1} \right)$$

Possibility 1: Matrices of Taylor Polynomials

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

Possibility 2: Taylor Polynomials of Matrices

$$\mathbb{P}^{M \times N} \ni [A] \quad = \quad [A_0, A_1, \dots, A_D] = \sum_{d=0}^{D} A_d t^d$$

Reverse on Matrices

- Objective function $\phi : \mathbb{R}^{N_q} \to \mathbb{R}$

$$
\begin{aligned}
\underbrace{\bar{\Phi}}_{\in\mathbb{R}} \mathrm{d}\Phi(\underbrace{X}_{\in\mathbb{R}^{N\times M}}) &= \sum_{n,m} \bar{\Phi} \frac{\partial \Phi}{\partial X_{nm}} \mathrm{d}X_{nm} \\
&= \mathrm{tr}\left(\underbrace{\bar{\Phi} \begin{bmatrix} \frac{\partial \Phi}{\partial X_{11}} & \cdots & \frac{\partial \Phi}{\partial X_{1N}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \Phi}{\partial X_{M1}} & \cdots & \frac{\partial \Phi}{\partial X_{MN}} \end{bmatrix}}_{=:\bar{X} \in \mathbb{R}^{M\times N}} \underbrace{\begin{bmatrix} \mathrm{d}X_{11} & \cdots & \mathrm{d}X_{1M} \\ \vdots & \ddots & \vdots \\ \mathrm{d}X_{N1} & \cdots & \mathrm{d}X_{NM} \end{bmatrix}}_{=:\mathrm{d}X \in \mathbb{R}^{N\times M}} \right) \\
&= \mathrm{tr}(\bar{X}\,\mathrm{d}X)
\end{aligned}
$$

- Interpretation: $\bar{X}_{nm} = \frac{\partial \Phi}{\partial X_{nm}}$

Example: Higher Order Derivatives of the Matrix Inversion

Forward: $Y = X^{-1}$

$$
\begin{aligned}
& [Y] = [Y_0, Y_1, \dots, Y_D] = [X]^{-1} \\
\Leftrightarrow \quad & I \stackrel{!}{=} [X][Y] \\
t^0: \quad & Y_0 = X_0^{-1} \\
t^d: \quad & Y_d = -Y_0 \left(\sum_{e=1}^{d} X_e Y_{d-e} \right) \qquad d = 0, \dots, D
\end{aligned}
$$

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

$$
\begin{aligned}
0 = \mathrm{d}I &= \mathrm{d}(XY) \\
&= (\mathrm{d}X)Y + X\mathrm{d}Y \\
\Leftrightarrow \mathrm{d}Y &= -Y\mathrm{d}XY \\
\mathrm{tr}(\bar{Y}\mathrm{d}Y) &= \mathrm{tr}(-\bar{Y}Y\mathrm{d}XY) \\
&= \mathrm{tr}(\underbrace{-Y\bar{Y}Y}_{=:\bar{X}^T}\mathrm{d}X)
\end{aligned}
$$

Combination: Forward + Reverse for $Y = X^{-1}$

$$\begin{aligned} \mathrm{tr}([\bar{Y}]\mathrm{d}[Y]) &= \mathrm{tr}(\underbrace{-[Y][\bar{Y}][Y]}_{=:[\bar{X}]^T}\mathrm{d}[X]) \\ &= \mathrm{tr}([\bar{X}]\mathrm{d}X) \end{aligned}$$

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

Problem with Matrices of Taylor Polynomials

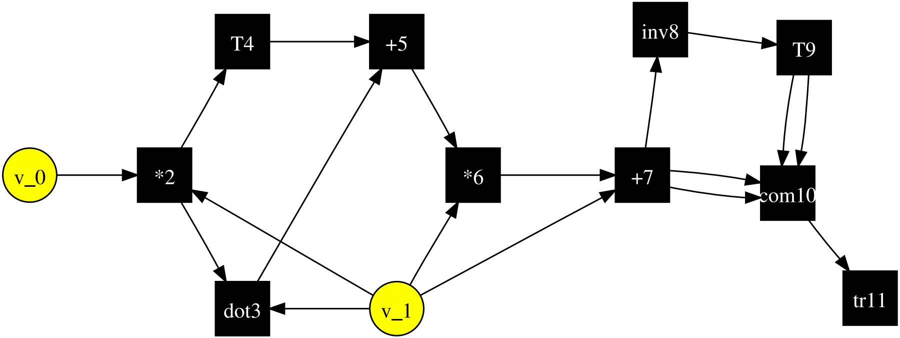
- Operator Overloading: (CppAD, ADOLC)
  1. differentiate existing algos: Retaping necessary(very slow)
  2. my self-made algorithms: order 100 slower than ATLAS and likely to be buggy.
- Source Trafo: no differentiated LAPACK code available (e.g. with Tapenade).

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 invesion)!

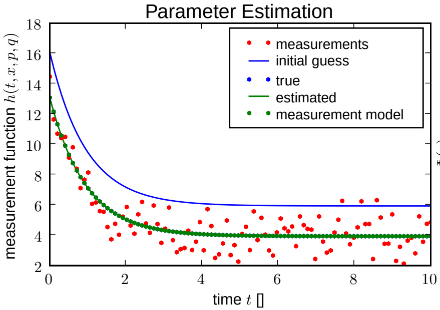
ALGOPY

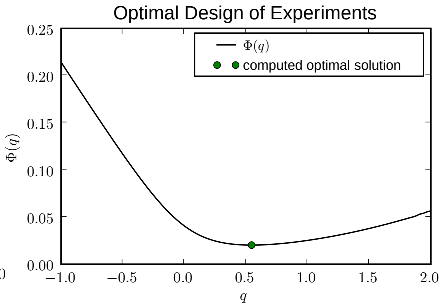
- All the theory presented here is implemented in ALGOPY a Python module to differentiate complex algorithms written in Python
- Uses operator overloading on scalars (arbitrary order) and matrices (currently second order), forward and reverse mode
- Big unit test: over 1500 lines of code, some examples (Newton's method on matrices, ODOE example)
- Support for numpy functions, in particular: `dot`, `trace`, `inv`, `prod`, `sum`
- Early alpha version is available at http://github.com/b45ch1/algopy




```
...
cg = CGraph()
FA = Function(Mtc(A,Adot))
FB = Function(Mtc(B,Bdot))
FA = FA*FB; FA = dot(FA,FB) + FA.T
FA = FB + FA * FB; FB = inv(FA); FB = FB.T
FC = Function([[FA,FB],[FB, FA]])
FTR = trace(FC)
cg.plot(filename = 'computational_graph_circo.svg', method = 'circo' )
g = gradient(cg,[A,B])
```

Applied to ODOE example





$$
\begin{aligned}
\dot{x}(t) &= p_2 + q x(t) \\
x(0) &= p_1 \\
F &:= (x(t_1), \dots, x(t_{N_m})) \\
J &:= \mathrm{d}F / \mathrm{d}p \\
\min_q \Phi(q) &= \min_q \mathrm{tr}(J^T(q) \underbrace{J(q)}_{\in \mathbb{R}^{N_M \times N_p}})^{-1}
\end{aligned}
$$

Summary and Outlook:

- Implement arbitrary order derivatives on matrices
- Implement seemless 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
- **Robust Aircraft Conceptual Design Using Automatic Differentation in Matlab** Mattia Padulo, Shaun A. Forth, Martin D. Guenov, Advances in Automatic Differentiation, Lecture Notes in Computational Science and Engineering, 2008
- **Evaluating Derivatives, Second Edition** Andreas Griewank, Andrea Walther, SIAM, 2008
- **ALGOPY, a Python module to differentiate complex algorithms on scalars and matrices** http://github.com/b45ch1/algopy