Mirko Franke, Jan Winkler
Faculty of Electrical and Computer Engineering, Institute of Control Theory

AdolC4Matlab – An Interface between MATLAB and ADOL-C for Applications in Nonlinear Control

Rostock, July 25, 2018
## Motivation

### Applications in Nonlinear Control

- **exact input-output linearization:**
  
  \[ h(x), L_f h(x), \ldots, L_f^n h(x), L_g L_f^{r-1} h(x) \]

- **nonlinear control with approximately linear tracking error:**
  
  \[ g(x), \text{ad}_f g(x), \ldots, \text{ad}^2_f g(x) \]

- **High-Gain observer, extended Luenberger observer:**
  
  \[ dh(x), dL_f h(x), \ldots, dL_f^n h(x) \]

- ...
# Motivation

Why Algorithmic Differentiation?

## Symbolic Differentiation
- efficient for derivatives of low order
- exponential expression growth for high order derivatives
- time-consuming computations

## Numeric Differentiation
- finite differences: cancellation and truncation errors
- not applicable for higher order derivatives
Motivation
Why Algorithmic Differentiation?

Algorithmic Differentiation

- function is given as algorithm
- application of elementary differentiation rules with chain rule
- intermediate values are floating point numbers
- no truncation errors (exact w.r.t. floating point numbers)
- no limitation of the derivative order
Outline

1. Algorithmic Differentiation
2. About the Toolbox
3. Example
4. Summary and Outlook
Algorithmic Differentiation
## Algorithmic Differentiation

### Forward Mode

**Example:** 
\[ z = F(x, y) = (\sin(x \cdot y) + x) \cdot (\sin(x \cdot y) - y) \]

<table>
<thead>
<tr>
<th>( F )</th>
<th>( \frac{\partial F}{\partial x} )</th>
<th>( \frac{\partial F}{\partial y} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>3.0</td>
<td>( \dot{x} )</td>
</tr>
<tr>
<td>( y )</td>
<td>4.0</td>
<td>( \dot{y} )</td>
</tr>
<tr>
<td>( v_1 = x \cdot y )</td>
<td>12.0</td>
<td>( \dot{v}_1 = \dot{x} y + \dot{y} x )</td>
</tr>
<tr>
<td>( v_2 = \sin(v_1) )</td>
<td>-0.5366</td>
<td>( \dot{v}_2 = \dot{v}_1 \cos(v_1) )</td>
</tr>
<tr>
<td>( v_3 = v_2 + x )</td>
<td>2.4634</td>
<td>( \dot{v}_3 = \dot{v}_2 + \dot{x} )</td>
</tr>
<tr>
<td>( v_4 = v_2 - y )</td>
<td>-4.5366</td>
<td>( \dot{v}_4 = \dot{v}_2 - \dot{y} )</td>
</tr>
<tr>
<td>( v_5 = v_3 \cdot v_4 )</td>
<td>-11.1755</td>
<td>( \dot{v}_5 = \dot{v}_3 v_4 + \dot{v}_4 v_3 )</td>
</tr>
<tr>
<td>( z = v_5 )</td>
<td>-11.1755</td>
<td>( \dot{z} = \dot{v}_5 )</td>
</tr>
</tbody>
</table>
Algorithmic Differentiation
Forward Mode – Implementation in C++

Replace floating point type `double` by a new class, e.g. `ddouble`:

```cpp
class ddouble
{
    double val;  // function value
    double der;  // derivative value
}
```

Overload all operations for additional derivative calculation:

```cpp
double operator * (ddouble x, ddouble y)
{
    ddouble z;
    z.val = x.val*y.val;    // multiplication
    z.der = x.der*y.val+x.val*y.der; // product rule
    return z;
}
```
Algorithmic Differentiation

Forward mode: directed derivative

\[ w = F'(x)v \]

for \( v = e_i \)
\[ \Rightarrow w \cong i\text{-th column of the Jacobian} \]

Reverse mode: weighted gradient

\[ \bar{a}^T = \bar{z}^T F'(x) \]

for \( \bar{z}^T = e_i^T \)
\[ \Rightarrow \bar{a}^T \cong i\text{-th row of the Jacobian} \]

Reverse Mode

- the chain rule is applied in the reverse order that the function is computed
- applied in the backpropagation algorithm for neural networks
Algorithmic Differentiation
Reverse Mode

\[ z = F(x, y) = (\sin(xy) + x)(\sin(xy) - y) \]

\[ \bar{v}_i = \bar{v}_i + \frac{\partial v_j}{\partial v_i}, \quad j > i \]

⇒ whole gradient in one pass

\[ \bar{x} = \frac{\partial F(x, y)}{\partial x}, \quad \bar{y} = \frac{\partial F(x, y)}{\partial y} \]

\[ x = 3.0 \]
\[ y = 4.0 \]
\[ v_1 = x \cdot y = 12.0 \]
\[ v_2 = \sin(v_1) = -0.5366 \]
\[ v_3 = v_2 + x = 2.4634 \]
\[ v_4 = v_2 - y = -4.5366 \]
\[ v_5 = v_3 \cdot v_4 = -11.1755 \]
\[ z = v_5 = -11.1755 \]
\[ \bar{v}_5 = \bar{z} = 1.0 \]
\[ \bar{v}_3 = \bar{v}_5 \cdot v_4 = -4.5366 \]
\[ \bar{v}_4 = \bar{v}_5 \cdot v_3 = 2.4634 \]
\[ \bar{v}_2 = \bar{v}_4 = 2.4634 \]
\[ \bar{y} = -\bar{v}_4 = -2.4634 \]
\[ \bar{v}_2 = \bar{v}_2 + \bar{v}_3 = -2.0732 \]
\[ \bar{x} = \bar{v}_3 = -4.5366 \]
\[ \bar{v}_1 = \bar{v}_2 \cdot \cos(v_1) = -1.7495 \]
\[ \bar{x} = \bar{x} + \bar{v}_1 \cdot y = -11.5346 \]
\[ \bar{y} = \bar{y} + \bar{v}_1 \cdot x = -7.7119 \]
Algorithmic Differentiation
Combination of Forward and Reverse Mode

Smooth map $F : \mathcal{M} \subset \mathbb{R}^n \to \mathbb{R}^m$ and truncated Taylor series

$$
\begin{align*}
  x(t) &= x_0 + x_1 t + \mathcal{O}(t^2), \\
  z(t) = F(x(t)) &= z_0 + z_1 t + \mathcal{O}(t^2),
\end{align*}
$$

$$
\begin{align*}
  x_k &= \frac{1}{k!} x^{(k)}(0) \in \mathbb{R}^n \\
  z_k &= \frac{1}{k!} z^{(k)}(0) \in \mathbb{R}^m
\end{align*}
$$

Forward mode: Taylor coefficients

$$
\begin{align*}
  z_0 &= F(x_0) \\
  z_1 &= F'(x_0) x_1
\end{align*}
$$

Reverse mode: Adjoints

$$
\begin{align*}
  a^T_0 &= \bar{z}^T \frac{\partial z_0}{\partial x_0} = \bar{z}^T F'(x_0) \\
  a^T_1 &= \bar{z}^T \frac{\partial z_1}{\partial x_0} = \bar{z}^T F''(x_0) x_1
\end{align*}
$$
Algorithmic Differentiation
ADOL-C

**ADOL-C:** Automatic Differentiation by OverLoading in C++

- first and higher derivatives of vector functions
- based on operator overloading — uses tapes
- drivers for:
  - forward and reverse calls
  - ordinary differential equations
  - sparse Jacobians and Hessians
  - Lie-derivatives
  - ...
- routines may be called from any programing language that can be linked with C
About the Toolbox
About the Toolbox
Distinction from other Tools

Available MATLAB Tools
ADiMAT, ADMAT, TOMLAB/MAD, ...
- mostly only first and second order derivatives
- target group: optimization, optimal control

ADOL-C4MATLAB
- uses the package ADOL-C
- designed for applications in nonlinear control
# About the Toolbox

## Functionality

### ADOL-C Wrappers

- madForward, madReverse
- madFunction
- madGradient, madJacobian, madHessian
- madLieScalar, madLieGradient, madLieBracket, ...

### Control Engineering related Functions

- madHighGainObs, madExtLuenObs
- madFeedbackLin, madCompTorqueControl, ...
## About the Toolbox

### Workflow

<table>
<thead>
<tr>
<th>C++ Code</th>
<th>MATLAB-Drivers</th>
</tr>
</thead>
</table>
| \( y[0] = x[0] \times \sin(x[0] \times x[1]) + x[0] \times x[1]; \) | - ADOL-C wrappers: *forward, reverse, jacobian, hessian*, ...
|                                               | - control engineering related: *High-Gain observer, extended Luenberger observer, feedback control, computed torque*, ...
| modification and compilation                  | - extendable by user defined functions               |
| **MEX-Function**                              |                                                      |
| generates the ADOL-C tape                     |                                                      |
| evaluation                                    |                                                      |
| **Tape with unique ID**                       |                                                      |
| trace of operations/evaluations               |                                                      |
About the Toolbox
Using the Toolbox

- code snippet is given as file (no headers, ...)
- load the settings:
  \[ S = \text{madSettings}(); \]
- generation of the corresponding MEX function:
  \[ \text{TapeId} = \text{madTapeCreate}(n, m, \text{keep}, \text{filename}, S); \]
  \( n, m \) ... number of independent and dependent variables
  \( \text{keep} \) ... prepare for an immediate call of the reverse mode
- use the toolbox drivers, e.g.:
  \[ \text{F} = \text{madFunction}(\text{TapeId}, \text{X}); \]
  \[ \text{J} = \text{madJacobian}(\text{TapeId}, \text{X}); \]
  \( \text{X} \) ... point of evaluation
- close the tape:
  \[ \text{madTapeClose}(\text{TapeId}); \]
- reopen the tape:
  \[ \text{TapeId} = \text{madTapeOpen}(.\text{filename}); \]
## About the Toolbox

### Use Cases

- derivatives are only required numerically
- highly nonlinear functions/systems
- calculation of first and higher order derivatives
- calculation of Lie derivatives
- algorithms in nonlinear control (controller, observer)
Example
Example

Consider the iterative function $z = f_k(x) = f(g_k(x))$ of order $k$:

$$
g_0(x) = x$$

$$g_i(x) = \sin(g_{i-1}(x) + 0.1g_{i-1}^3(x)) \quad i = 1 \ldots k$$

$$f_k(x) = g_k^4(x) \cdot \log(g_k^2(x)) \cdot \exp\left(g_k^3(x) - \tanh(g_k(x))\right) \quad \Rightarrow \quad 9 + 5k \text{ operations}$$

Runtime comparison – algorithmic vs. symbolic computation:

- MATLAB functions: generation, evaluation
- Algorithmic: tape generation, derivative computation using the tape
Example

Tape/Function generation

- algorithmic
- symbolic – $f_k(x)$
- symbolic – $f'_k(x)$
- symbolic – $f''_k(x)$

Computation time in s

Order $k$ of iteration

AdolC4Matlab
Institute of Control Theory // M. Franke
Rostock, July 25, 2018
Example

Evaluation of $f_k(x)$

Evaluation of $f_k''(x)$

Computation time in μs

Order $k$ of iteration
Summary and Outlook
### Summary and Outlook

#### Conclusion

- alternative to symbolic computation of derivatives
- toolbox for MATLAB and Octave
- available on GitLab: [https://gitlab.com/mfranke/ADOL-C4MATLAB](https://gitlab.com/mfranke/ADOL-C4MATLAB)

#### There's still work to do!

- code optimization
- extension for other applications