TECHNISCHE UNIVERSITÄT DRESDEN

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:

$$h(\boldsymbol{x}), L_{\boldsymbol{f}} h(\boldsymbol{x}), \dots, L_{\boldsymbol{f}}^{r} h(\boldsymbol{x}), L_{\boldsymbol{g}} L_{\boldsymbol{f}}^{r-1} h(\boldsymbol{x})$$

- nonlinear control with approximately linear tracking error:

$$g(\boldsymbol{x}), \mathrm{ad}_{-\boldsymbol{f}} \boldsymbol{g}(\boldsymbol{x}), \dots, \mathrm{ad}_{-\boldsymbol{f}}^{2n-1} \boldsymbol{g}(\boldsymbol{x})$$

- High-Gain observer, extended Luenberger observer:

$$\mathrm{d}h(\boldsymbol{x}), \mathrm{d}L_{\boldsymbol{f}} h(\boldsymbol{x}), \dots, \mathrm{d}L_{\boldsymbol{f}}^{n-1} h(\boldsymbol{x})$$

- ...

# 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

# Algorithmic Differentiation

# Algorithmic Differentiation
## Forward Mode

Example: $z = F(x, y) = \big(\sin(x \cdot y) + x\big) \cdot \big(\sin(x \cdot y) - y\big)$

| $F$ | | | $\frac{\partial F}{\partial x}$ | $\frac{\partial F}{\partial y}$ |
|---|---|---|---|---|
| $x$ | 3.0 | $\dot{x}$ | 1.0 | 0.0 |
| $y$ | 4.0 | $\dot{y}$ | 0.0 | 1.0 |
| $v_1 = x \cdot y$ | 12.0 | $\dot{v}_1 = \dot{x}y + \dot{y}x$ | 4.0 | 3.0 |
| $v_2 = \sin(v_1)$ | −0.5366 | $\dot{v}_2 = \dot{v}_1 \cos(v_1)$ | 3.3754 | 2.5316 |
| $v_3 = v_2 + x$ | 2.4634 | $\dot{v}_3 = \dot{v}_2 + \dot{x}$ | 4.3754 | 2.5316 |
| $v_4 = v_2 - y$ | −4.5366 | $\dot{v}_4 = \dot{v}_2 - \dot{y}$ | 3.3754 | 1.5316 |
| $v_5 = v_3 \cdot v_4$ | −11.1755 | $\dot{v}_5 = \dot{v}_3 v_4 + \dot{v}_4 v_3$ | −11.5345 | −7.7119 |
| $z = v_5$ | −11.1755 | $\dot{z} = \dot{v}_5$ | −11.5345 | −7.7119 |

# Algorithmic Differentiation
## Forward Mode – Implementation in C++

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

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

Overload all operations for additional derivative calculation:

```
ddouble 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

$$\boldsymbol{w} = \boldsymbol{F}'(\boldsymbol{x})\boldsymbol{v}$$

for $\boldsymbol{v} = \boldsymbol{e}_i$
$\Rightarrow \boldsymbol{w} \mathrel{\widehat{=}} i$-th column of the Jacobian

Reverse mode: weighted gradient

$$\bar{\boldsymbol{a}}^T = \bar{\boldsymbol{z}}^T \boldsymbol{F}'(\boldsymbol{x})$$

for $\bar{\boldsymbol{z}}^T = \boldsymbol{e}_i^T$
$\Rightarrow \bar{\boldsymbol{a}}^T \mathrel{\widehat{=}} i$-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

$$
\begin{aligned}
z &= F(x, y) \\
&= \big(\sin(xy) + x\big)\big(\sin(xy) - y\big)
\end{aligned}
$$

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

$\Rightarrow$ whole gradient in one pass

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

$$
\begin{aligned}
x &= 3.0 \\
y &= 4.0 \\
v_1 &= x \cdot y = 12.0 \\
v_2 &= \sin(v_1) = -0.5366 \\
v_3 &= v_2 + x = 2.4634 \\
v_4 &= v_2 - y = -4.5366 \\
v_5 &= v_3 \cdot v_4 = -11.1755 \\
z &= v_5 = -\mathbf{11.1755} \\
\bar{v}_5 &= \bar{z} = 1.0 \\
\bar{v}_3 &= \bar{v}_5 \cdot v_4 = -4.5366 \\
\bar{v}_4 &= \bar{v}_5 \cdot v_3 = 2.4634 \\
\bar{v}_2 &= \bar{v}_4 = 2.4634 \\
\bar{y} &= -\bar{v}_4 = -2.4634 \\
\bar{v}_2 &= \bar{v}_2 + \bar{v}_3 = -2.0732 \\
\bar{x} &= \bar{v}_3 = -4.5366 \\
\bar{v}_1 &= \bar{v}_2 \cdot \cos(v_1) = -1.7495 \\
\bar{x} &= \bar{x} + \bar{v}_1 \cdot y = -\mathbf{11.5346} \\
\bar{y} &= \bar{y} + \bar{v}_1 \cdot x = -\mathbf{7.7119}
\end{aligned}
$$

# Algorithmic Differentiation
## Combination of Forward and Reverse Mode

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

$$\begin{aligned} \boldsymbol{x}(t) &= \boldsymbol{x}_0 + \boldsymbol{x}_1 t + \mathcal{O}(t^2), \quad && \boldsymbol{x}_k = \tfrac{1}{k!}\boldsymbol{x}^{(k)}(0) \in \mathbb{R}^n \\ \boldsymbol{z}(t) = \boldsymbol{F}(\boldsymbol{x}(t)) &= \boldsymbol{z}_0 + \boldsymbol{z}_1 t + \mathcal{O}(t^2), \quad && \boldsymbol{z}_k = \tfrac{1}{k!}\boldsymbol{z}^{(k)}(0) \in \mathbb{R}^m \end{aligned}$$

### Forward mode: Taylor coefficients

$$\begin{aligned} \boldsymbol{z}_0 &= \boldsymbol{F}(\boldsymbol{x}_0) \\ \boldsymbol{z}_1 &= \boldsymbol{F}'(\boldsymbol{x}_0)\,\boldsymbol{x}_1 \end{aligned}$$

### Reverse mode: Adjoints

$$\begin{aligned} \mathbf{a}_0^T &= \bar{\boldsymbol{z}}^T \frac{\partial \boldsymbol{z}_0}{\partial \boldsymbol{x}_0} = \bar{\boldsymbol{z}}^T \boldsymbol{F}'(\boldsymbol{x}_0) \\ \mathbf{a}_1^T &= \bar{\boldsymbol{z}}^T \frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{x}_0} = \bar{\boldsymbol{z}}^T \boldsymbol{F}''(\boldsymbol{x}_0)\,\boldsymbol{x}_1 \end{aligned}$$

# Algorithmic Differentiation
## ADOL-C

**ADOL-C: A**utomatic **D**ifferentiation by **O**ver**L**oading 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 oder 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

**C++ Code**

```
y[0] = x[0]*sin(x[0]*x[1]) + x[0]*x[1];
```

↓ modification and compilation

**MEX-Function**

generates the ADOL-C tape

↓ evaluation

**Tape with unique ID**

trace of operations/evaluations

usage

**MATLAB-Drivers**

- ADOL-C wrappers:
  *forward, reverse, jacobian, hessian,*
  ...
- control engineering related:
  *High-Gain observer, extended Luenberger observer, feedback control, computed torque, ...*
- extendable by user defined functions

# About the Toolbox
## Using the Toolbox

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

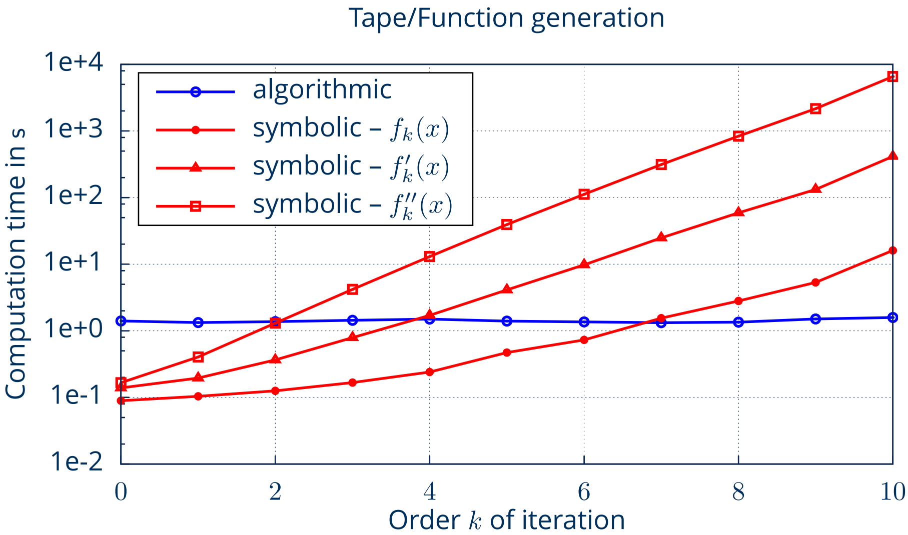
Consider the iterative function $z = f_k(x) = f(g_k(x))$ of order $k$:

$$\begin{aligned} g_0(x) &= x \\ g_i(x) &= \sin\left(g_{i-1}(x) + 0.1 g_{i-1}^3(x)\right), \quad i = 1 \dots k \\ f_k(x) &= g_k^4(x) \cdot \log(g_k^2(x)) \cdot \exp\left(g_k^3(x) - \tanh\left(g_k(x)\right)\right) \quad \Rightarrow \quad 9 + 5k \text{ operations} \end{aligned}$$

Runtime comparison – algorithmic vs. symbolic computation:

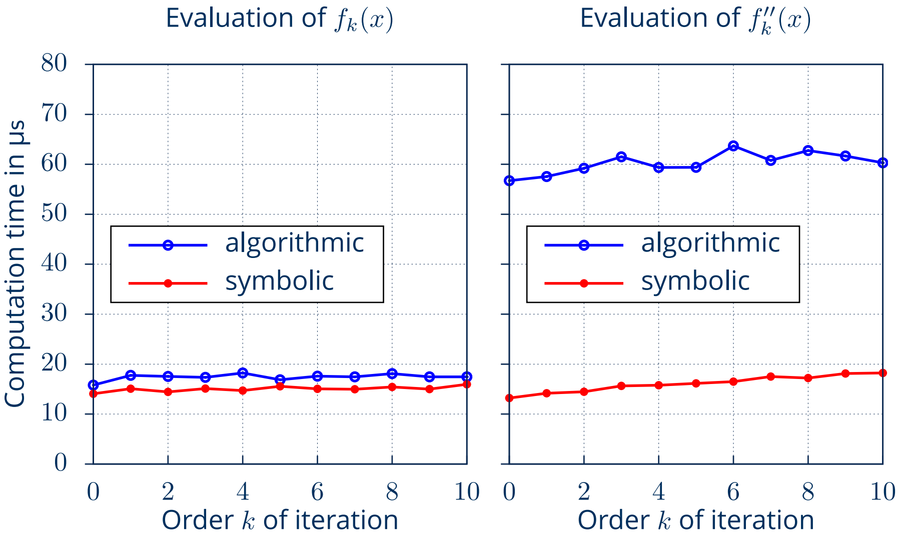
- MATLAB functions: generation, evaluation
- algorithmic: tape generation, derivative computation using the tape

# Example



Tape/Function generation
algorithmic
symbolic – $f_k(x)$
symbolic – $f'_k(x)$
symbolic – $f''_k(x)$
Computation time in s
1e+4
1e+3
1e+2
1e+1
1e+0
1e-1
1e-2
0
2
4
6
8
10
Order $k$ of iteration

# Example


Evaluation of $f_k(x)$
Evaluation of $f_k''(x)$
Computation time in µs
80
70
60
50
40
30
20
10
0
algorithmic
symbolic
0
2
4
6
8
10
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`

## There's still work to do!

- code optimization
- extension for other applications