Symbolic Calculation of Partial Derivatives for Linearization of Non-linear Modelica Models in OpenModelica

The world of Jacobian matrices

Willi Braun, Bernhard Bachmann and Lennart Ochel

Department of Applied Mathematics
University of Applied Sciences Bielefeld
33609 Bielefeld, Germany

2011-02-07

Motivation

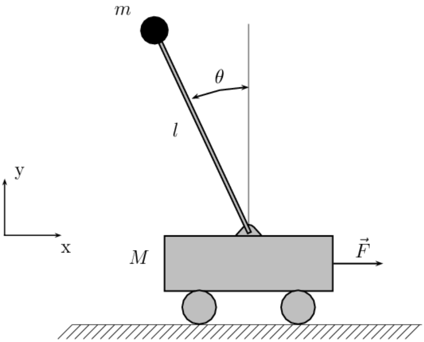
What are linear models useful for?

Motivation

What are linear models useful for?

For example

- Control technique
- Optimization
- Model analysis



Outline

Introduction: Linear Models

Derivation of Linear Models

Flatten Modelica model:

$$0 = F(\underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p}, t), \quad \underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix}$$

$\downarrow$ matching and sorting algorithm transform to

$$\underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \hat{F}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

This results in the state space equations:

$$\begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} \underline{h}(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ \underline{k}(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Introduction: Linear Models

Derivation of Linear Models

State-Space Equations

$$\begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Introduction: Linear Models

Derivation of Linear Models

State-Space Equations

$$\begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

by Taylor series approximation and cancelling the quadratic and higher order terms.

$\Downarrow$

Linearization

$$\underline{\dot{x}}(t) = A(t) * \underline{x}(t) + B(t) * \underline{u}(t)$$
$$\underline{y}(t) = C(t) * \underline{x}(t) + D(t) * \underline{u}(t)$$

Introduction: Linear Models

Derivation of Linear Models

State-Space Equations

$$\begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

by Taylor series approximation and cancelling the quadratic and higher order terms.

$\Downarrow$

Linearization

$$\underline{\dot{x}}(t) = A(t) * \underline{x}(t) + B(t) * \underline{u}(t)$$
$$\underline{y}(t) = C(t) * \underline{x}(t) + D(t) * \underline{u}(t)$$

For linear models the following matrices are needed:

$\Downarrow$

Jacobian matrices

$$A(t) = \frac{\partial \underline{h}}{\partial \underline{x}}$$
$$B(t) = \frac{\partial \underline{h}}{\partial \underline{u}}$$
$$C(t) = \frac{\partial \underline{k}}{\partial \underline{x}}$$
$$D(t) = \frac{\partial \underline{k}}{\partial \underline{u}}$$

Introduction: Differentiation

Methods for Differentiation

Common Methods

- Numerical Differentiation
- Automatic Differentiation
- Symbolic Differentiation

Introduction: Differentiation

Numerical Methods

Differentiation Methods

- Numerical
- Automatic
- Symbolic

Forward difference:

$$\dot{f}(x) = \lim_{\delta \to 0} \frac{(f(x+\delta) - f(x))}{\delta}$$

Drawback

Even if δ is optimal selected:

$$\left|\frac{\partial f(x)}{\partial x} - \frac{(f(x+\delta_{opt}) - f(x))}{\delta_{opt}}\right| \approx \sqrt{\epsilon_{RND}}$$

Some significat digits are lost by truncation.

Introduction: Differentiation

Automatic Differentiation

Differentiation Methods

- Numerical
- Automatic
- Symbolic

Introduction: Differentiation

Automatic Differentiation

Differentiation Methods

- Numerical
- Automatic
- Symbolic

Basic Differentiation Rules

Chain rule:

$$\nabla\phi(u) = \dot{\phi}(u)\nabla u$$

Arithmetic operations:

$$\begin{aligned}\nabla(u \pm v) &= \nabla u \pm \nabla v \\ \nabla(uv) &= u\nabla v + v\nabla u \\ \nabla(\frac{u}{v}) &= \frac{(\nabla u - \frac{u}{v}\nabla v)}{v}\end{aligned}$$

Introduction: Differentiation

Automatic Differentiation

Basic Rules

Chain rule:

$$\nabla \phi(u) = \dot{\phi}(u) \nabla u$$

Arithmetic operations:

$$\begin{aligned} \nabla(u \pm v) &= \nabla u \pm \nabla v \\ \nabla(uv) &= u \nabla v + v \nabla u \\ \nabla\left(\frac{u}{v}\right) &= \frac{(\nabla u - \frac{u}{v} \nabla v)}{v} \end{aligned}$$

Example

$$y = f(x_1, x_2) = (x_1 x_2 + \sin(x_1))(x_2 + \cos(x_2))$$

Introduction: Differentiation

Automatic Differentiation

Basic Rules

Chain rule:

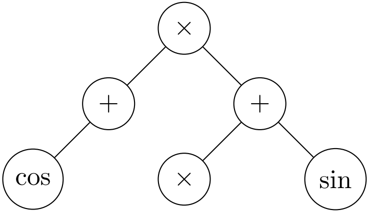
$$\nabla \phi(u) = \dot{\phi}(u) \nabla u$$

Arithmetic operations:

$$\begin{aligned} \nabla(u \pm v) &= \nabla u \pm \nabla v \\ \nabla(uv) &= u \nabla v + v \nabla u \\ \nabla\left(\frac{u}{v}\right) &= \frac{\left(\nabla u - \frac{u}{v} \nabla v\right)}{v} \end{aligned}$$

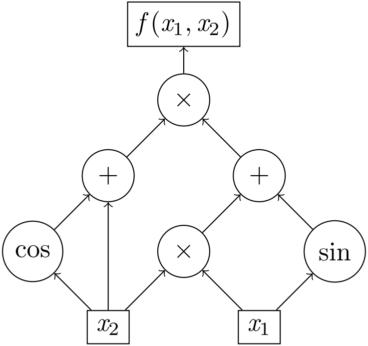
Example

$$y = f(x_1, x_2) = (x_1 x_2 + \sin(x_1))(x_2 + \cos(x_2))$$



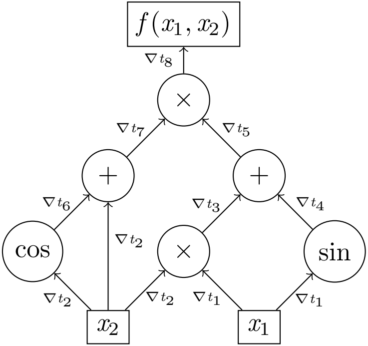
Introduction: Differentiation

Automatic Differentiation Example



Introduction: Differentiation

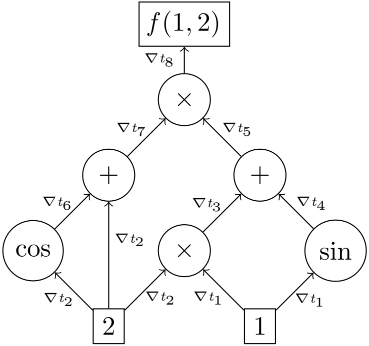
Automatic Differentiation Example



Operations	Differentiate(t_i, $\{x_1, x_2\}$)
$t_1 = x_1$	$\nabla t_1 = [1, 0]$
$t_2 = x_2$	$\nabla t_2 = [0, 1]$
$t_3 = t_1 t_2$	$\nabla t_3 = t_1 \nabla t_2 + \nabla t_1 t_2$
$t_4 = \sin(t_1)$	$\nabla t_4 = \cos(t_1) \nabla t_2$
$t_5 = t_3 + t_4$	$\nabla t_5 = \nabla t_3 + \nabla t_4$
$t_6 = \cos(t_2)$	$\nabla t_6 = -\sin(t_2) \nabla t_2$
$t_7 = t_6 + t_2$	$\nabla t_7 = \nabla t_6 + \nabla t_2$
$t_8 = t_5 t_7$	$\nabla t_8 = \nabla t_5 t_7 + t_5 \nabla t_7$

Introduction: Differentiation

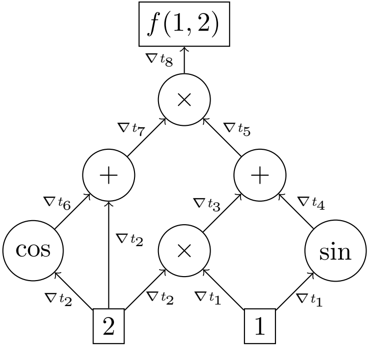
Automatic Differentiation Example



Operations	Differentiate(t_i, $\{x_1, x_2\}$)
$t_1 = x_1$	$\nabla t_1 = [1, 0]$
$t_2 = x_2$	$\nabla t_2 = [0, 1]$
$t_3 = t_1 t_2$	$\nabla t_3 = t_1 \nabla t_2 + \nabla t_1 t_2$
$t_4 = \sin(t_1)$	$\nabla t_4 = \cos(t_1) \nabla t_2$
$t_5 = t_3 + t_4$	$\nabla t_5 = \nabla t_3 + \nabla t_4$
$t_6 = \cos(t_2)$	$\nabla t_6 = -\sin(t_2) \nabla t_2$
$t_7 = t_6 + t_2$	$\nabla t_7 = \nabla t_6 + \nabla t_2$
$t_8 = t_5 t_7$	$\nabla t_8 = \nabla t_5 t_7 + t_5 \nabla t_7$

Introduction: Differentiation

Automatic Differentiation Example



Operations	eval	Differentiate(t_i, $\{x_1, x_2\}$)	∇f
$t_1 = x_1$	1	$\nabla t_1 = [1,0]$	$[1,0]$
$t_2 = x_2$	2	$\nabla t_2 = [0,1]$	$[0,1]$
$t_3 = t_1 t_2$	2	$\nabla t_3 = t_1 \nabla t_2 + \nabla t_1 t_2$	$[2,1]$
$t_4 = \sin(t_1)$	0.91	$\nabla t_4 = \cos(t_1) \nabla t_2$	$[0.54, 0]$
$t_5 = t_3 + t_4$	2.91	$\nabla t_5 = \nabla t_3 + \nabla t_4$	$[2.54, 1]$
$t_6 = \cos(t_2)$	-0.42	$\nabla t_6 = -\sin(t_2) \nabla t_2$	$[0, -0.91]$
$t_7 = t_6 + t_2$	1.58	$\nabla t_7 = \nabla t_6 + \nabla t_2$	$[0, 0.09]$
$t_8 = t_5 t_7$	4.60	$\nabla t_8 = \nabla t_5 t_7 + t_5 \nabla t_7$	$[4.08, 1.84]$

Introduction: Differentiation

Symbolic Differentiation

Differentiation Methods

- Numerical
- Automatic
- Symbolic

Basic Differentiation Rules

Chain rule:

$$\nabla \phi(u) = \dot{\phi}(u) \nabla u$$

Arithmetic operations:

$$\begin{aligned} \nabla(u \pm v) &= \nabla u \pm \nabla v \\ \nabla(uv) &= u \nabla v + v \nabla u \\ \nabla(\frac{u}{v}) &= \frac{(\nabla u - \frac{u}{v} \nabla v)}{v} \end{aligned}$$

Introduction: Differentiation

Symbolic Differentiation

Basic Differentiation Rules

Chain rule:

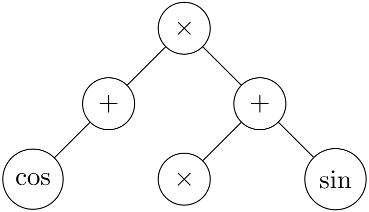
$$\nabla \phi(u) = \dot{\phi}(u) \nabla u$$

Arithmetic operations:

$$\begin{aligned} \nabla(u \pm v) &= \nabla u \pm \nabla v \\ \nabla(uv) &= u \nabla v + v \nabla u \\ \nabla\left(\frac{u}{v}\right) &= \frac{\left(\nabla u - \frac{u}{v} \nabla v\right)}{v} \end{aligned}$$

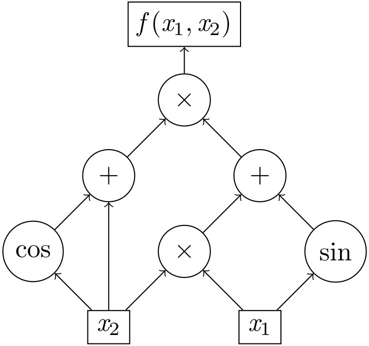
Example

$$y = f(x_1, x_2) = (x_1 x_2 + \sin(x_1))(x_2 + \cos(x_2))$$



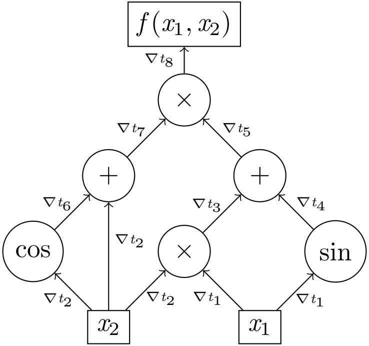
Introduction: Differentiation

Symbolic Differentiation Example



Introduction: Differentiation

Symbolic Differentiation Example



Operations	Differentiate$(t_i, \{x_1, x_2\})$
$t_1 = x_1$	$\nabla t_1 = [1, 0]$
$t_2 = x_2$	$\nabla t_2 = [0, 1]$
$t_3 = t_1 t_2$	$\nabla t_3 = t_1 \nabla t_2 + \nabla t_1 t_2$
$t_4 = \sin(t_1)$	$\nabla t_4 = \cos(t_1) \nabla t_2$
$t_5 = t_3 + t_4$	$\nabla t_5 = \nabla t_3 + \nabla t_4$
$t_6 = \cos(t_2)$	$\nabla t_6 = -\sin(t_2) \nabla t_2$
$t_7 = t_6 + t_2$	$\nabla t_7 = \nabla t_6 + \nabla t_2$
$t_8 = t_5 t_7$	$\nabla t_8 = \nabla t_5 t_7 + t_5 \nabla t_7$

Introduction: Differentiation

Symbolic Differentiation Example

Operations	Differentiate(t_i, $\{x_1, x_2\}$)
$t_1 = x_1$	$\nabla t_1 = [1, 0]$
$t_2 = x_2$	$\nabla t_2 = [0, 1]$
$t_3 = t_1 t_2$	$\nabla t_3 = t_1 \nabla t_2 + \nabla t_1 t_2$
$t_4 = \sin(t_1)$	$\nabla t_4 = \cos(t_1) \nabla t_2$
$t_5 = t_3 + t_4$	$\nabla t_5 = \nabla t_3 + \nabla t_4$
$t_6 = \cos(t_2)$	$\nabla t_6 = -\sin(t_2) \nabla t_2$
$t_7 = t_6 + t_2$	$\nabla t_7 = \nabla t_6 + \nabla t_2$
$t_8 = t_5 t_7$	$\nabla t_8 = \nabla t_5 t_7 + t_5 \nabla t_7$

$$\frac{\partial f(x1, x2)}{x_1} = (x_2 + \cos(x_1))(x_2 + \cos(x_2))$$

$$\frac{\partial f(x1, x2)}{x_2} = (x_1 (x_2 + \cos(x_2)) + (x_1 x_2 + \sin(x_1))(1 - sin(x_2))$$

Differentiate a Modelica Model

Modelica language features

State-Space equations

$$\begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Language Elements

- Equations
- Algorithms
- Functions

Differentiate a Modelica Model

Modelica language features

State-Space equations

$$\begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Language Elements

- Equations
- Algorithms
- Functions

Equations

Using symbolic differentiaten.
$\Rightarrow$ Straight forward!

Differentiate a Modelica Model

Modelica language features

State-Space equations

$$\begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Language Elements

- Equations
- Algorithms
- Functions

Differentiate a Modelica Model

Modelica language features

State-Space equations

$$\begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Language Elements

- Equations
- Algorithms
- Functions

Algorithms

Using a combination of automatic differentiation and symbolic differentiation.

Example

```
algorithm
  y := 0.0;
  fac := 1.0;
  for i in 0:n loop
    j := Real(i);
    if j > 0.0 then
      fac := fac * j;
    end if;
    y := y + x ^ j / fac;
  end for;
equation
 der(z) = y;
```

Differentiate a Modelica Model

Modelica language features

Differentiated algorithm

```
algorithm
  y$pDERz := 0.0;
  y := 0.0;
  fac$pDERz := 0.0;
  fac := 1.0;
  for i in 0:n loop
    j$pDERz := 0.0;
    j := /*REAL*/(i);
    if j > 0.0 then
      fac$pDERz := fac$pDERz * j +
        fac * j$pDERz;
      fac := fac * j;
    end if;
    y$pDERz := y$pDERz + (j$pDERz *
    log(x) * x ^ j * fac - x ^ j *
     fac$pDERz) * fac ^ -2.0;
    y := y + x ^ j / fac;
  end for;
```

Algorithms

Using a combination of automatic differentiation and symbolic differentiation.

Example

```
algorithm
  y := 0.0;
  fac := 1.0;
  for i in 0:n loop
    j := Real(i);
    if j > 0.0 then
      fac := fac * j;
    end if;
    y := y + x ^ j / fac;
  end for;
equation
 der(z) = y;
```

Differentiate a Modelica Model

Functions

State-Space equations

$$\begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Language Elements

- Equations
- Algorithms
- Functions

Differentiate a Modelica Model

Functions

State-Space equations

$$\begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Language Elements

- Equations
- Algorithms
- Functions

Functions with derivative annotation

- A function can have an annotation `derivative`.

derivative annotation

```
function f
  annotation(derivative=df);
  input Real x;
  output Real y;
algorithm
  y := cos(x);
end f;

function df
  input Real x;
  input Real dx;
  output Real dy;
algorithm
  dy := -sin(x)*dx;
end df;
```

Differentiate a Modelica Model

Functions

State-Space equations

$$\begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Language Elements

- Equations
- Algorithms
- Functions

Differentiate a Modelica Model

Functions

State-Space equations

$$\begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} h(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ k(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{pmatrix}$$

Language Elements

- Equations
- Algorithms
- Functions

Example

```
function f1
 input Real a;
 output Real b;
 external b = myfoo(a)
 annotation(Library="foo.o",
  Include="#include \"myfoo.h\"");
end f1;
```

```
if (a > tol || a < -tol) then
 delta = a*tol;
else
 delta = tol;
$DERf1$pDERa =
(f1(a+delta) - f(a))/delta;
```

Differentiate a Modelica Model

Special issue

Non-linear

- equations
- algebraic loops

Differentiate a Modelica Model

Special issue

Non-linear

- equations
- algebraic loops

Example

$$\underline{f}(\underline{x}, \underline{p}, t) := \begin{pmatrix} \underline{\dot{x}_1} \\ \underline{\dot{x}_2} \end{pmatrix} = \begin{pmatrix} ax_1 + \frac{1}{2}\underline{\dot{x}_2}^2 \\ bx_2 - \frac{1}{2}\underline{\dot{x}_1}^2 \end{pmatrix}$$

Differentiate a Modelica Model

Special issue

Non-linear

- equations
- algebraic loops

Example

$$\underline{f}(\underline{x}, \underline{p}, t) := \begin{pmatrix} \dot{x}_1 \\ \underline{\dot{x}_2} \end{pmatrix} = \begin{pmatrix} ax_1 + \frac{1}{2}\dot{x}_2^{\,2} \\ bx_2 - \frac{1}{2}\underline{\dot{x}_1}^{\,2} \end{pmatrix}$$

differentiate with respect to x_1

$$\Downarrow$$

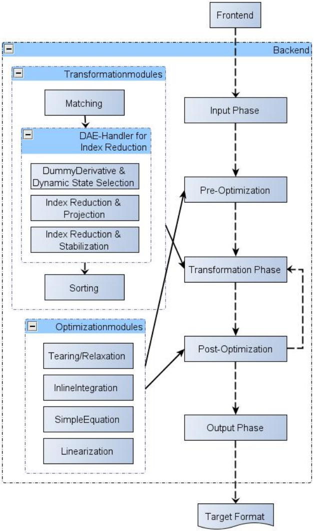
$$\begin{pmatrix} \frac{\partial f_1}{\partial x_1} := \frac{\partial \dot{x}_1}{\partial x_1} = a + \frac{\partial \dot{x}_2}{\partial x_1}\dot{x}_2 \\ \frac{\partial f_2}{\partial x_1} := \frac{\partial \dot{x}_2}{\partial x_1} = -\frac{\partial \dot{x}_1}{\partial x_1}\dot{x}_1 \end{pmatrix}$$

$\Rightarrow$ Nonlinear equations and algebraic loops that are differentiated, result in equations depending linearly on the differentiated variables.

Differentiate a Modelica Model

Implementation

- Generates a new BackendDAE System.
- Use the power of OpenModelica.
- Linearization is an Optimization Backend Module.



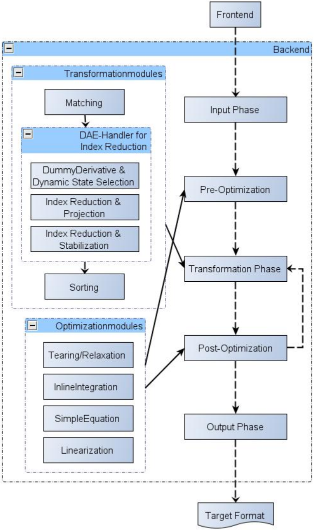
Differentiate a Modelica Model

Implementation

- Generates a new BackendDAE System.
- Use the power of OpenModelica.
- Linearization is an Optimization Backend Module.

experemental status

- omc usage with debug flag +d=linearization
- linear model is created with -l <time>



Application for Linear Model

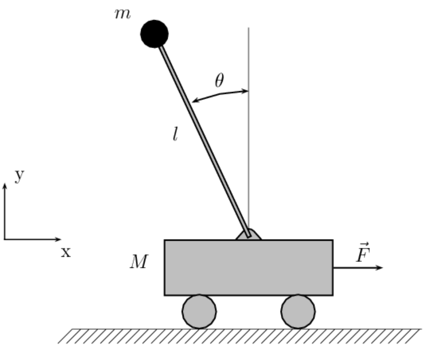
Controlling an inverse Pendulum by a linear model

```
model InversePendulum
 parameter Real M = 0.5;
 parameter Real m = 0.2;
 parameter Real b = 0.1;
 parameter Real i = 0.006;
 parameter Real g = 9.8;
 parameter Real l = 0.3;
 parameter Real pi = 3.14;
 Real c_x, c_v;
 Real p_theta, p_w;
 input Real u;
equation
 der(c_x) = c_v;
 der(p_theta) = p_w;

 (M + m)*der(c_v) + b*c_v + u =
 m*l*der(p_w)*cos(p_theta+pi)
 -m*l*p_w^2*sin(p_theta+pi);

 (i+m*l^2)*der(p_w) +
 m*l*g*sin(p_theta+pi)=
 -m*l*der(c_v)*cos(p_theta+pi);

end InversePendulum;
```



Application for Linear Model

Controlling an inverse Pendulum by a linear model

```
model InversePendulum
 parameter Real M = 0.5;
 parameter Real m = 0.2;
 parameter Real b = 0.1;
 parameter Real i = 0.006;
 parameter Real g = 9.8;
 parameter Real l = 0.3;
 parameter Real pi = 3.14;
 Real c_x, c_v;
 Real p_theta, p_w;
 input Real u;
equation
 der(c_x) = c_v;
 der(p_theta) = p_w;

 (M + m)*der(c_v) + b*c_v + u =
 m*l*der(p_w)*cos(p_theta+pi)
 -m*l*p_w^2*sin(p_theta+pi);

 (i+m*l^2)*der(p_w) +
 m*l*g*sin(p_theta+pi)=
 -m*l*der(c_v)*cos(p_theta+pi);

end InversePendulum;
```

```
Equations (16)
=========
1 : $DER$Pc_x$pDERc_x = 0.0
2 : $DER$Pc_x$pDERc_v = 1.0
3 : $DER$Pc_x$pDERp_theta = 0.0
4 : $DER$Pc_x$pDERp_w = 0.0
5 : $DER$Pp_theta$pDERc_x = 0.0
6 : $DER$Pp_theta$pDERc_v = 0.0
7 : $DER$Pp_theta$pDERp_theta = 0.0
8 : $DER$Pp_theta$pDERp_w = 1.0
9 : (M + m) * $DER$Pc_v$pDERc_x +
m * (l * ($DER$Pp_w$pDERc_x * cos(p_theta + pi))) = 0.0
10 : (M + m) * $DER$Pc_v$pDERc_v +
(b + m * (l * ($DER$Pp_w$pDERc_v * cos(p_theta + pi)))) = 0.0
11 : (M + m) * $DER$Pc_v$pDERp_theta +
m * (l * ($DER$Pp_w$pDERp_theta * cos(p_theta +
pi) + (-der(p_w)) * sin(p_theta + pi))) -
m * (l * (p_w ^ 2.0 * cos(p_theta + pi))) = 0.0
12 : (M + m) * $DER$Pc_v$pDERp_w +
m * (l * ($DER$Pp_w$pDERp_w * cos(p_theta + pi))) -
2.0 * (m * (l * (p_w * sin(p_theta + pi)))) = 0.0
13 : (i + m * l ^ 2.0) * $DER$Pp_w$pDERc_x =
(-m) * (l * ($DER$Pc_v$pDERc_x * cos(p_theta + pi)))
14 : (i + m * l ^ 2.0) * $DER$Pp_w$pDERc_v =
(-m) * (l * ($DER$Pc_v$pDERc_v * cos(p_theta + pi)))
15 : (i + m * l ^ 2.0) * $DER$Pp_w$pDERp_theta +
m * (l * (g * cos(p_theta + pi))) =
(-m) * (l * ($DER$Pc_v$pDERp_theta * cos(p_theta + pi)
+ (-der(c_v)) * sin(p_theta + pi)))
16 : (i + m * l ^ 2.0) * $DER$Pp_w$pDERp_w =
(-m) * (l * ($DER$Pc_v$pDERp_w * cos(p_theta + pi)))
```

Application for Linear Model

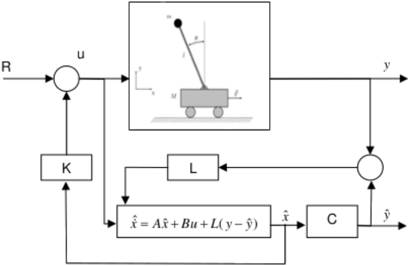
Controlling an inverse Pendulum by a linear model

```
model linear_InversePendulum
  [...]
  parameter Real x0[4] = {0,0,0,0};
  parameter Real u0[1] = {0};
  parameter Real A[4,4] = [0,1,0,0;
      0,-0.18,2.672,0;
      0,0,0,1;
      0,-0.45,31.18,0];
  parameter Real B[4,1] = [0;1.81;
    0;4.54];
  parameter Real C[2,4] = [1,0,0,0;
      0,0,1,0];
  parameter Real D[2,1] = [0;0];
  Real x[4](start=x0);
  output Real y[2];
  input Real u[1](start=u0);
  [...]
equation
  der(x) = A * x + B * u;
  y = C * x + D * u;
end linear_InversePendulum;
```

Application for Linear Model

Controlling an inverse Pendulum by a linear model

```
model linear_testInversePendulum
  [...]
  parameter Real x0[4] = {0,0,0,0};
  parameter Real u0[1] = {0};
  parameter Real A[4,4] =
   [0,1,0,0;  -0.18,2.672,0;
    0,0,0,1;  0,-0.45,31.18,0];
  parameter Real B[4,1]=[0;1.818;0;4.54];
  parameter Real C[2,4]=[1,0,0,0;0,0,1,0];
  parameter Real D[2,1] = [0;0];
  parameter Real L[4,2] = 1.0e+03 *
   [ 0.0826,  -0.0010; 1.6992,  -0.0402;
   -0.0014, 0.0832;  -0.0762,  1.7604];
  parameter Real K[1,4]=[-70.7107,
-37.8345,  105.5298,   20.9238];
  Real x[4](start=x0);
  Real y[2];
  output Real kx = scalar(K*x);
  input Real u[1](start=u0);
  input Real y_nonlinear[2];
  [...]
equation
  der(x) = A * x + B * u + L*(y_nonlinear - y);
  y = C * x + D * u;
end linear_testInversePendulum;
```



Applications for Symbolic Jacobian

Provide the analytical jacobian matrix to DASSL

Numerical Integration with DASSL

$$0 = f(t_{n+1}, x_{n+1}, \alpha x_{n+1} + \beta)$$

Applications for Symbolic Jacobian

Provide the analytical jacobian matrix to DASSL

Numerical Integration with DASSL

$$0 = f(t_{n+1}, x_{n+1}, \alpha x_{n+1} + \beta)$$

Solving this with a modified Newton iteration

$$x^{m+1} = y^m - c(\frac{\partial \underline{h}}{\partial \underline{x}} + cj * \frac{\partial \underline{h}}{\partial \underline{\dot{x}}})^{-1} h(t, x, \hat{\alpha} x + \beta).$$

The iteration matrix

$$M = \frac{\partial \underline{h}}{\partial \underline{x}} + cj * \frac{\partial \underline{h}}{\partial \underline{\dot{x}}}$$

is numerically determined by DASSL.

Applications for Symbolic Jacobian

Provide the analytical jacobian matrix to DASSL

Numerical Integration with DASSL

$$0 = f(t_{n+1}, x_{n+1}, \alpha x_{n+1} + \beta)$$

Solving this with a modified Newton iteration

$$x^{m+1} = y^m - c\left(\frac{\partial \underline{h}}{\partial \underline{x}} + cj * \frac{\partial \underline{h}}{\partial \underline{\dot{x}}}\right)^{-1} h(t, x, \hat{\alpha} x + \beta).$$

The iteration matrix

$$M = \frac{\partial \underline{h}}{\partial \underline{x}} + cj * \frac{\partial \underline{h}}{\partial \underline{\dot{x}}}$$

is numerically determined by DASSL.

Providing symbolical iteration matrix

$$M = A - cj * Id$$

Summary

- The OMC can automatically generate symbolic derivatives for Linearization.
- This offers a variety of different application fields.

Outlook

- The performance of the current implementation can be improved:
  - compile time.
  - evaluating the Jacobians.
- In the future it is possible to improve this module in two directions:
  - The user could select some functions and the variables.
  - Generate directly a Modelica model with the symbolic derivative expressions.

Outlook

- The performance of the current implementation can be improved:
  - compile time.
  - evaluating the Jacobians.
- In the future it is possible to improve this module in two directions:
  - The user could select some functions and the variables.
  - Generate directly a Modelica model with the symbolic derivative expressions.

Further Applications

- Optimization
- Parameter identification
- Sensitivity analysis
- Uncertainty calculation