



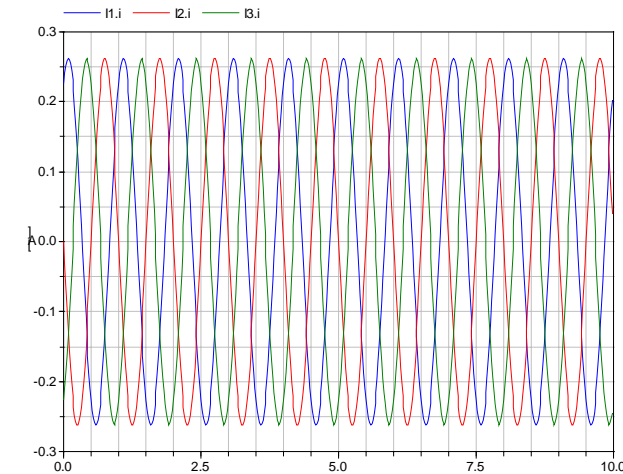
# Mathematical Aspects of Object-Oriented Modeling and Simulation

*4<sup>th</sup> International Modelica Conference*  
*Hamburg-Harburg*

Bernhard Bachmann  
University of Applied Sciences  
Bielefeld

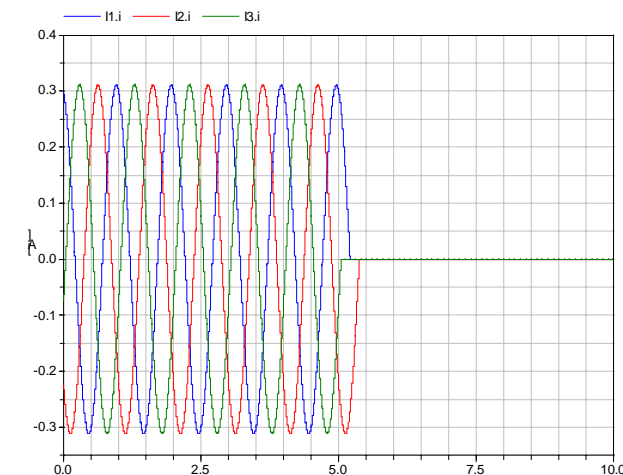
## ■ Continuous system simulation

- Introductory example
- Symbolic transformation
- Efficiency issues and non-linearities
- Higher index problems
- Initialization
- Numerical issues



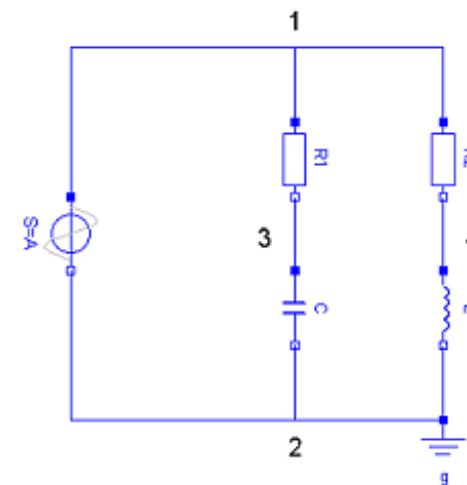
## ■ Mixed system simulation

- Event handling
- State events  $\Leftrightarrow$  time events
- Symbolic transformation
- Ideal components
- Varying higher index problems
- Efficiency and numerical issues

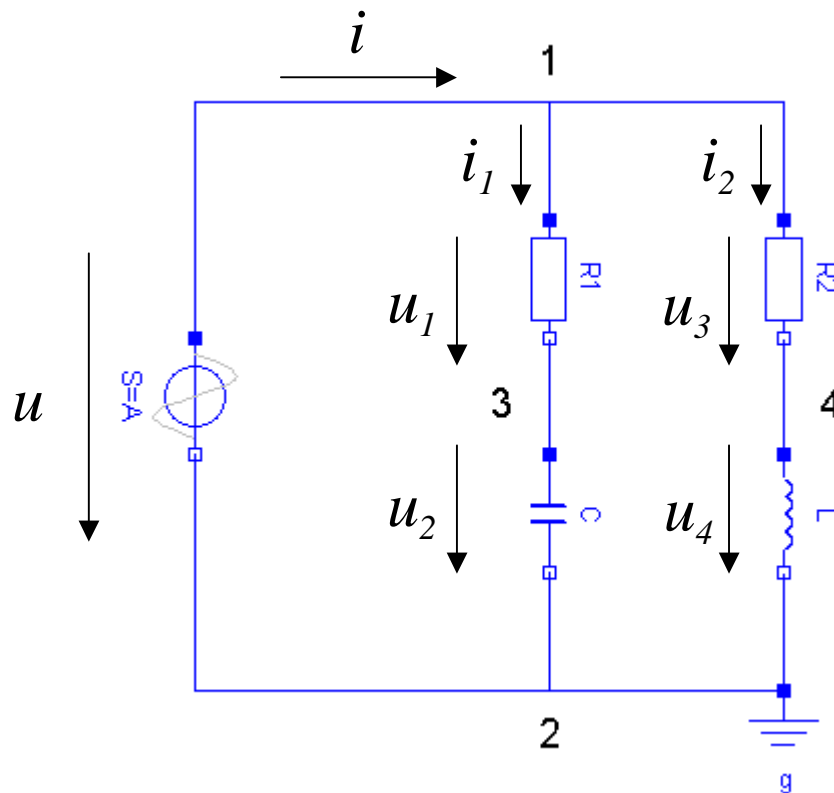


# Introductory Example

- Flat model description
  - Differential equations, algebraic equations
  - Parameters, constants, and continuous variables
- Mathematical formalism
  - Differential-algebraic equations (DAEs)
  - Principles of numerical solution methods
  - Same number of equations and unknowns
  - Basic transformation steps
  - Explicit state-space representation
- Hierarchical modeling
  - Local description by equations
  - Connection equations describe interaction
  - Elimination of trivial equations
  - Explicit state-space representation



# Example: A Simple Electrical System Flat Representation



$$u = A \cdot \sin(\omega t)$$

$$u = u_1 + u_2, \quad u = u_3 + u_4$$

$$i = i_1 + i_2$$

$$u_1 = R_1 i_1, \quad u_3 = R_2 i_2,$$

$$C \frac{du_2}{dt} = i_1, \quad L \frac{di_2}{dt} = u_4$$

The dynamical behavior of the system is given by a

**System of differential-algebraic equations (DAEs)**

# Mathematical Formalism

General representation of DAEs:

$$\underline{0} = \underline{f} \left( t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p} \right)$$

$t$	time
$\underline{\dot{x}}(t)$	vector of differentiated state variables
$\underline{x}(t)$	vector of state variables
$\underline{y}(t)$	vector of algebraic variables
$\underline{u}(t)$	vector of input variables
$\underline{p}$	vector of parameters and/or constants

# Example: A Simple Electrical System

## Model of the electrical system:

$$u = A \cdot \sin(\omega t + \varphi)$$

$$u = u_1 + u_2, \quad u = u_3 + u_4$$

$$i = i_1 + i_2$$

$$u_1 = R_1 i_1, \quad u_3 = R_2 i_2$$

$$C \dot{u}_2 = i_1, \quad L \dot{i}_2 = u_4$$

- 8 equations!
- Number of unknowns?
- Solution of the system?

## General representation:

$$\underline{0} = \underline{f}\left(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p}\right)$$

$$t \text{ time, } \underline{\dot{x}}(t) = \begin{pmatrix} \dot{u}_2 \\ \dot{i}_2 \end{pmatrix}, \quad \underline{x}(t) = \begin{pmatrix} u_2 \\ i_2 \end{pmatrix},$$

$$\underline{y}(t) = \begin{pmatrix} u \\ u_1 \\ u_3 \\ u_4 \\ i \\ i_1 \end{pmatrix}, \quad \underline{u}(t) \text{ not present, } \underline{p} = \begin{pmatrix} A \\ \omega \\ R_1 \\ R_2 \\ C \\ L \end{pmatrix}$$

# Understand Numerical Integration Methods (Explicit Euler Method)

Integration of explicit ordinary differential equations (ODEs):

$$\dot{\underline{x}}(t) = \underline{f}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right), \quad \underline{x}(t_0) = \underline{x}_0$$

Numerical approximation of the derivative  
and/or right-hand-side:

$$\dot{\underline{x}}(t_n) \approx \frac{\underline{x}(t_{n+1}) - \underline{x}(t_n)}{t_{n+1} - t_n} \approx \underline{f}\left(t_n, \underline{x}(t_n), \underline{u}(t_n), \underline{p}\right)$$

Iteration scheme:

$$\underline{x}(t_{n+1}) \approx \underline{x}(t_n) + (t_{n+1} - t_n) \cdot \underline{f}\left(t_n, \underline{x}(t_n), \underline{u}(t_n), \underline{p}\right)$$

Calculating an  
approximation of  
 $\underline{x}(t_{n+1})$  based on the  
values of  $\underline{x}(t_n)$

Here:

Explicit Euler  
integration method

Convergence?

# Basic Transformation Steps

Transformation to explicit state-space representation:

$$\begin{array}{ccc}
 \underline{0} = \underline{f}\left(t, \dot{\underline{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p}\right) & & \underline{z}(t) = \begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \underline{g}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right) \\
 \downarrow & \nearrow & \downarrow \\
 \underline{0} = \underline{f}\left(t, \underline{z}(t), \underline{x}(t), \underline{u}(t), \underline{p}\right), \quad \underline{z}(t) = \begin{pmatrix} \dot{\underline{x}}(t) \\ \underline{y}(t) \end{pmatrix} & & \begin{aligned} \dot{\underline{x}}(t) &= \underline{h}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right) \\ \underline{y}(t) &= \underline{k}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right) \end{aligned}
 \end{array}$$

Implicit function theorem:

Necessary condition for the existence of the transformation is that the following matrix is regular at the point of interest:

$$\det\left(\frac{\partial}{\partial \underline{z}} \underline{f}\left(t, \underline{z}(t), \underline{x}(t), \underline{u}(t), \underline{p}\right)\right) \neq 0$$

# Example: A Simple Electrical System

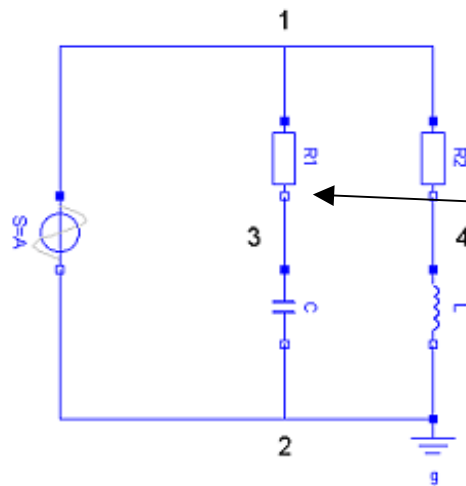
Model of the electrical system:

$$u = A \cdot \sin(\omega t)$$

$$u = u_1 + u_2, \quad u = u_3 + u_4, \quad i = i_1 + i_2$$

$$u_1 = R_1 i_1, \quad u_3 = R_2 i_2$$

$$C \dot{u}_2 = i_1, \quad L \dot{i}_2 = u_4$$



Transformation to explicit state space representation:

$$u = A \cdot \sin(\omega t)$$

$$u_3 = R_2 i_2$$

$$u_1 = u - u_2$$

$$u_4 = u - u_3$$

$$i_1 = u_1 / R_1$$

$$i = i_1 + i_2$$

$$\dot{u}_2 = i_1 / C$$

$$\dot{i}_2 = u_4 / L$$

Causality!

# Transformed Simulation Model

Model of the electrical system in explicit state space representation:

$$\begin{aligned}
 u &= A \cdot \sin(\omega t) \\
 u_3 &= R_2 i_2 \\
 u_1 &= u - u_2 \\
 u_4 &= u - u_3 \\
 i_1 &= u_1 / R_1 \\
 i &= i_1 + i_2 \\
 \hline
 \dot{u}_2 &= i_1 / C \\
 \dot{i}_2 &= u_4 / L
 \end{aligned}$$

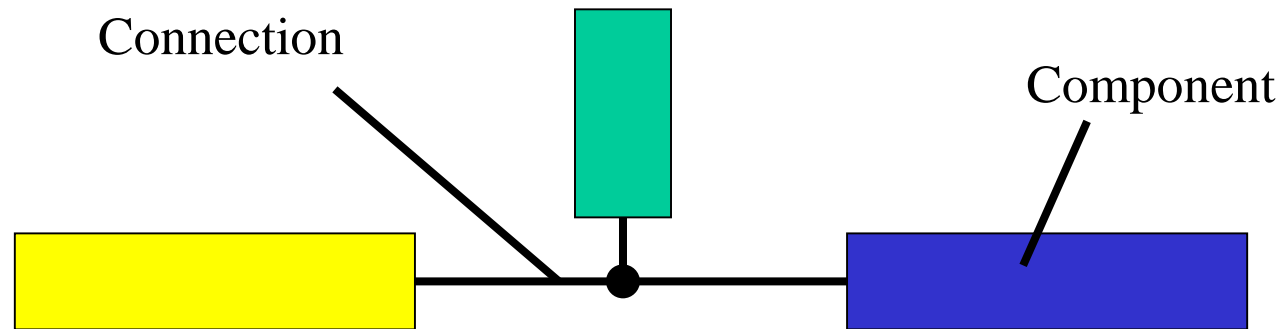
General representation:

$$\dot{\underline{x}}(t) = \underline{h}(t, \underline{x}(t), \underline{u}(t), \underline{p})$$

$$\underline{y}(t) = \underline{k}(t, \underline{x}(t), \underline{u}(t), \underline{p})$$

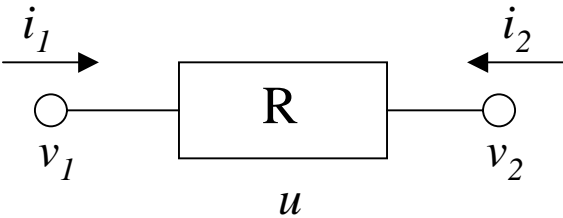
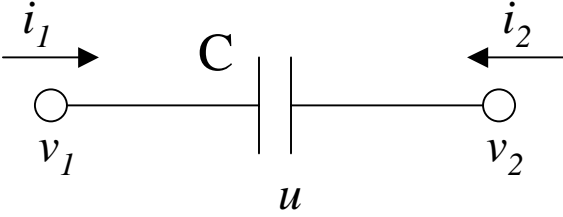
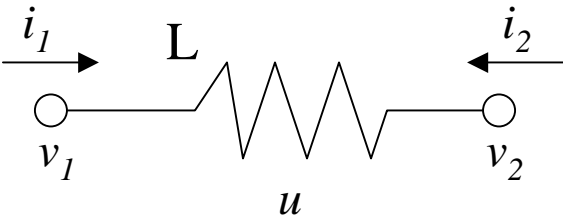
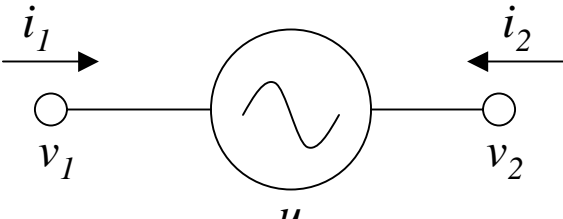
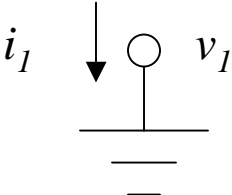
$$\begin{aligned}
 \begin{pmatrix} \dot{u}_2(t) \\ \dot{i}_2(t) \end{pmatrix} &= \underline{h} \left( t, \underbrace{u_2(t), i_2(t)}_{\text{states}}, \underbrace{A, \omega, R_1, R_2, C, L}_{\text{parameters/constants}} \right) \\
 \begin{pmatrix} u(t) \\ u_1(t) \\ u_3(t) \\ u_4(t) \\ i(t) \\ i_1(t) \end{pmatrix} &= \underline{k} \left( t, \underbrace{u_2(t), i_2(t)}_{\text{states}}, \underbrace{A, \omega, R_1, R_2, C, L}_{\text{parameters/constants}} \right)
 \end{aligned}$$

# Hierarchical Object-Oriented Modeling



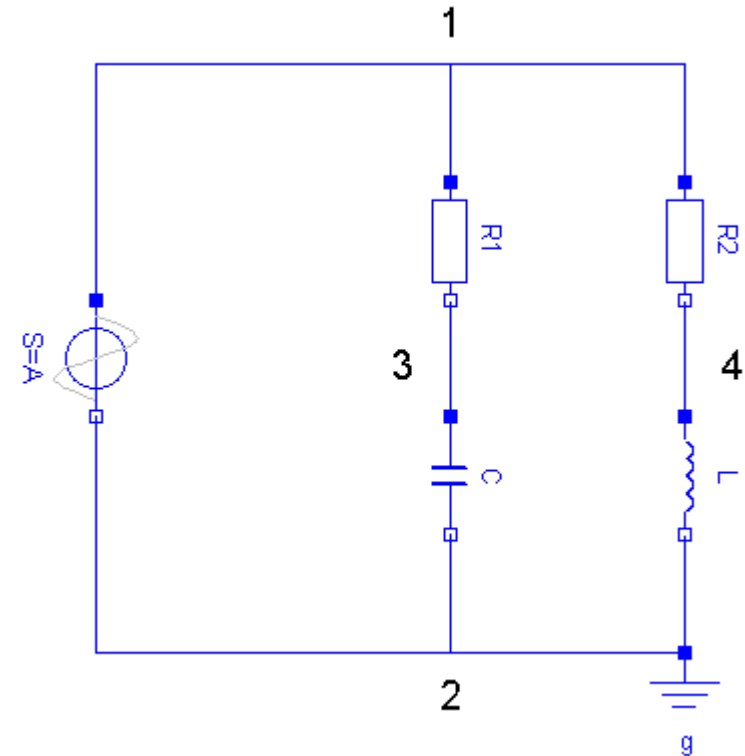
- Each icon represents a physical component
  - Electrical resistor, mechanical gearbox, pump
- Composition lines are the actual physical connections
  - Electrical line, mechanical connection, heat flow between two components
- Variables at the interfaces describe interaction with other components
- Physical behavior of a component is described by equations
- Hierarchical decomposition of components

# Hierarchical Model of the Electrical System

<b>Resistor</b>		$0 = i_1 + i_2$ $u = v_1 - v_2$ $u = R i_1$
<b>Capacitor</b>		$0 = i_1 + i_2$ $u = v_1 - v_2$ $i_1 = C du/dt$
<b>Inductor</b>		$0 = i_1 + i_2$ $u = v_1 - v_2$ $u = L di_1/dt$
<b>Voltage source</b>		$0 = i_1 + i_2$ $u = v_1 - v_2$ $u = A \sin(wt)$
<b>Ground</b>		$v_1 = 0$

# Hierarchical Model of the Electrical System

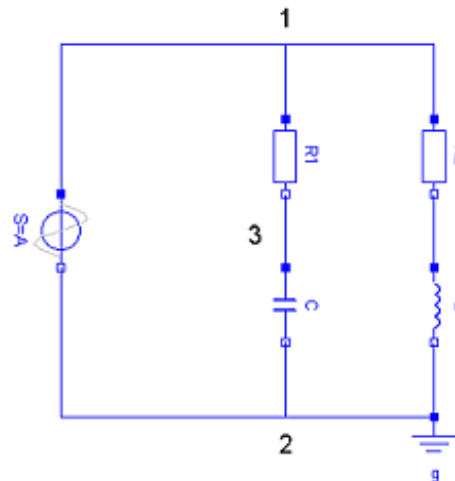
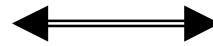
$0 = R1.i_1 + R1.i_2$ $R1.u = R1.v_1 - R1.v_2$ $R1.u = R1.R \cdot R1.i_1$	<b>R1</b>	$0 = R2.i_1 + R2.i_2$ $R2.u = R2.v_1 - R2.v_2$ $R2.u = R2.R \cdot R2.i_1$	<b>R2</b>
$0 = C.i_1 + C.i_2$ $C.u = C.v_1 - C.v_2$ $C.i_1 = C.C \cdot dC.u/dt$	<b>C</b>	$0 = L.i_1 + L.i_2$ $L.u = L.v_1 - L.v_2$ $L.u = L.L \cdot dL.i_1/dt$	<b>L</b>
$0 = S.i_1 + S.i_2$ $S.u = S.v_1 - S.v_2$ $S.u = S.A \sin(S.w \cdot t)$	<b>S</b>	$g.v_1 = 0$	<b>g</b>
$S.v_1 = R1.v_1$ $S.v_1 = R2.v_1$ $0 = S.i_1 + R1.i_1 + R2.i_1$	<b>1</b>	$R1.v_2 = C.v_1$ $0 = R1.i_2 + C.i_1$	<b>3</b>
$g.v_1 = C.v_2$ $g.v_1 = L.v_2$ $g.v_1 = S.v_2$ $0 = g.i_1 + C.i_2 + L.i_2 + S.i_2$	<b>2</b>	$R2.v_2 = L.v_1$ $0 = R2.i_2 + L.i_1$	<b>4</b>



- 27 equations!
- number of unknowns?
- many trivial equations!

Transformation to explicit state space representation with elimination of trivial equations and further simplifications

$$\begin{aligned}
 R1.v_1 &:= S.A \cdot \sin(S.w \cdot t) \\
 R2.u &:= R2.R \cdot L.i_1 \\
 R1.u &:= R1.v_1 - C.u \\
 L.u &:= R1.v_1 - R2.u \\
 C.i_1 &:= R1.u / R1.R \\
 S.i_1 &:= C.i_1 + L.i_1 \\
 dC.u / dt &:= C.i_1 / C.C \\
 dL.i_1 / dt &:= L.u / L.L
 \end{aligned}$$



$$\begin{aligned}
 u &= A \cdot \sin(\omega t) \\
 u_3 &= R_2 i_2 \\
 u_1 &= u - u_2 \\
 u_4 &= u - u_3 \\
 i_1 &= u_1 / R_1 \\
 i &= i_1 + i_2 \\
 \dot{u}_2 &= i_1 / C \\
 \dot{i}_2 &= u_4 / L
 \end{aligned}$$

# Symbolic Transformation

## Algorithmic Steps

- DAEs and bipartite graph representation

- Structural representation of the equation system

$$\underline{0} = \underline{f}(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p})$$



- The matching problem

- Assign to each variable exact one equation
- Same number of equations and unknowns

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



- Construct a directed graph

- Find sinks, sources and strong components
- Sorting the equation system

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



- Adjacency Matrix and structural regularity

- Block-lower triangular form (BLT-Transformation)

$$\underline{\dot{x}}(t) = \underline{h}(t, \underline{x}(t), \underline{u}(t), \underline{p})$$

$$\underline{y}(t) = \underline{k}(t, \underline{x}(t), \underline{u}(t), \underline{p})$$

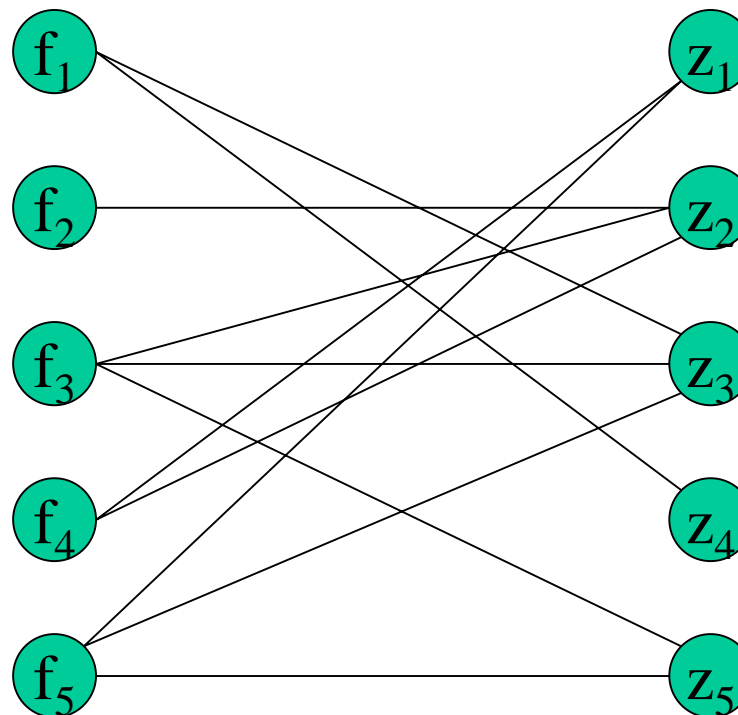
# DAEs and Bipartite Graph Representation

**Example of a regular DAE:**

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

$$\begin{aligned} f_1(z_3, z_4) &= 0 \\ f_2(z_2) &= 0 \\ f_3(z_2, z_3, z_5) &= 0 \\ f_4(z_1, z_2) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \end{aligned}$$

**Bipartite graph**



**Adjacency matrix**

$$\begin{matrix} & z_1 & z_2 & z_3 & z_4 & z_5 \\ \begin{matrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{matrix} & \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{matrix}$$

# Solve the Matching Problem

Example of a regular DAE:

$$f_1(z_3, z_4) = 0$$

$$f_2(z_2) = 0$$

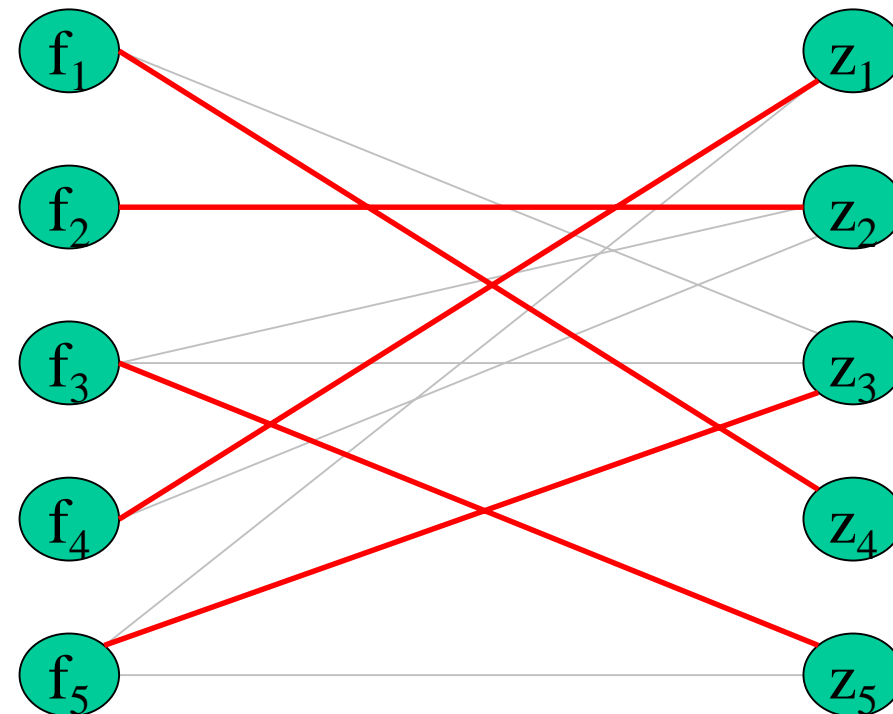
$$f_3(z_2, z_3, z_5) = 0$$

$$f_4(z_1, z_2) = 0$$

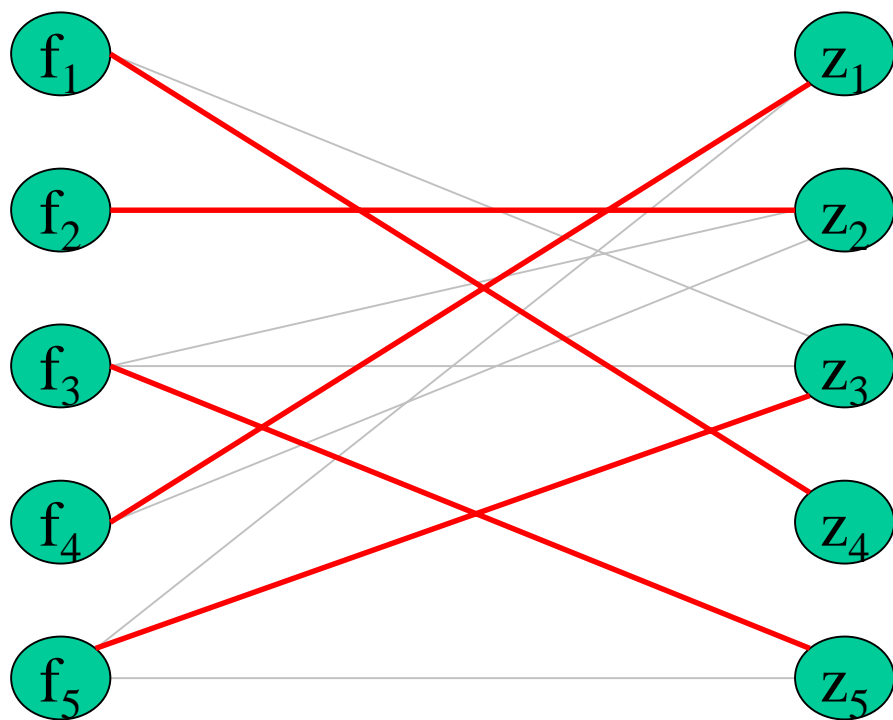
$$f_5(z_1, z_3, z_5) = 0$$

$$\begin{array}{c} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{array} \begin{pmatrix} z_1 & z_2 & z_3 & z_4 & z_5 \\ 0 & 0 & 1 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & \mathbf{1} \\ \mathbf{1} & 1 & 0 & 0 & 0 \\ 1 & 0 & \mathbf{1} & 0 & 1 \end{pmatrix}$$

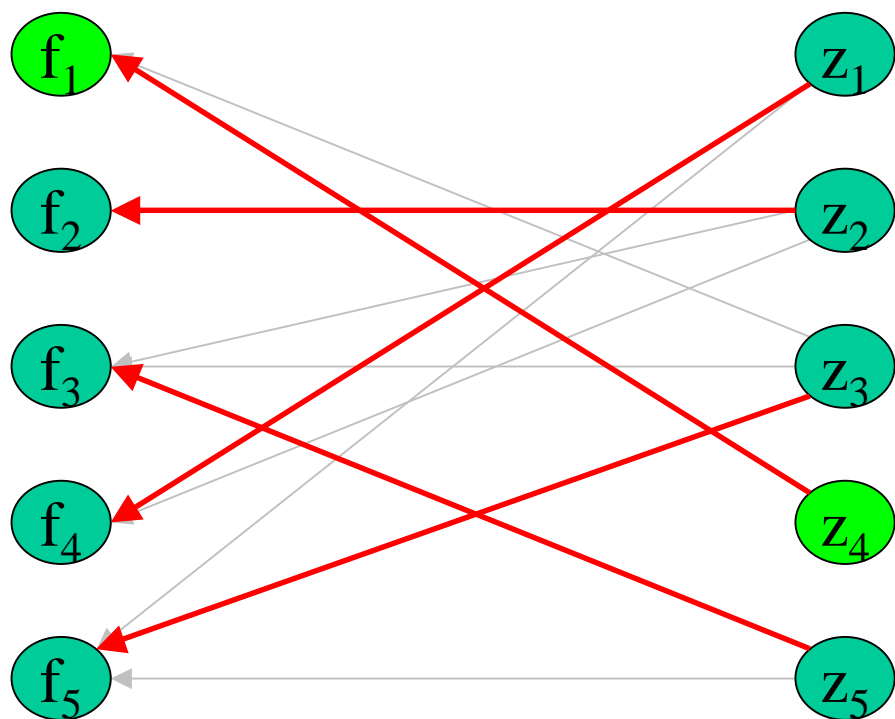
Bipartite graph



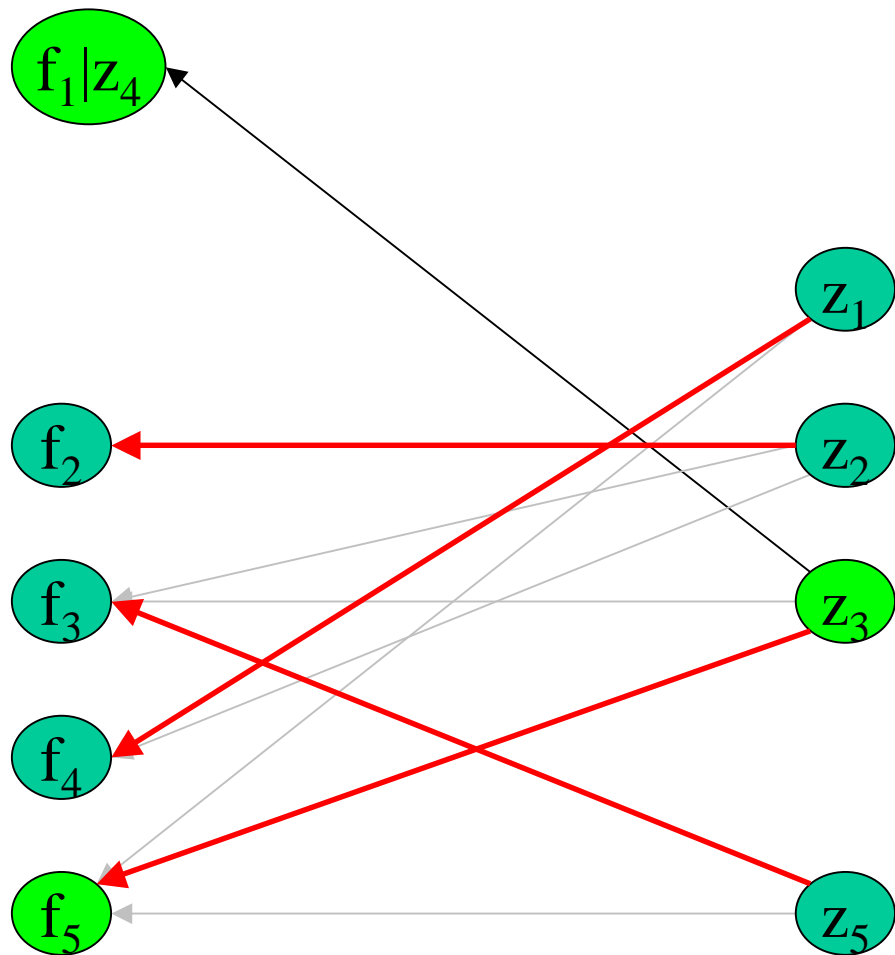
# Construct a Directed Graph



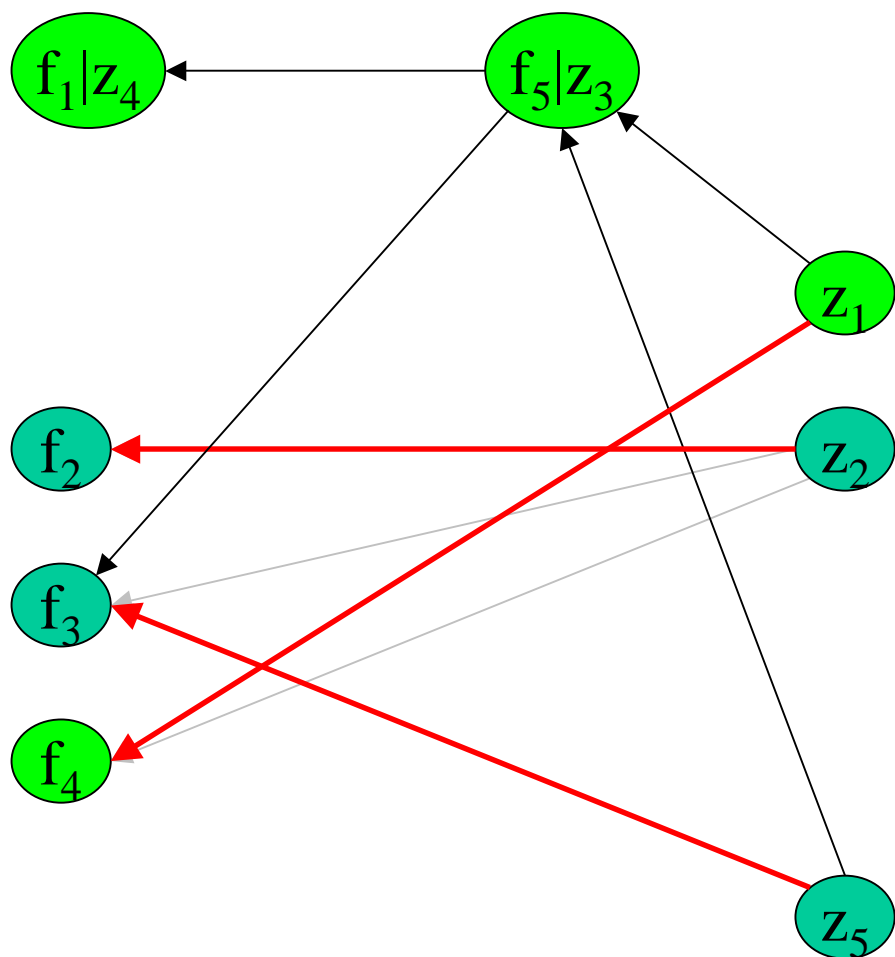
# Construct a Directed Graph



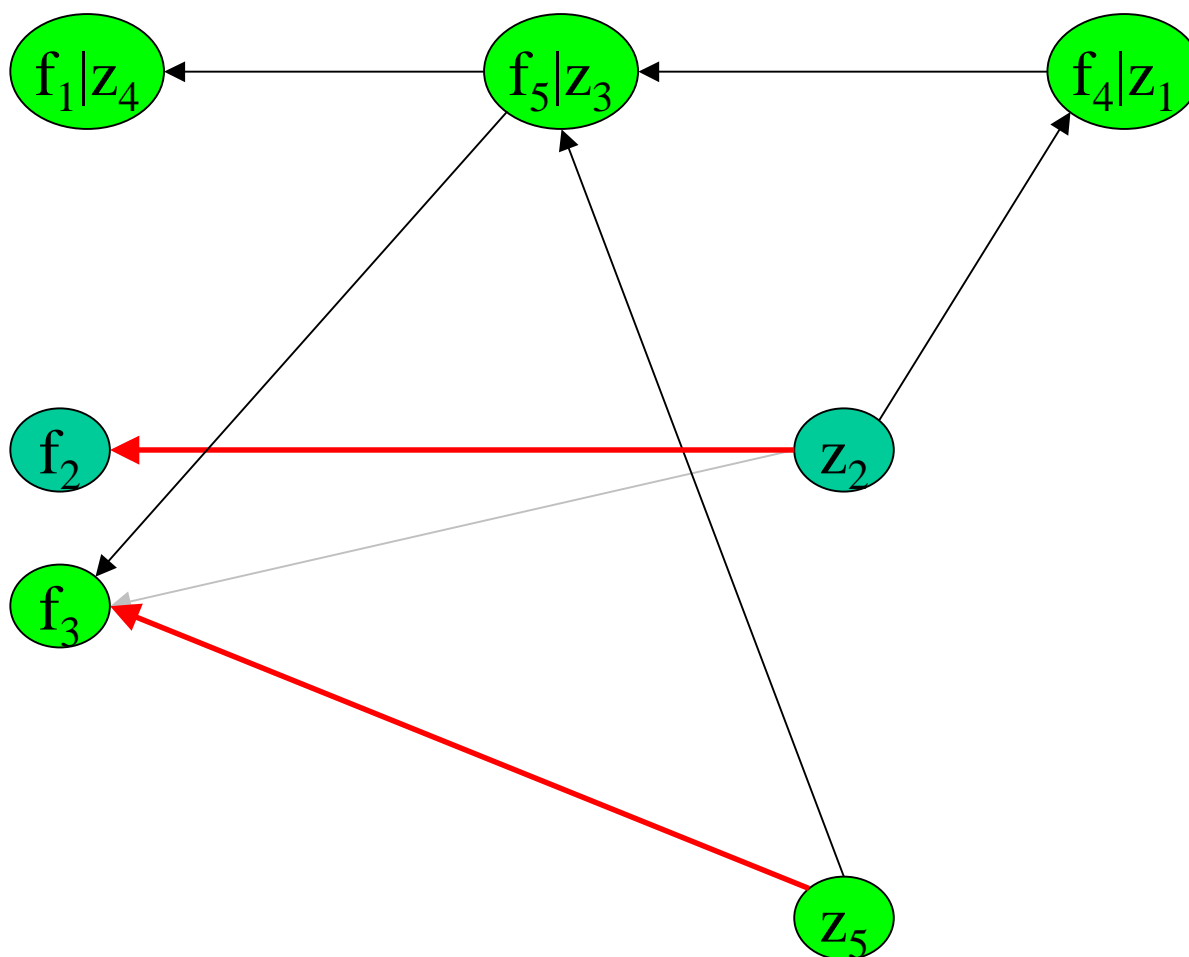
# Construct a Directed Graph



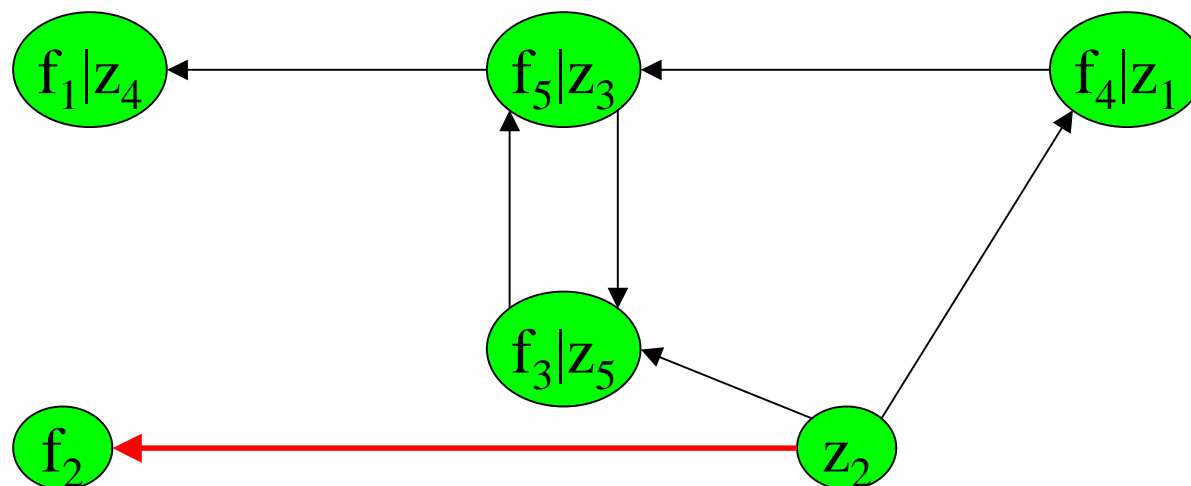
# Construct a Directed Graph



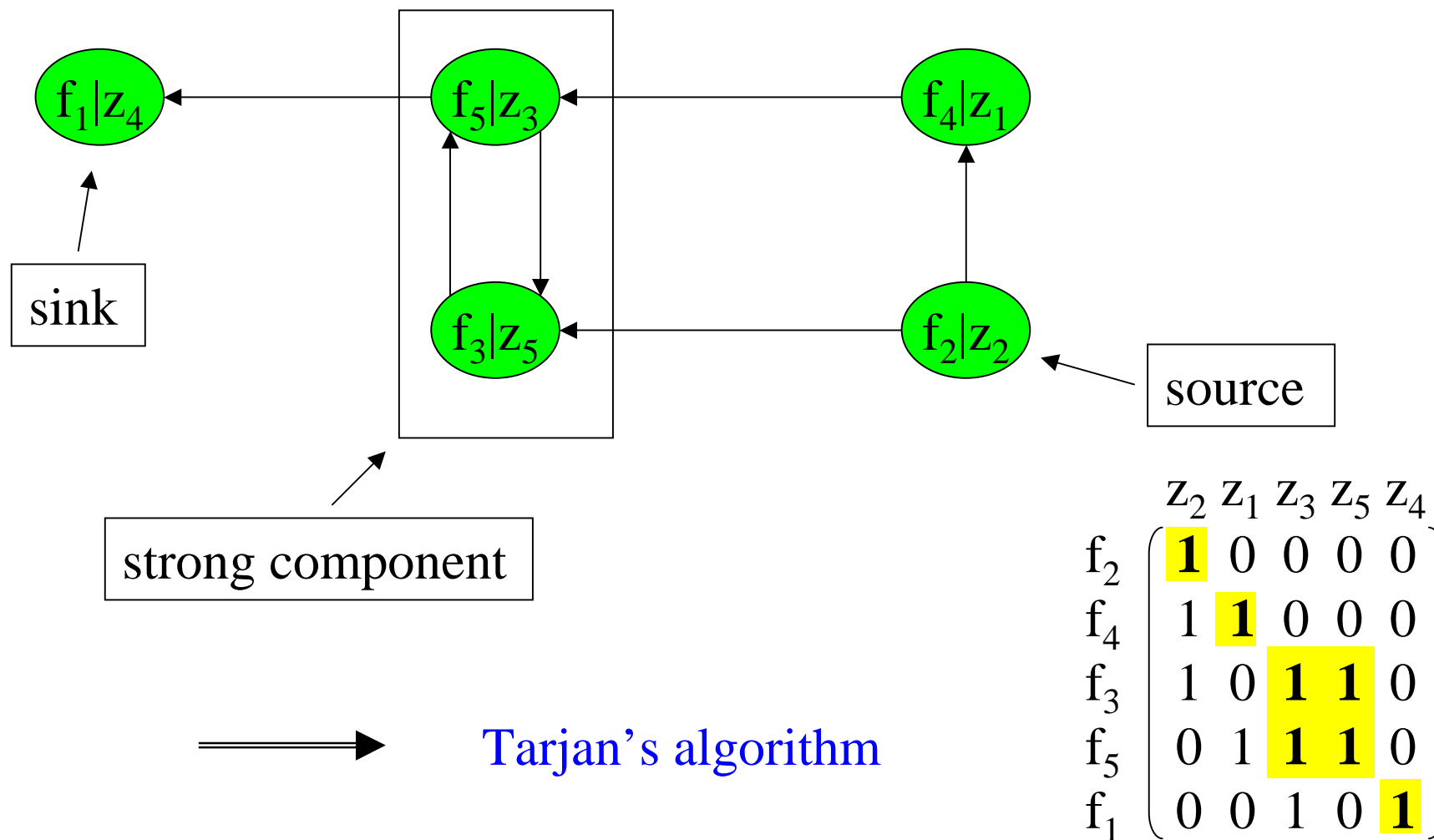
# Construct a Directed Graph



# Construct a Directed Graph



# Construct a Directed Graph



# Transformation Algorithms and the BLT-Form (Block-Lower-Triangle Form)

Example of a regular DAE:

$$\begin{array}{lll}
 f_1(z_3, z_4) = 0 & f_1(z_3, [z_4]) = 0 & f_2(\underline{z}_2) = 0 \\
 f_2(z_2) = 0 & f_2([z_2]) = 0 & f_4(\underline{z}_1, z_2) = 0 \\
 f_3(z_2, z_3, z_5) = 0 & f_3(z_2, z_3, [z_5]) = 0 & f_3(z_2, \underline{z}_3, \underline{z}_5) = 0 \\
 f_4(z_1, z_2) = 0 & f_4([z_1], z_2) = 0 & f_5(z_1, \underline{z}_3, \underline{z}_5) = 0 \\
 f_5(z_1, z_3, z_5) = 0 & f_5(z_1, [z_3], z_5) = 0 & f_1(z_3, \underline{z}_4) = 0
 \end{array}$$

BLT-form of the adjacency matrix

$$\begin{array}{c}
 f_1 \\
 f_2 \\
 f_3 \\
 f_4 \\
 f_5
 \end{array}
 \begin{pmatrix}
 z_1 & z_2 & z_3 & z_4 & z_5 \\
 0 & 0 & 1 & 1 & 0 \\
 0 & 1 & 0 & 0 & 0 \\
 0 & 1 & 1 & 0 & 1 \\
 1 & 1 & 0 & 0 & 0 \\
 1 & 0 & 1 & 0 & 1
 \end{pmatrix}$$

$$\begin{array}{c}
 f_1 \\
 f_2 \\
 f_3 \\
 f_4 \\
 f_5
 \end{array}
 \begin{pmatrix}
 z_1 & z_2 & z_3 & z_4 & z_5 \\
 0 & 0 & 1 & \mathbf{1} & 0 \\
 0 & \mathbf{1} & 0 & 0 & 0 \\
 0 & 1 & 1 & 0 & \mathbf{1} \\
 \mathbf{1} & 1 & 0 & 0 & 0 \\
 1 & 0 & \mathbf{1} & 0 & 1
 \end{pmatrix}$$

$$\begin{array}{c}
 f_2 \\
 f_4 \\
 f_3 \\
 f_5 \\
 f_1
 \end{array}
 \begin{pmatrix}
 z_2 & z_1 & z_3 & z_5 & z_4 \\
 \mathbf{1} & 0 & 0 & 0 & 0 \\
 1 & \mathbf{1} & 0 & 0 & 0 \\
 1 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
 0 & 1 & \mathbf{1} & \mathbf{1} & 0 \\
 0 & 0 & 1 & 0 & \mathbf{1}
 \end{pmatrix}$$

# Structural Regularity of the Adjacency Matrix

- Transformation algorithm works on the model structure
  - Assign each variable to exact one equation (matching problem)
  - Find equations which have to be solved simultaneously
- Nevertheless, certain parameter settings could make a model singular

model singular

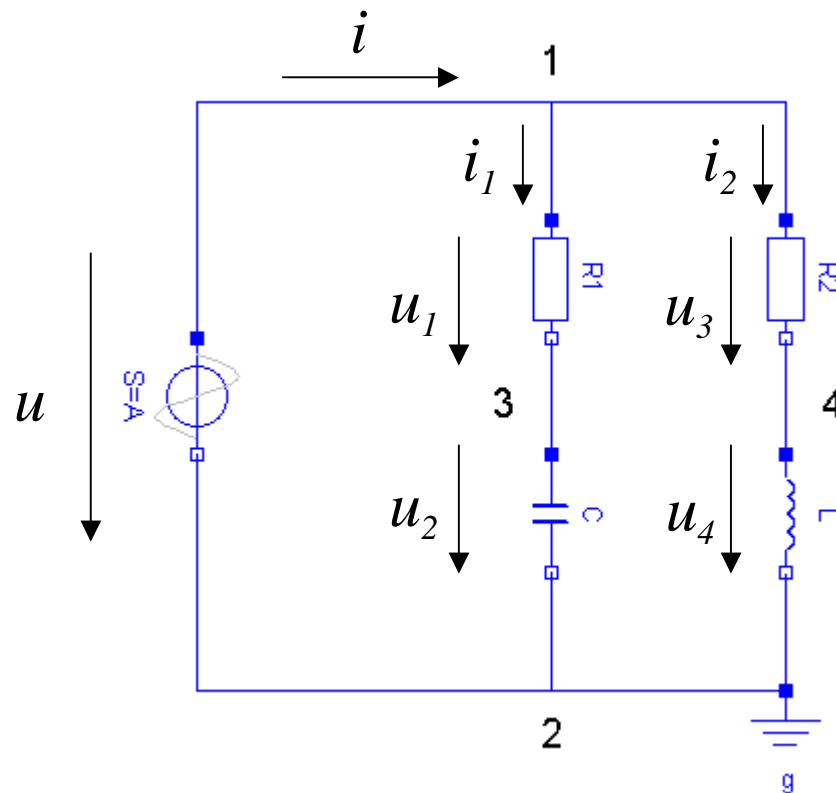
$$\Leftrightarrow \begin{vmatrix} a & b \\ 1 & 1 \end{vmatrix} = a - b = 0$$

$$\Leftrightarrow a = b$$

```

model SingularModel
  parameter Real a=2;
  parameter Real b=1;
  Real x,y;
equation
  a*der(x) + b*der(y) = sin(time);
  der(x) + der(y) = cos(time);
end SingularModel
  
```

# Example: A Simple Electrical System



$$u = A \cdot \sin(\omega t)$$

$$u = u_1 + u_2$$

$$u_3 = R_2 i_2$$

$$u = u_3 + u_4$$

$$u_1 = R_1 i_1$$

$$i = i_1 + i_2$$

$$C \frac{du_2}{dt} = i_1$$

$$L \frac{di_2}{dt} = u_4$$

# Efficiency Issues and Non-Linear Equations

## ■ Transformed equation system

- Efficient calculation of the unknowns and derivatives of the states
- Solve single linear equations based on causality

$$\underline{0} = \underline{f}\left(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p}\right)$$



## ■ Algebraic loops

- Linear equation systems can be treated by direct or iterative routines
- Non-linear equation systems need more sophisticated solution techniques (only local convergence!)

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



$$\underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \underline{g}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right)$$



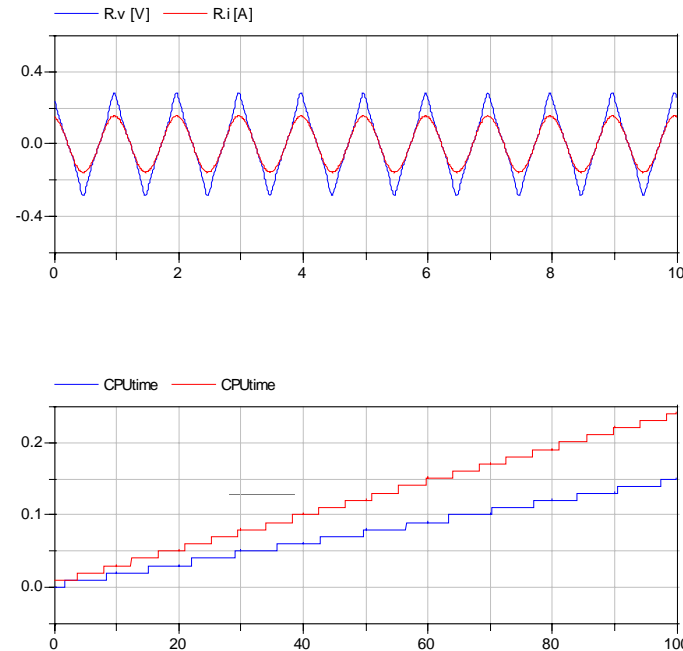
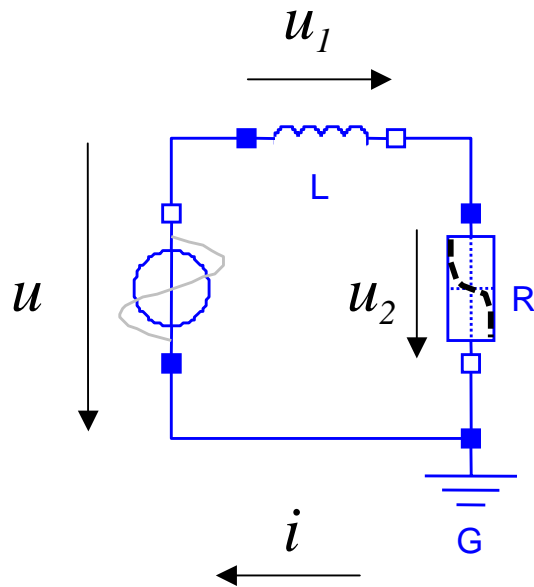
$$\underline{\dot{x}}(t) = \underline{h}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right)$$

$$\underline{y}(t) = \underline{k}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right)$$

## ■ Real-time aspects

- Causality of non-linear equation may influence the runtime efficiency

# Example: A Non-Linear Electrical System



$$u = A \cdot \sin(\omega t)$$

$$i = a \tanh(b u_2)$$

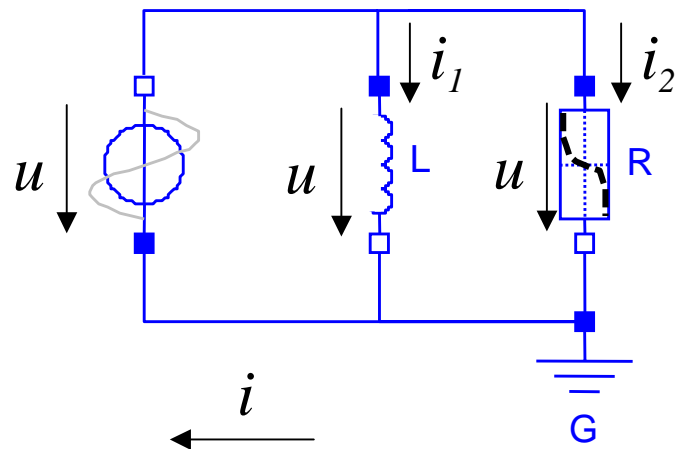
$$u = u_1 + u_2$$

$$L \frac{di}{dt} = u_1$$

Re-writing the non-linear equation (Causality):

$$u_2 = \frac{1}{2b} \log \left( \frac{(1 + i/a)}{(1 - i/a)} \right)$$

# Example: A Non-Linear Electrical System



$$u = A \cdot \sin(\omega t)$$

$$i_1 = a \tanh(b u)$$

$$L \frac{di_2}{dt} = u$$

$$i = i_1 + i_2$$

Causality of the non-linear equation correct

# Higher Index Problems

## Structurally Singular Systems

### Higher-index DAEs

- Differential index of a DAE
- Structural singularity of the adjacency matrix
- Index reduction method using symbolic differentiation of equations

$$\underline{0} = \underline{f}(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p})$$



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



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



### Numerical issues

- Consistent initialization
- Drift phenomenon
- Dummy derivative method

### State selection mechanism in Modelica

- See also initialization of 3-phase electrical system

$$\underline{\dot{x}}(t) = \underline{h}(t, \underline{x}(t), \underline{u}(t), \underline{p})$$

$$\underline{y}(t) = \underline{k}(t, \underline{x}(t), \underline{u}(t), \underline{p})$$

## General representation of DAEs:

$$\underline{0} = \underline{f}\left(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p}\right)$$

$t$  time

$\underline{\dot{x}}(t)$  vector of differentiated state variables

$\underline{x}(t)$  vector of state variables

$\underline{y}(t)$  vector of algebraic variables

$\underline{u}(t)$  vector of input variables

$\underline{p}$  vector of parameters and/or constants

## Differential index of a DAE:

The minimal number of analytical differentiations of the equation system necessary to extract by algebraic manipulations an explicit ODE for all unknowns.

DAE with differential index 0:

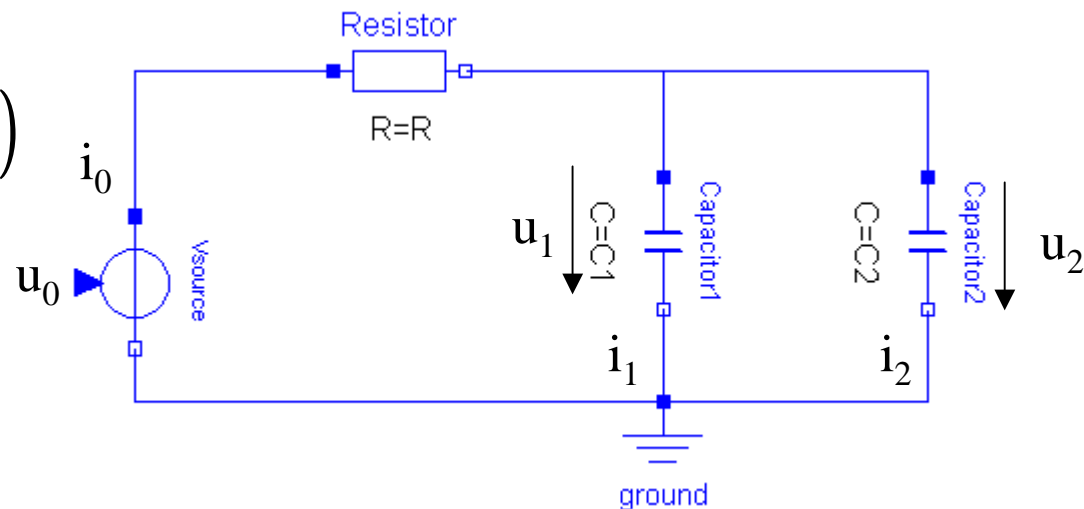
$$\underline{0} = \underline{f}\left(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{u}(t), \underline{p}\right) \longrightarrow \underline{\dot{x}}(t) = \underline{g}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right)$$

DAE with differential index 1:

$$\begin{array}{ccc} \underline{0} = \underline{f}\left(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p}\right) & & \\ \downarrow & & \\ \begin{array}{ccc} \underline{\dot{x}}(t) = \underline{h}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right) & \longrightarrow & \underline{\dot{x}}(t) = \underline{h}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right) \\ \underline{y}(t) = \underline{k}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right) & & \underline{\dot{y}}(t) = \frac{d}{dt} \underline{k}\left(t, \underline{x}(t), \underline{u}(t), \underline{p}\right) \end{array} \end{array}$$

# Higher-Index-DAE Example (index 2)

$$0 = \underline{f}(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p})$$



$$C_1 \cdot \dot{u}_1 = i_1$$

$$C_2 \cdot \dot{u}_2 = i_2$$

$$u_1 = u_2$$

$$i_0 = i_1 + i_2$$

$$u_0 = R \cdot i_0 + u_1$$

$$\mathbf{x} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

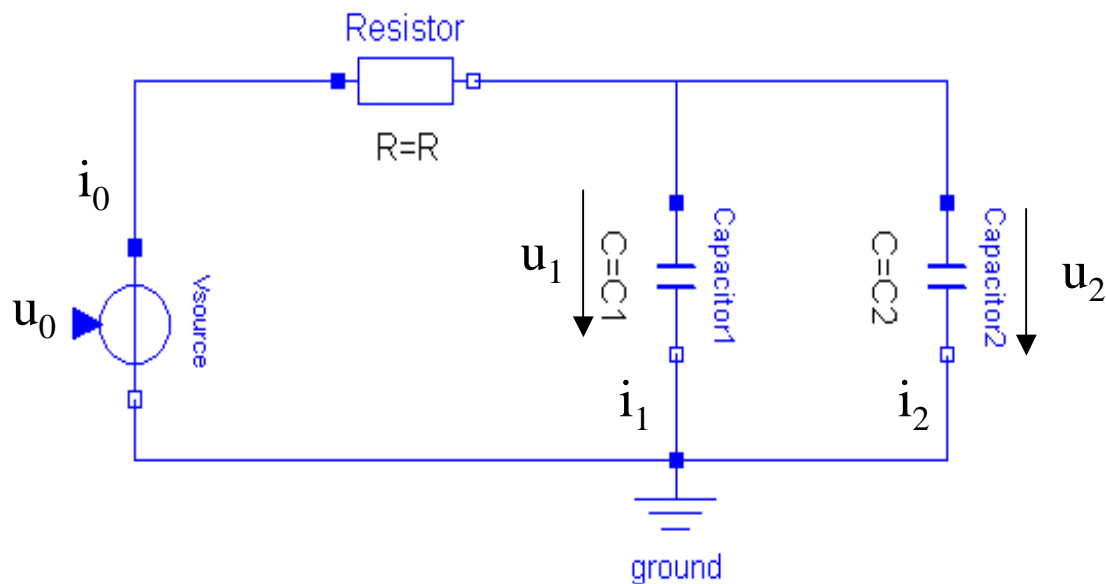
$$\mathbf{y} = \begin{bmatrix} i_0 \\ i_1 \\ i_2 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ i_0 \\ i_1 \\ i_2 \end{bmatrix}$$

5 equations and 5 unknowns.

**BUT:** system is singular!

constrained equation  
between states

# Higher-Index-DAE Example (index 2)



$$C_1 \cdot \dot{u}_1 = i_1$$

$$C_2 \cdot \dot{u}_2 = i_2$$

~~$$u_1 = u_2$$~~

$$i_0 = i_1 + i_2$$

$$u_0 = R \cdot i_0 + u_1$$

$$\dot{u}_1 = \dot{u}_2$$

## Algorithmic steps

- Differentiation of constrained equations
- Extracting underlying ODE (DAE of index 1)

⇒ **ODE with 5 equations and 5 unknowns**

- Consistent initialization necessary

$$\mathbf{x} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ i_0 \\ i_1 \\ i_2 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} i_0 \\ i_1 \\ i_2 \end{bmatrix}$$

# Higher-Index-DAEs -- Numerical Problems

- Consistent initial conditions
  - Relation between states are eliminated when differentiating
  - Initial conditions need to be determined using the algebraic constraints
  - Automatic procedure possible using assign algorithm on the constrained equations
  
- Drift phenomenon
  - Algebraic constraints no longer fulfilled during simulation
  - Even worse when simulating stiff problems

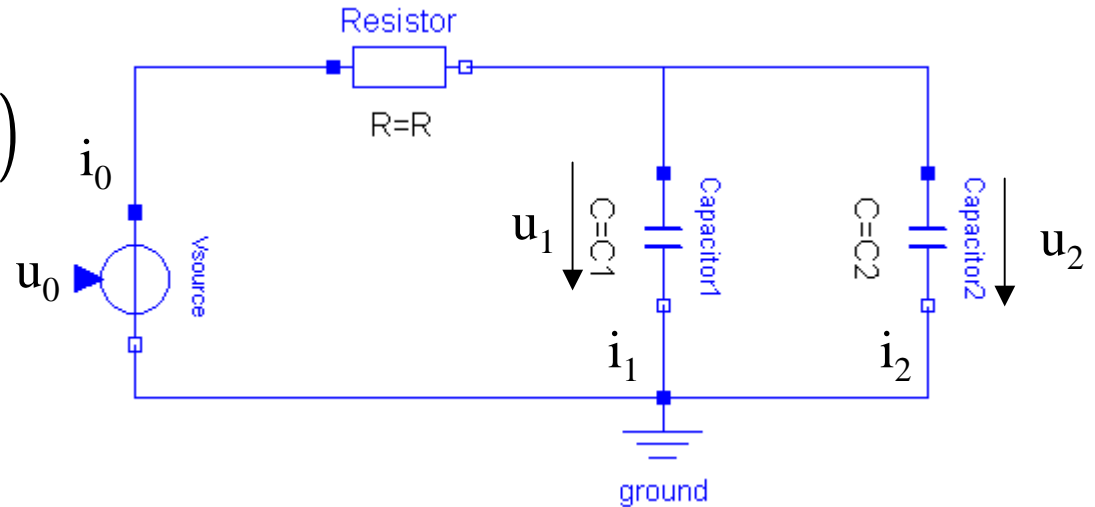
=> Dummy-Derivative method

- Matching algorithm fails
  - System is structurally singular
  - Find minimal subset of equations
    - more equations than unknown variables
  - Singularity is due to equations constraining states
  
- Differentiate subset of equations
  - Static state selection during compile time
    - choose one state and corresponding derivative as purely algebraic variable
      - so-called dummy-state and dummy derivative
    - by differentiation introduced variables are algebraic
    - continue matching algorithm
    - check initial conditions
  - Dynamic state selection during simulation time
    - store information on constrained states
    - make selection dynamically based on heuristic criteria
    - new state selection triggers an event (re-initialize states)

# Dummy-Derivative method

## Example (index 2)

$$\underline{0} = \underline{f}\left(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p}\right)$$



$$C_1 \cdot \dot{u}_1 = i_1$$

$$C_2 \cdot \dot{u}_2 = i_2$$

$$u_1 = u_2$$

$$i_0 = i_1 + i_2$$

$$u_0 = R \cdot i_0 + u_1$$

$$\mathbf{x} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

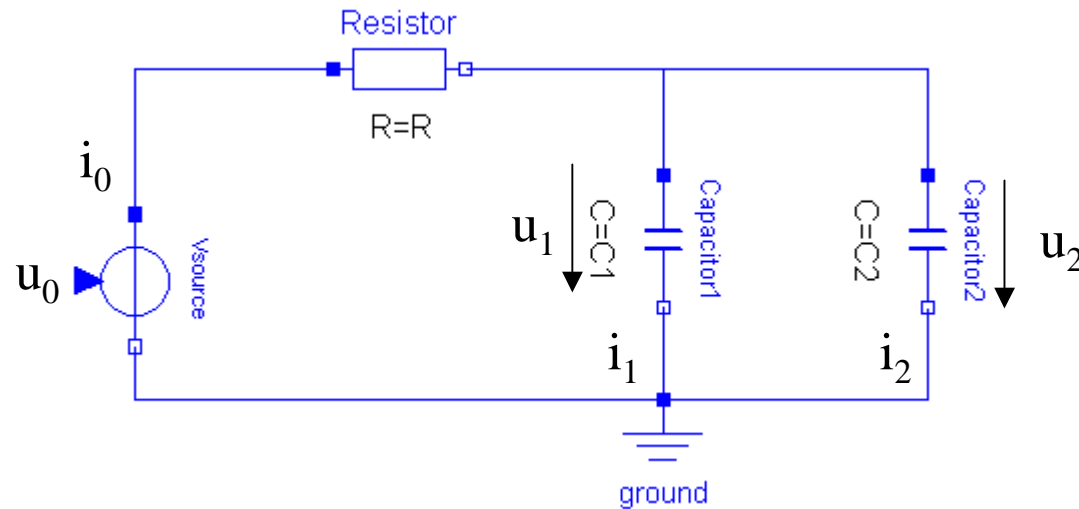
$$\mathbf{y} = \begin{bmatrix} i_0 \\ i_1 \\ i_2 \end{bmatrix}$$

$$\mathbf{z} = \begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ i_0 \\ i_1 \\ i_2 \end{bmatrix}$$

constrained equation

# Dummy-Derivative method

## Example (index 2)



$$C_1 \cdot \dot{u}_1 = i_1$$

$$C_2 \cdot u_2' = i_2$$

$$u_1 = u_2$$

$$i_0 = i_1 + i_2$$

$$u_0 = R \cdot i_0 + u_1$$

$$\dot{u}_1 = u_2'$$

### Algorithmic steps

- Differentiation of constrained equations
- Choose  $u_2$  as dummy-state and  $u_2'$  as dummy-derivative
- Extracting underlying ODE (DAE of index 1)

⇒ **ODE with 6 equations and 6 unknowns**

- Check initial condition

$$\mathbf{x} = \begin{bmatrix} u_1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} i_0 \\ i_1 \\ i_2 \\ u_2 \\ u_2' \end{bmatrix}$$

$$\mathbf{z} = \begin{bmatrix} \dot{u}_1 \\ i_0 \\ i_1 \\ i_2 \\ u_2 \\ u_2' \end{bmatrix}$$

## ■ Influence default state selection mechanism

- Attribute `StateSelect stateSelect = StateSelect.default;`

- Possible choices:

- `never` – Do not use as state at all

- `avoid` – Use as state, if it cannot be avoided (but only if variable appears differentiated and no other potential state with attribute default, prefer, or always can be selected)

- `default` – Use as state if appropriate, but only if variable appears differentiated

- `prefer` – Prefer it as state over those having the default value (also variables can be selected, which do not appear differentiated)

- `always` – Do use it as a state

# Initialization of Dynamic Models

## ■ Initialization of “free” state variables

- Transformed DAE after index-reduction
- States can be chosen at start time
  - same number of additional equations and “free” states

$$\underline{0} = \underline{f}(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p})$$



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



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



$$\underline{\dot{x}}(t) = \underline{h}(t, \underline{x}(t), \underline{u}(t), \underline{p})$$

$$\underline{y}(t) = \underline{k}(t, \underline{x}(t), \underline{u}(t), \underline{p})$$

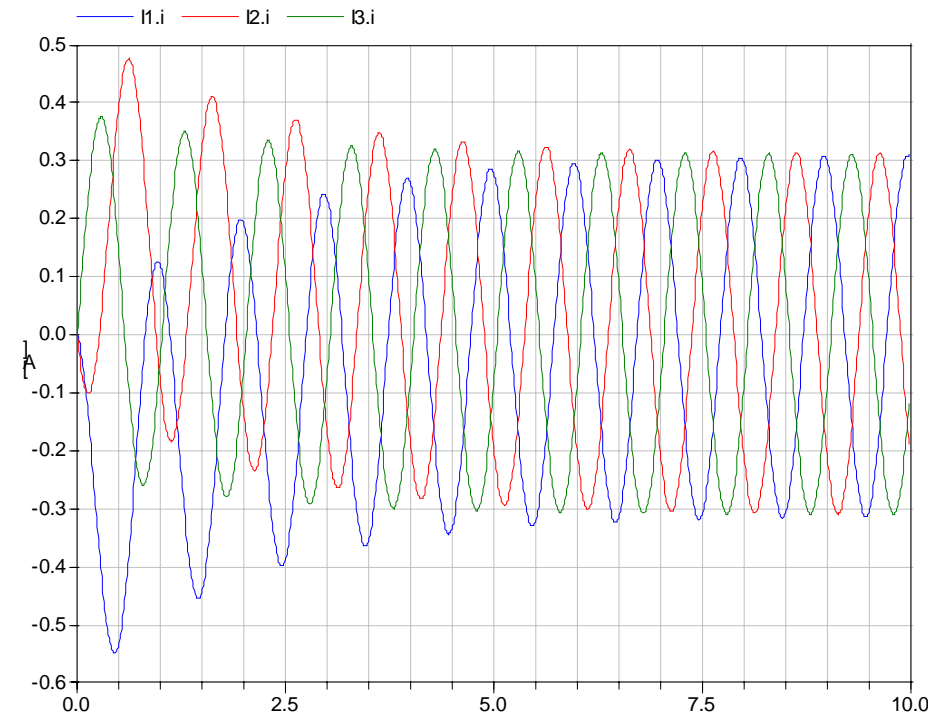
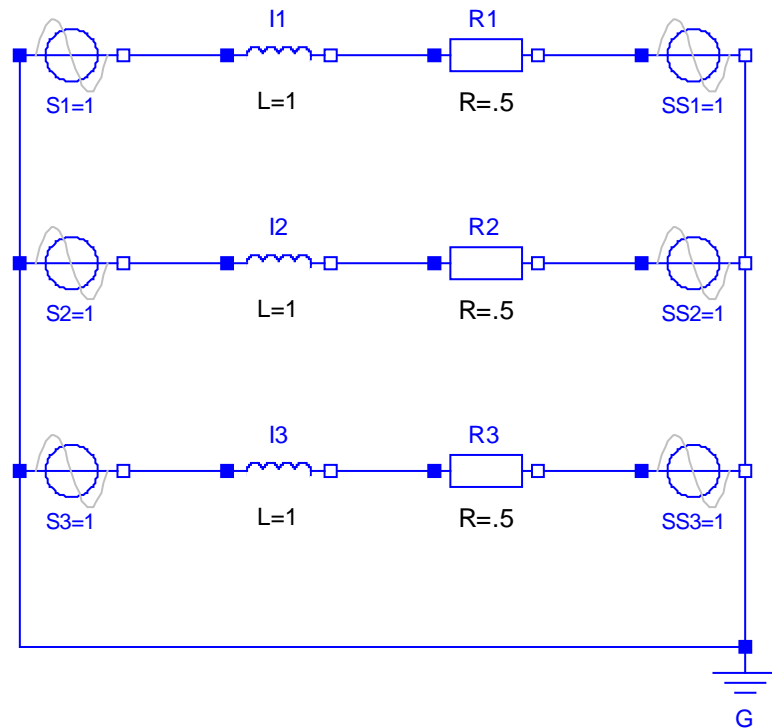
## ■ Initialization of parameters

- Determine parameter settings
- Parameters can be calculated at start time
  - same number of additional equations and “free” parameters

## ■ Initialization mechanism in Modelica

- attribute start
- initial equation section
  - attribute fixed for parameters

# Example: 3-Phase Electrical System



## Steady-state initialization?

- States  $I1.i$ ,  $I2.i$ ,  $I3.i$  are not constant at initial time!
- Here: Initial values set to 0 (standard settings, attribute `start`)
- Transformation to rotating reference system necessary

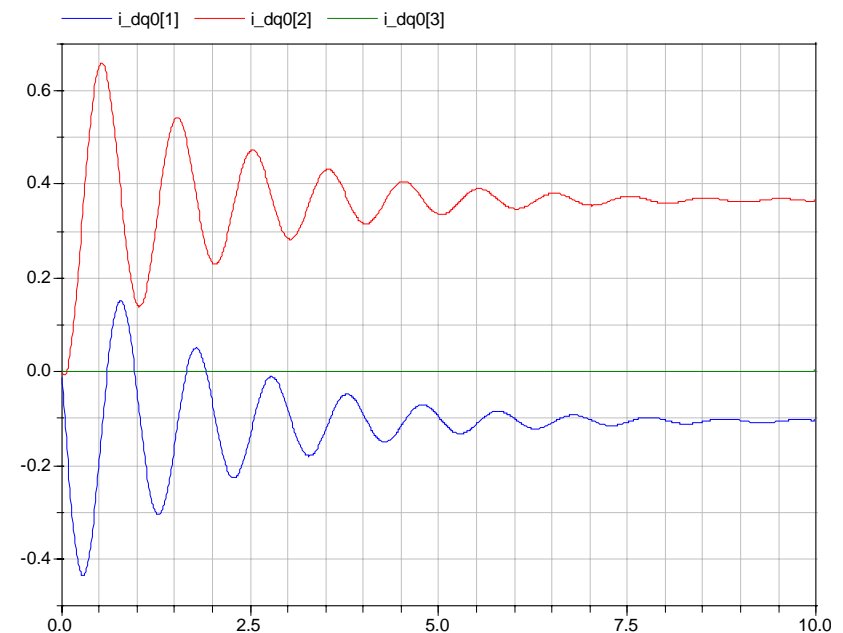
# Example: 3-Phase Electrical System

## Park-Transformation to rotating reference system:

- Write states as vector  $i\_abc[3] = \{I1.i, I2.i, I3.i\}$
- Park-transformation to dq0-reference frame

$$P = \frac{\sqrt{2}}{\sqrt{3}} \begin{pmatrix} \sin(\omega t) & \sin\left(\omega t + \frac{2\pi}{3}\right) & \sin\left(\omega t + \frac{4\pi}{3}\right) \\ \cos(\omega t) & \cos\left(\omega t + \frac{2\pi}{3}\right) & \cos\left(\omega t + \frac{4\pi}{3}\right) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$i\_dq0 = P \cdot i\_abc$$



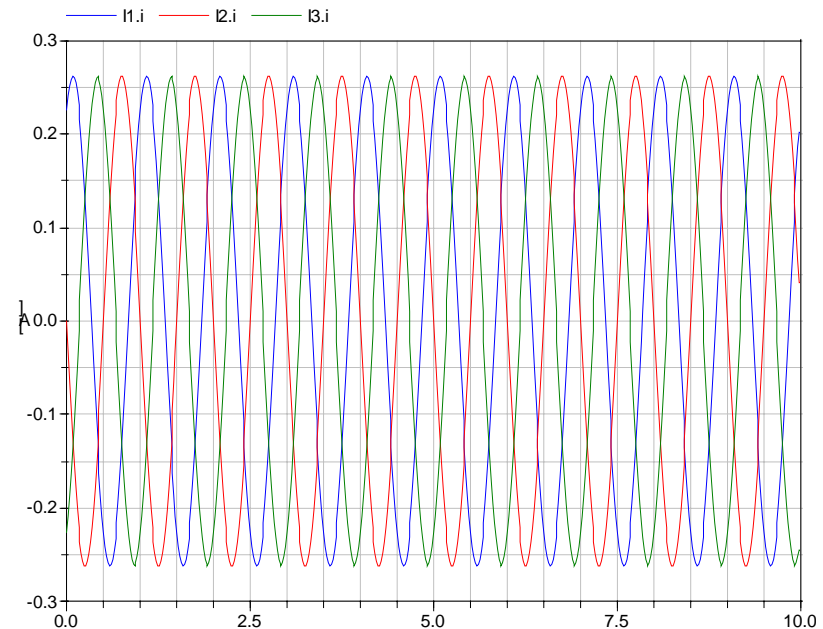
# Example: 3-Phase Electrical System

## Initialize States

```
model Test3PhaseSystemInit
  parameter Real shift=0.4;
  Real i_abc[3]={I1.i,I2.i,I3.i};
  Real i_dq0[3];
  ...
  initial equation
    der(i_dq0)={0,0,0};
  equation
    ...
    i_dq0 = P*i_abc;
end Test3PhaseSystemInit
```

### Steady-state initialization:

- Derivatives of  $i_{dq0}$  are only introduced during initialization
- Differentiation of  $i_{dq0} = P \cdot i_{abc}$  necessary
- No higher-index problem



# Example: 3-Phase Electrical System

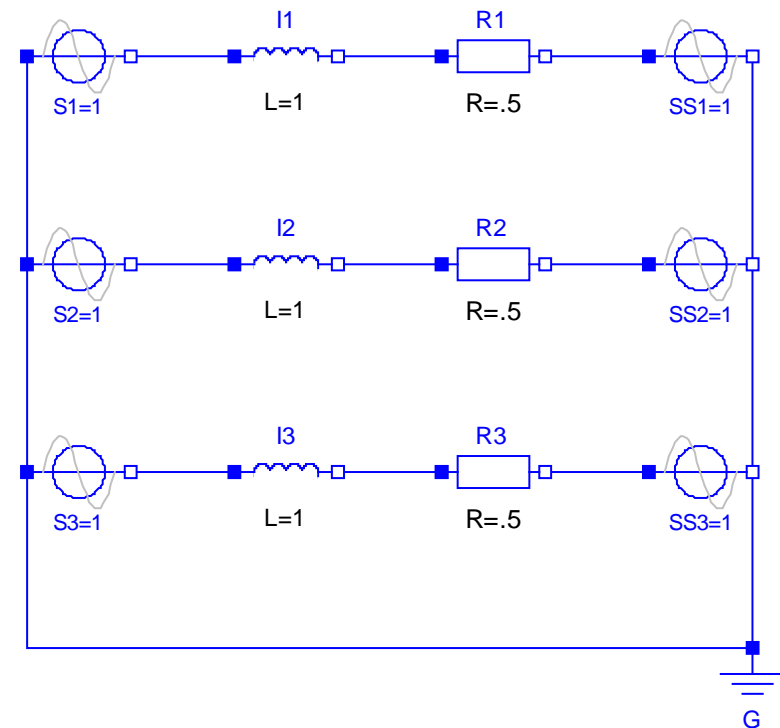
## Initialize Parameters

```

model Test3PhaseSystemInitParam
  parameter Real shift(fixed=false,start=0.1);
  Real i_abc[3]={I1.i,I2.i,I3.i}, u_abc[3]={S1.v,S2.v,S3.v};
  ...
initial equation
  der(i_dq0)={0,0,0};
  power = -0.12865;
equation
  ...
  u_dq0 = P*u_abc;
  i_dq0 = P*i_abc;
  power = u_dq0*i_dq0;
end Test3PhaseSystemInitParam
  
```

### Parameter initialization:

- Non-linear equation, no unique solution  
 $\text{power} = u_{dq0} \cdot i_{dq0} = -0.12865$
- Attribute fixed (default true)
- Attribute start (default 0)



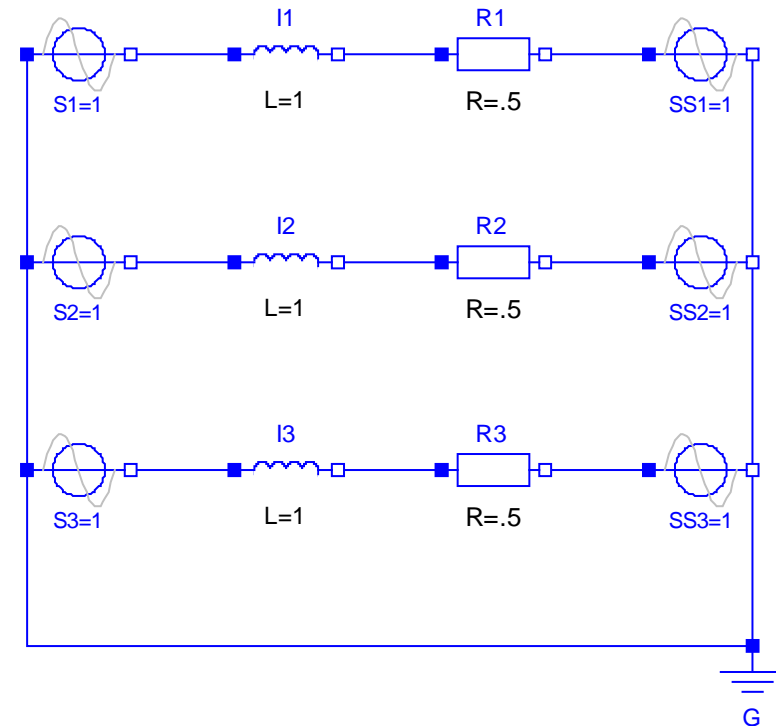
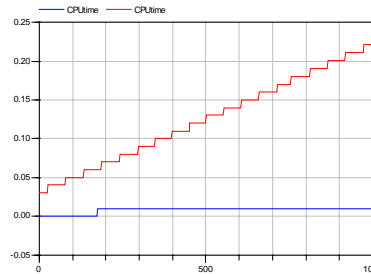
# Example: 3-Phase Electrical System

## Influence State Selection

```

model Test3PhaseSystemStateSelect
  Real i_abc[3]={I1.i,I2.i,I3.i};
  Real i_dq0[3](each stateSelect=StateSelect.always);

  ...
  initial equation
  ...
  equation
  ...
  i_dq0 = P*i_abc;
end Test3PhaseSystemStateSelect
  
```



### State selection:

- Introduce  $i_{dq0}$  as states
 

```

      Real di_dq0[3];
      der(i_dq0)=di_dq0;
      
```
- Constrained equation between states must be differentiated
 
$$i_{dq0} = P \cdot i_{abc};$$

# Example: 3-Phase Electrical System

## Introduce Dummy-States/Derivatives

Mathematical formulas to the Park-transformation:  
(using trigonometric identities)

$$P = \frac{\sqrt{2}}{\sqrt{3}} \begin{pmatrix} \sin(\omega t) & \sin\left(\omega t + \frac{2\pi}{3}\right) & \sin\left(\omega t + \frac{4\pi}{3}\right) \\ \cos(\omega t) & \cos\left(\omega t + \frac{2\pi}{3}\right) & \cos\left(\omega t + \frac{4\pi}{3}\right) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$P \cdot P^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$P' \cdot P^T = \begin{pmatrix} 0 & \omega & 0 \\ -\omega & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$i_{dq0} = P \cdot i_{abc}$$



$$\frac{d}{dt} i_{dq0} = \omega \cdot \begin{pmatrix} i_{dq0_2} \\ -i_{dq0_1} \\ 0 \end{pmatrix} + P \cdot \frac{d}{dt} i_{abc}$$

# Example: 3-Phase Electrical System

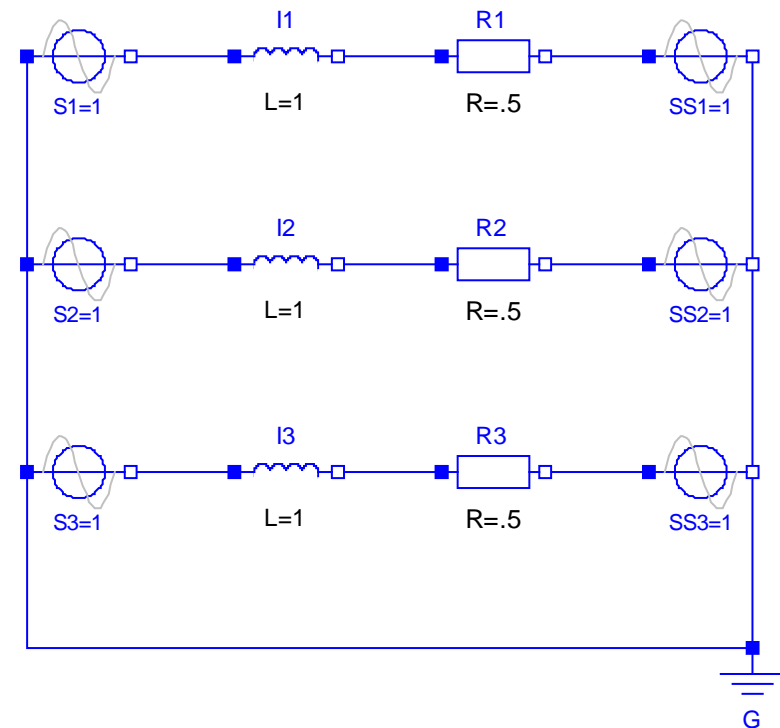
## Introduce Dummy-States/Derivatives

```

model Test3PhaseSystemDummyInit
  Real i_abc[3]={I1.i,I2.i,I3.i};
  Real i_dq0[3];
  Real di_dq0[3]=2*pi*{i_dq0[2],-i_dq0[1],0} + P*der(i_abc);
  ...
initial equation
  di_dq0={0,0,0};
equation
  ...
  i_dq0 = P*i_abc;
end Test3PhaseSystemDummyInit
  
```

### State selection:

- Introduce  $di\_dq0$  as dummy derivative
- Only used during initialization
- No differentiation necessary
- States  $I1.i$ ,  $I2.i$ ,  $I3.i$  used during simulation



# Example: 3-Phase Electrical System

## Introduce Dummy-States/Derivatives

### State selection:

- Introduce  $di\_abc$  as dummy derivative
- Rewrite inductor equation:  $L \cdot di = v$ ;
- No differentiation necessary
- Efficiency increase during simulation (real-time aspect)

```
model Test3PhaseSystemDummy
  Real i_abc[3]={I1.i,I2.i,I3.i};
  Real di_abc[3]={I1.di,I2.di,I3.di};
  Real i_dq0[3];
```

...

### initial equation

```
der(i_dq0)={0,0,0};
```

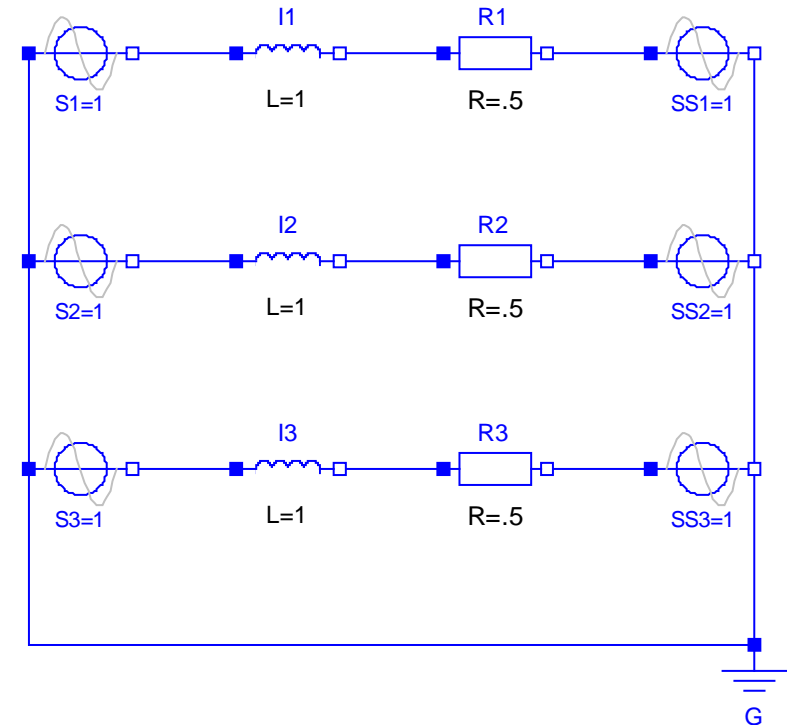
### equation

...

```
i_dq0 = P*i_abc;
```

```
der(i_dq0)=2*pi*{i_dq0[2],-i_dq0[1],0} + P*di_abc;
```

```
end Test3PhaseSystemDummy
```



# Numerical Integration of ODEs/DAEs

## ■ Integration methods for ODEs

- Single-step methods
  - explicit and implicit Runge-Kutta formulas
  - fixed and variable step-size
    - Richardson-Extrapolation
    - embedded Runge-Kutta Formulas (Dopri5)
- Multi-step methods
  - explicit and implicit schemes
    - Adams/Bashforth/Moulton formulas
    - BDF-formulas (i.e. DASSL)
  - start-up using single-step methods (events)
- Event handling
  - stop simulation and restart integration

## ■ Stability region and stiffness detection

- Linearize the system
- Determine eigenvalues

$$\underline{0} = \underline{f}(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p})$$



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



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



$$\underline{\dot{x}}(t) = \underline{h}(t, \underline{x}(t), \underline{u}(t), \underline{p})$$

$$\underline{y}(t) = \underline{k}(t, \underline{x}(t), \underline{u}(t), \underline{p})$$

# Stability Analysis of Runge-Kutta Methods

ODE problem:

$$\dot{\underline{x}}(t) = \underline{f}(t, \underline{x}(t))$$

describing disturbance:

$$\underline{y}'(t) = J(t) \cdot \underline{y}(t), \quad J(t) = \frac{\partial \underline{f}}{\partial \underline{x}}(t, \underline{x})$$

Euler integration:  
( $J$  constant)

$$\underline{y}_{m+1} = \underline{y}_m + \Delta t \cdot J \cdot \underline{y}_m = R(\Delta t \cdot J) \cdot \underline{y}_m$$

$$R(z) = 1 + z$$

eigenvector  $\underline{y}_0$   
eigenvalue  $\lambda$

$$\underline{y}_m = [R(\Delta t \cdot \lambda)]^m \cdot \underline{y}_0$$

bounded solution, iff

$$|R(\Delta t \cdot \lambda)| \leq 1, \quad \lambda \text{ eigenvalue of } J$$

# Stability Analysis of Runge-Kutta Methods

Dahlquist test equation:  $y' = \lambda \cdot y, \quad y_0 = 1, \quad z = \Delta t \cdot \lambda$

Stability function  $R(z)$  is corresponding numerical solution at  $t + \Delta t$ :

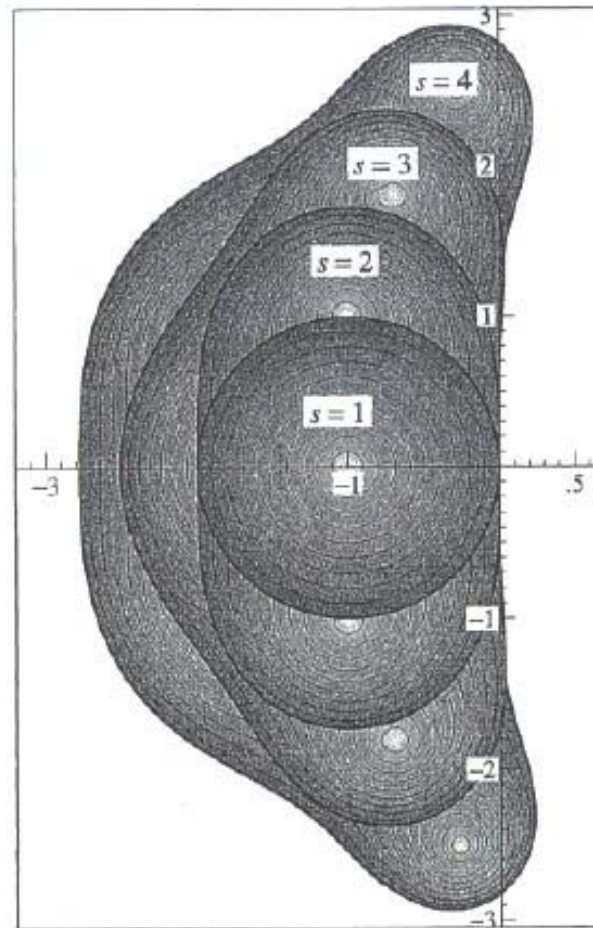
Runge-Kutta method  
(order=p): 
$$R(z) = 1 + \frac{z}{1!} + \dots + \frac{z^p}{p!} + O(z^{p+1})$$

Runge-Kutta method  
(order=s, explicit, s\_stage): 
$$R(z) = 1 + \frac{z}{1!} + \dots + \frac{z^s}{s!}$$

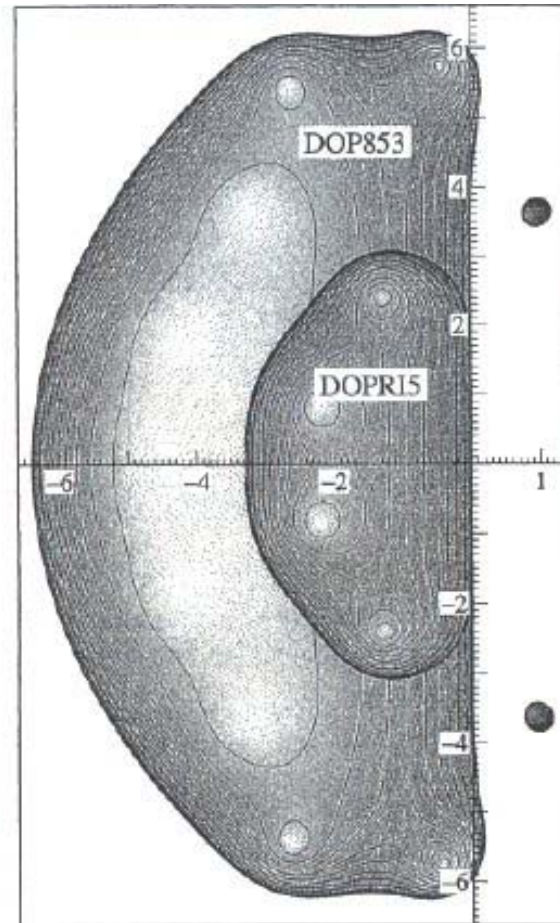
Dopri5  
(order=5, explicit, 6\_stage): 
$$R(z) = 1 + \frac{z}{1!} + \dots + \frac{z^5}{5!} + \frac{z^6}{600}$$

bounded solution, iff  $|R(\Delta t \cdot \lambda)| \leq 1, \quad \lambda \text{ eigenvalue of } J$

# Stability Region for Explicit Runge-Kutta Methods



**Fig. 2.1.** Stability domains for explicit Runge-Kutta methods of order  $p = s$



**Fig. 2.2.** Stability domains for DOPRI methods

# Stability Region for Implicit Runge-Kutta Methods

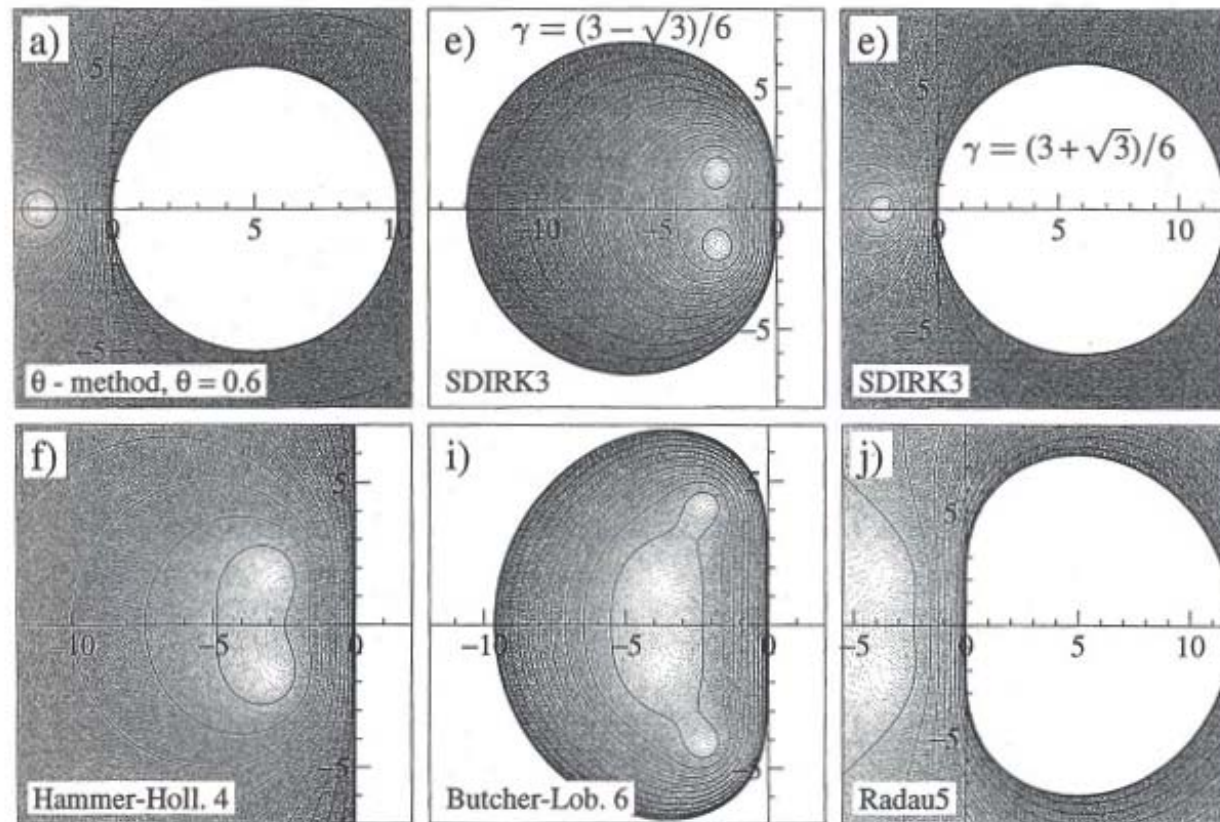


Fig. 3.1. Stability domains for implicit Runge-Kutta methods

# Stability Region for Adams-Multi-Step Methods

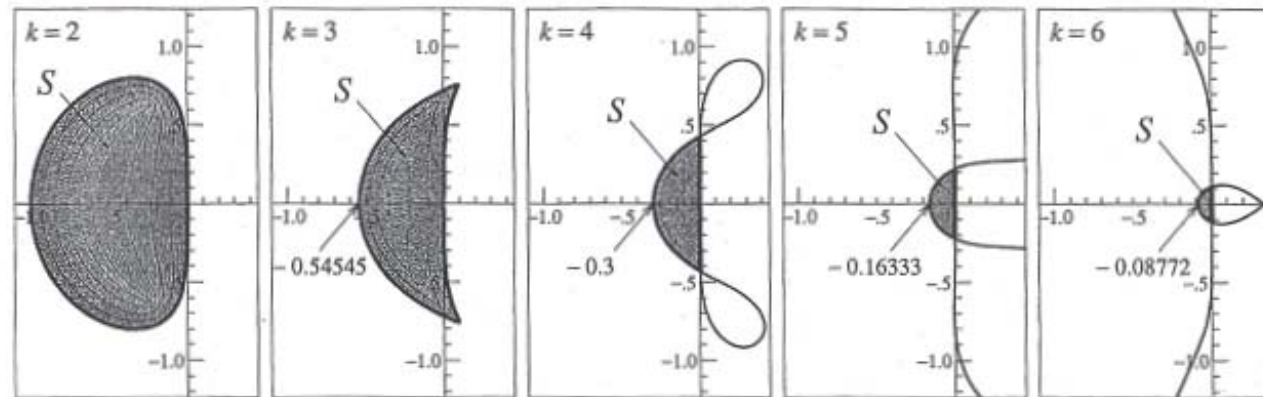


Fig. 1.2. Stability domains for explicit Adams methods

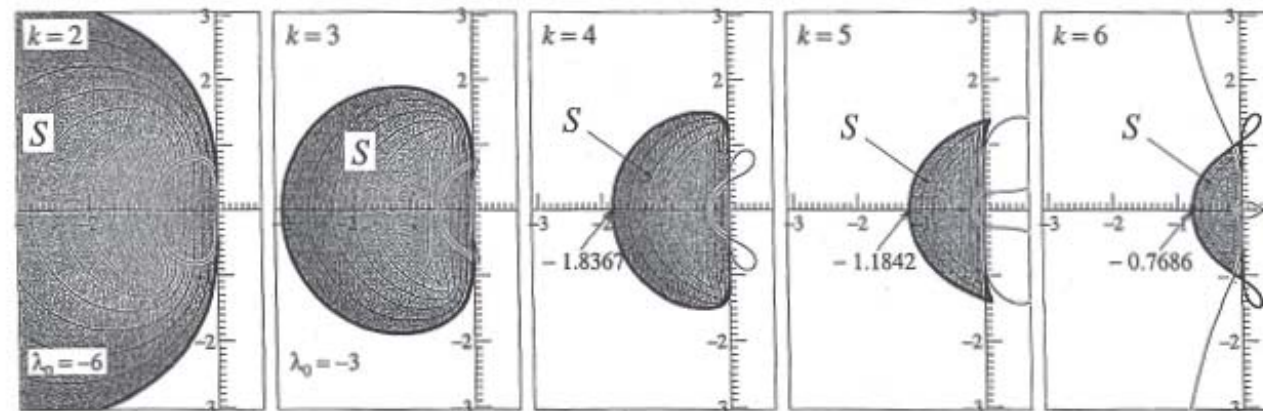
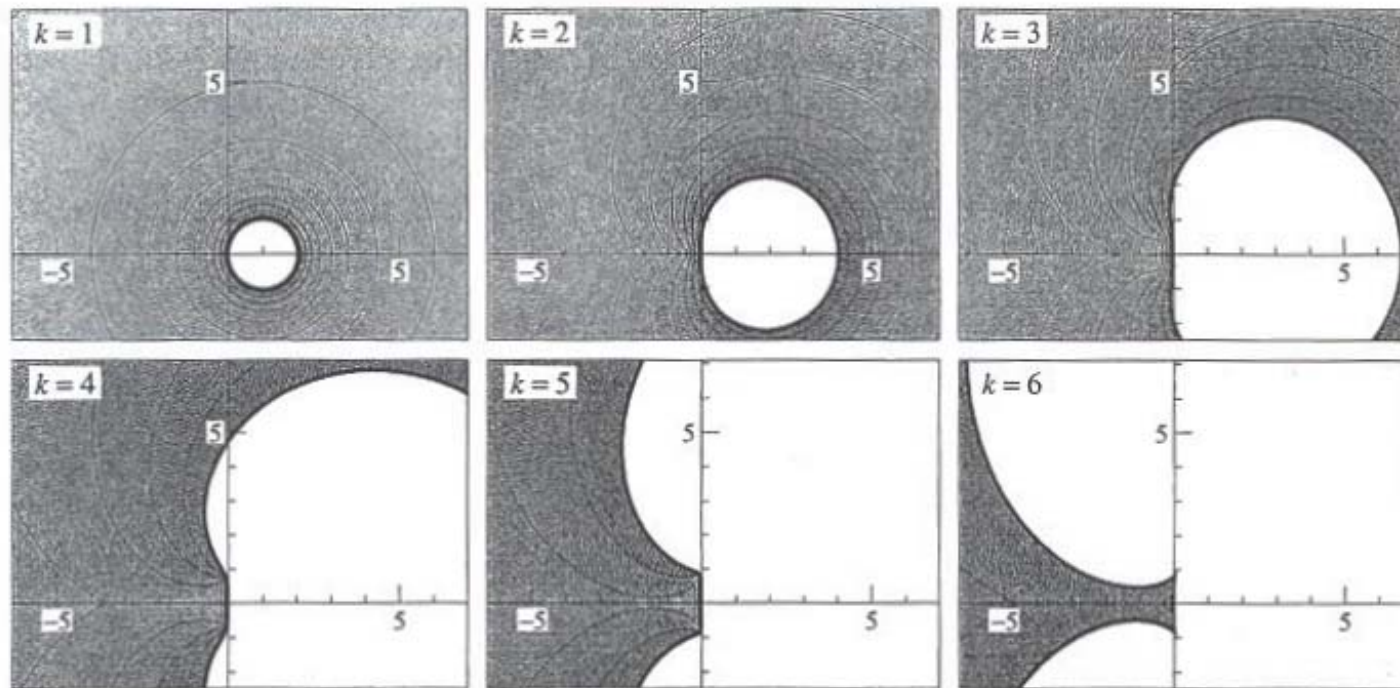


Fig. 1.3. Stability domains of implicit Adams methods, compared to those of the explicit ones

# Stability Region for BDF – Methods (i.e. DASSL)



**Fig. 1.6.** Root locus curves and stability domains of BDF methods

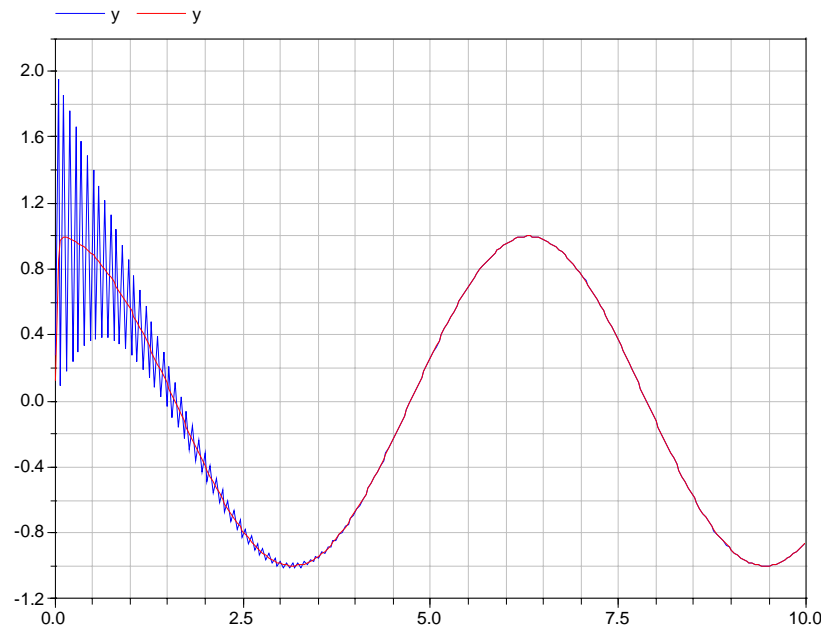
# Numerical Examples (Stability Region)

## ■ Curtiss & Hirschfelder

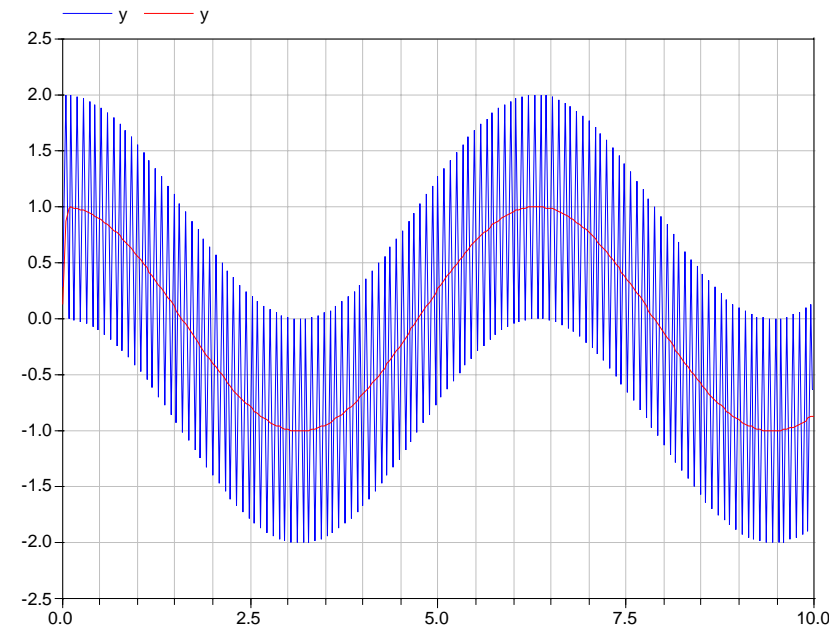
- Transition phase at the initial point
- Determine maximal fixed step-size

$$y' = -50(y - \cos(t)), \text{ eigenvalue } \lambda = -50$$

$\Delta t = 0.039$ : (explicit Euler)



$\Delta t = 0.04$ : (explicit Euler)



# Numerical Examples

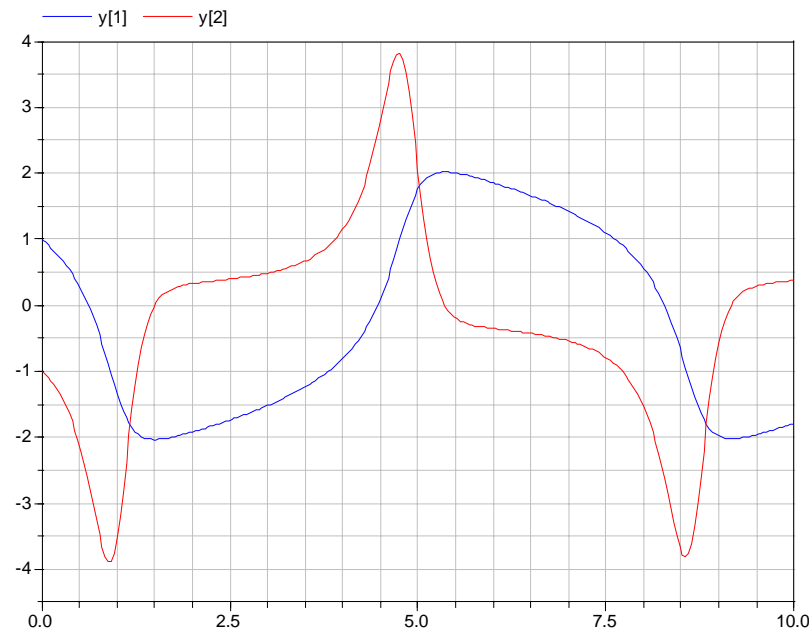
## (Stability Region and Stiffness)

### ■ Van der Pol

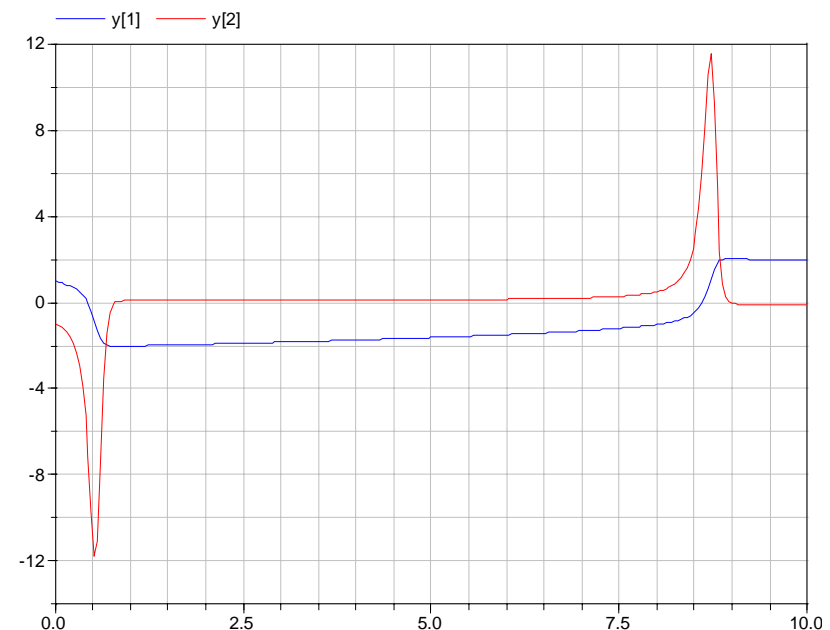
- Simple nonlinear equation describing electrical systems
- Stiff system, if  $\varepsilon \gg 1$ .

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \varepsilon(1 - y_1^2)y_2 - y_1 \end{aligned} \quad \text{mit } J(t) = \begin{pmatrix} 0 & 1 \\ -2\varepsilon y_1 y_2 - 1 & \varepsilon(1 - y_1^2) \end{pmatrix}$$

$\varepsilon = 2$ :



$\varepsilon = 8$ :



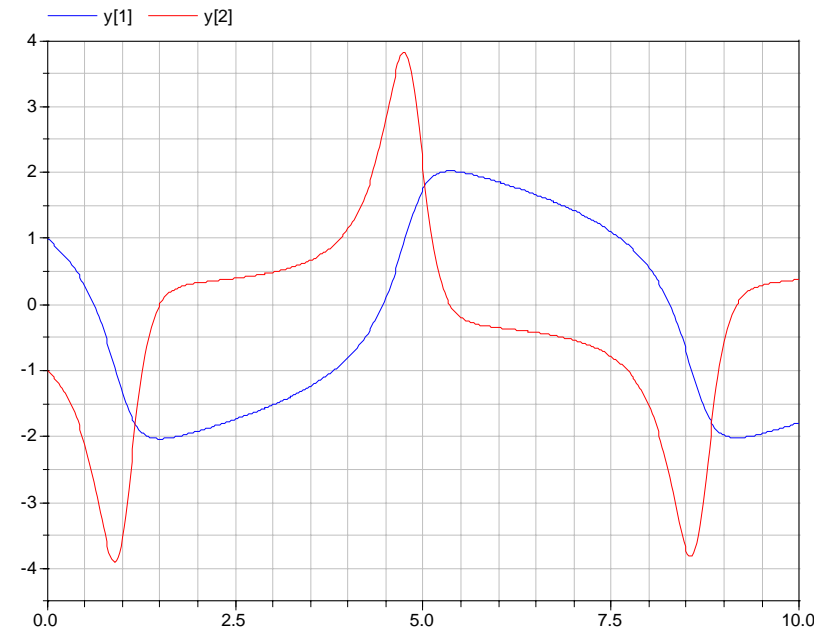
# Determine Stability Region (Van der Pol equation)

## ■ Linearize the system at initial time

- Calculate the Jacobians
- Read matrix into Matlab
- Determine the eigenvalues
- Calculate maximal step-size

$$\begin{aligned} \dot{\underline{x}}(t) &= \underline{h}(t, \underline{x}(t), \underline{u}(t), \underline{p}) \\ \underline{y}(t) &= \underline{k}(t, \underline{x}(t), \underline{u}(t), \underline{p}) \end{aligned} \longrightarrow \begin{aligned} \dot{\underline{x}}(t) &= A\underline{x}(t) + B\underline{u}(t) \\ \underline{y}(t) &= C\underline{x}(t) + D\underline{u}(t) \end{aligned}$$

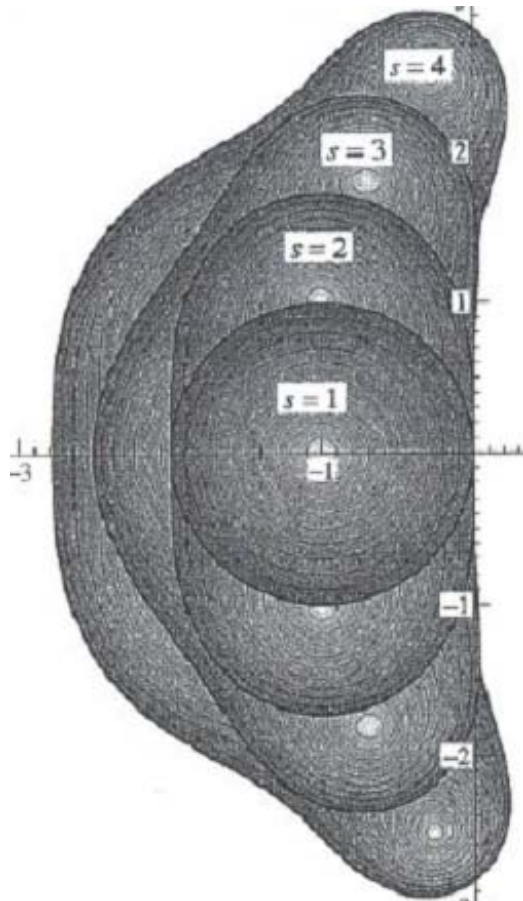
```
>> A = tloadlin
>c:\...\dslin.mat loaded.
A =      0      1.0000
    39.0000      0
>> l=eig(A)
l =    6.245
     -6.245
>> delta_t=2/6.245
delta_t = 0.3203
```



# Determine Stability Region (Van der Pol equation)

bounded solution, iff

$$|R(\Delta t \cdot \lambda)| \leq 1, \quad \lambda \text{ eigenvalue of } J$$



explicit Euler method:  $R(z) = 1 + z$

$$|R(\Delta t \cdot \lambda)| = |1 + \Delta t \cdot \lambda| = \left| \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \Delta t \begin{pmatrix} x \\ y \end{pmatrix} \right| \leq 1$$

$$\Leftrightarrow (1 + \Delta t \cdot x)^2 + (\Delta t \cdot y)^2 \leq 1$$

$$\Leftrightarrow \Delta t^2 (x^2 + y^2) + 2 \cdot \Delta t \cdot x \leq 0$$

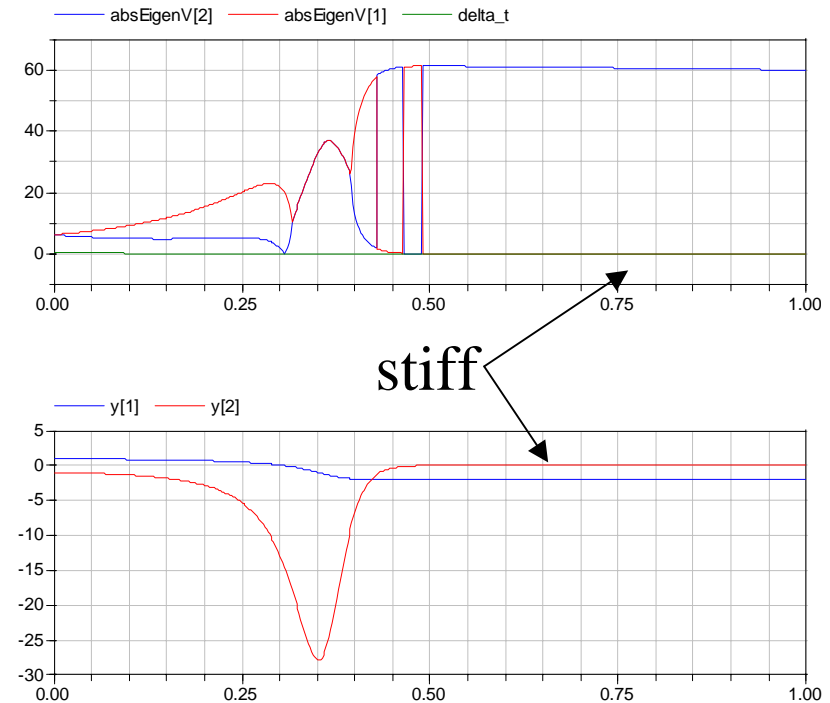
$$\Leftrightarrow \Delta t \leq -\frac{2 \cdot x}{x^2 + y^2}$$

# Determine Stability Region (Van der Pol equation)

- Calculate eigenvalues during simulation

```

model VanDerPol
  import Matrices.eigenValues;
  import Vectors.norm;
  parameter Real epsilon=2;
  Real y[2] (start={1,-1});
  Real eigenV[2,2];
  Real delta_t;
equation
  der(y[1]) = y[2];
  der(y[2]) = epsilon*(1 - y[1]*y[1])*y[2] - y[1];
  eigenV=eigenValues(
    [
      0, 1;
      -2*epsilon*y[1]*y[2]-1, epsilon*(1-y[1]*y[1])]);
  delta_t=max(-2*eigenV[1,1]/(eigenV[1,:]*eigenV[1,:]),
    -2*eigenV[2,1]/(eigenV[2,:]*eigenV[2,:]));
end VanDerPol;
  
```



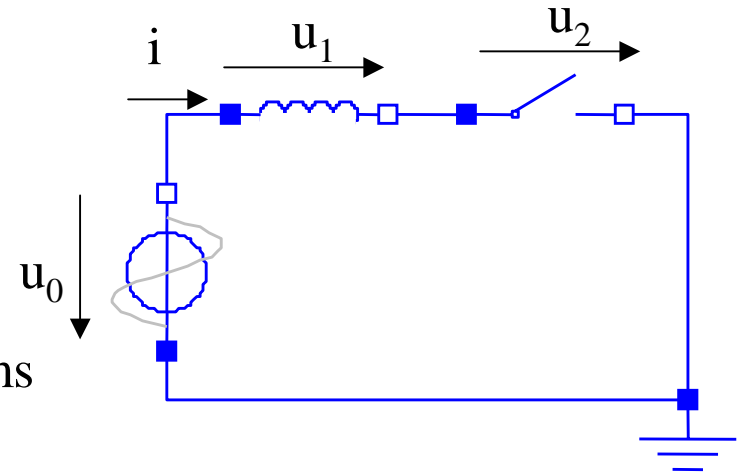
$$\Longrightarrow \Delta t_{\max} = 0.0326524$$

# Event Handling

## Motivation and Numerical Issues

- Need of discontinuous components

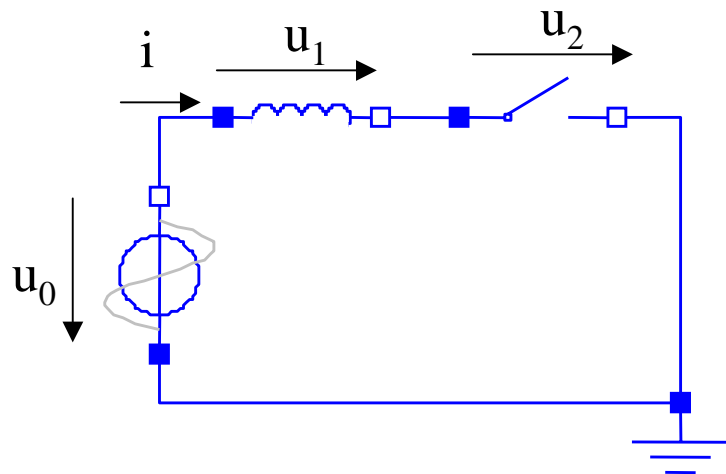
- Detailed modeling too complex
- Model parameters are not known for detailed models
- Non-ideal approximation leads to stiff systems
- Speed up simulation time (omit stiffness)



- Numerical problems at events

- ODE solver based on polynomial approximation
- Usually some signals are not continuous or differentiable
- Fixed step size leads to bad approximation of events
- Varying higher-index problems can occur at events

# Example: Non-Ideal Switch Stiffness Problem



$$u_0 = A \sin(\omega t) \longrightarrow L \frac{di}{dt} = A \sin(\omega t) - R i$$

$$u_2 = R i$$

$$L \frac{di}{dt} = u_1$$

$$u_0 = u_1 + u_2$$

$$\lambda = -\frac{R}{L}$$

Non-ideal switch:

closed:  $u_2 = R i$ ,  $R=1e-5$

open:  $G i = u_2$ ,  $G=1e-5$

$$u_0 = A \sin(\omega t) \longrightarrow L \frac{di}{dt} = A \sin(\omega t) - \frac{i}{G}$$

$$i = G u_2$$

$$L \frac{di}{dt} = u_1$$

$$u_0 = u_1 + u_2$$

$$\lambda = -\frac{1}{L \cdot G}$$



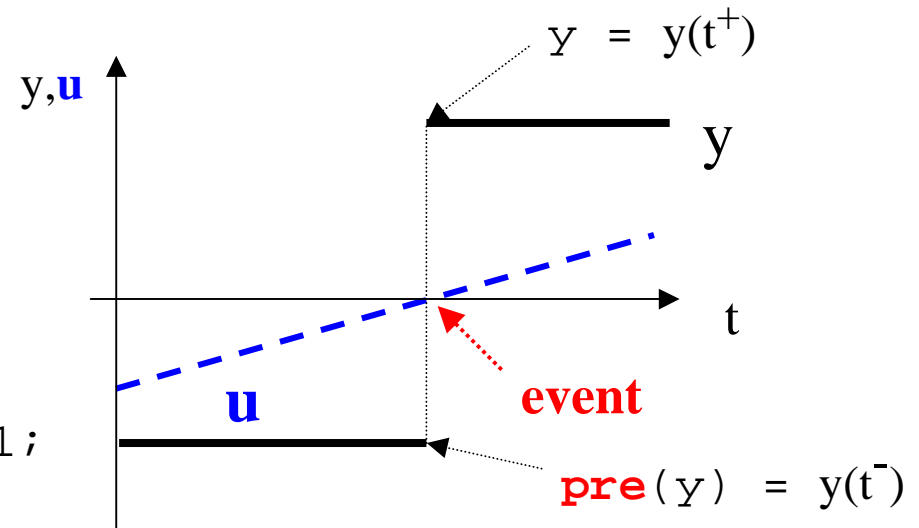
If the switch is open, system is stiff!

# Continuous Variables and the if-Statement

## ■ State events

- zero-crossing function
- i.e. bi-section method between adjacent time points

```
y = if u>0 then 1 else -1;
```



## ■ Time events

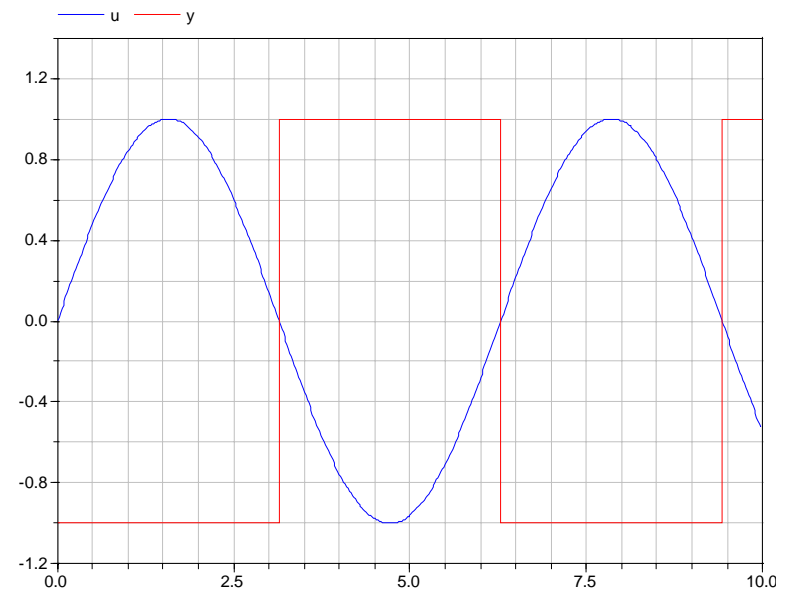
- no zero-crossing function necessary
- special treatment during compile time possible

```
y = if time>.5 then 1 else -1;
```

# Principle Strategy at Events (Simple Example)

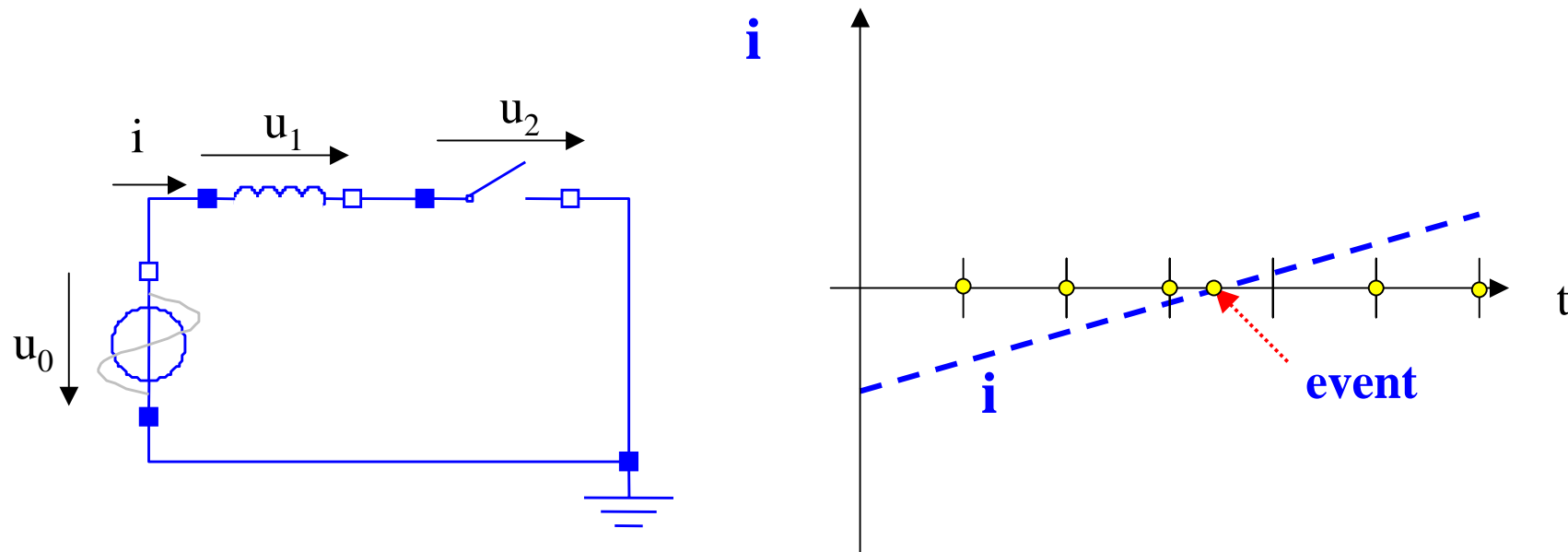
- **Standard-solution method**
  - Stop integration at event time
  - Do necessary adjustments
    - change input signals and/or discrete variables, re-initialize states
  - Restart integration routine
    - start-up of multi-step methods, initial step-size
- **Continuous time integration between events**
  - Appropriate step size control
- **Store signals twice at event time**
  - Variables before and after the event

```
model StateEvent
  import Modelica.Math.sin;
  Real u;
  Real y;
equation
  u = sin(time);
  y = if u<0 then 1 else -1;
end StateEvent
```



# Real-Time Solution to Events

- One-step method with fixed step-size
  - Detect event ( $u$  crosses 0)
  - Approximate exact event time
    - use (linear) polynomial approximation of the signal
  - Restart integration routine
    - combine two equidistant steps



# Influence Event-Handling

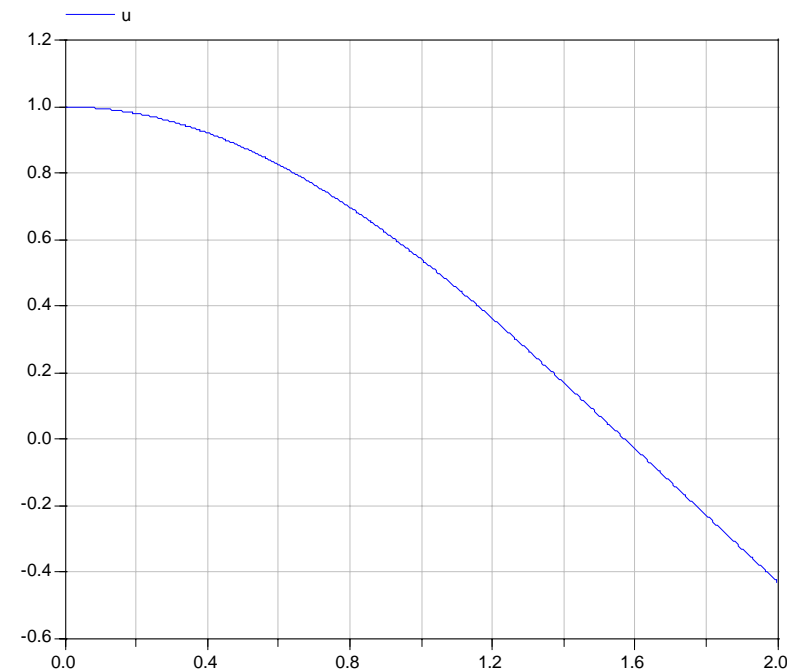
```

model SmoothEvent
  constant Real pi=Modelica.Constants.pi;
  Real x,z,y,u;
equation
  x = if time<pi/2 then cos(time) else pi/2-time;
  y = noEvent( if time<pi/2 then cos(time) else pi/2-time);
  z = smooth(1,if time<pi/2 then cos(time) else pi/2-time);
  u = smooth(1, noEvent(if time<pi/2 then cos(time) else pi/2-time));
end SmoothEvent

```

## Semantical interpretation:

- Equation for x: triggers event
- Equation for y: no event handling
- Equation for z: may trigger event
  - may depend on the integration method
- Equation for u: no event handling



# Sorting Equations Including if-Equations

## ■ Modelica model:

- Differential equations, algebraic equations

$$\underline{0} = \underline{f} \left( t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p} \right)$$

- if-equations

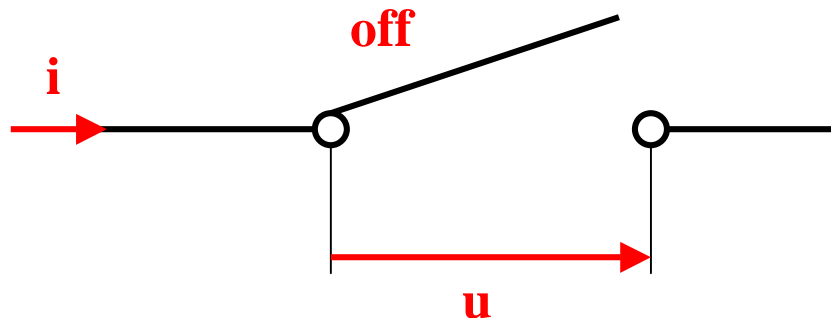
- $0 = \text{if } \langle \text{cond} \rangle \text{ then } \langle \text{expr1} \rangle \text{ else } \langle \text{expr2} \rangle ;$

## ■ Mathematical view:

- $\langle \text{cond} \rangle c$  is a boolean expression
  - may depend on a state or time event (Causality)
- An if-equation is treated as one equation depending on  $\langle \text{cond} \rangle c$  and the variables  $w_1, \dots, w_n$  in  $\langle \text{expr1} \rangle$  and  $v_1, \dots, v_m$  in  $\langle \text{expr2} \rangle$

$$0 = g \left( c, w_1, \dots, w_n, v_1, \dots, v_m, \underline{p} \right)$$

# Example: Ideal Electrical Switch



The ideal switch is described by the following two equations:

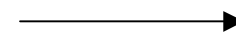
$$\begin{aligned} \text{off} = \text{true}: & \quad \mathbf{i} = \mathbf{0} \\ \text{off} = \text{false}: & \quad \mathbf{u} = \mathbf{0} \end{aligned}$$

Corresponding Modelica code:

```
0 = if off then i else u;
```

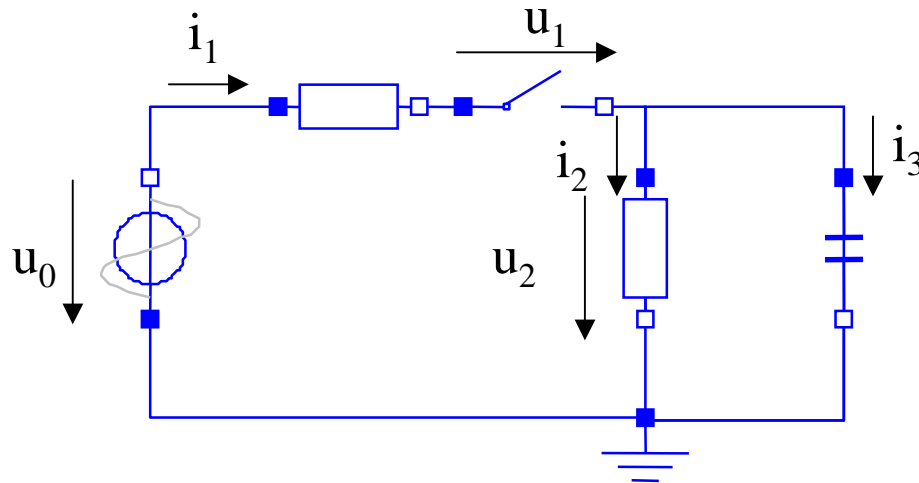
Equivalent description (**off = 0/1**):

$$0 = \text{off} \, i + (1 - \text{off}) \, u$$



$$0 = f(i, u, \text{off})$$

# Example: Ideal Electrical Switch



$$u_0 = A \sin(\omega t)$$

$$u_0 = R_1 \cdot i_1 + u_1 + u_2$$

$$0 = \text{off} \cdot i_1 + (1 - \text{off}) \cdot u_1$$

$$i_2 = R_2 / u_2$$

$$i_3 = i_1 - i_2$$

$$C \dot{u}_2 = i_3$$

Linear system of two equations:

$$\begin{aligned} u_0 &= R_1 \cdot i_1 + u_1 + u_2 \\ 0 &= \text{off} \cdot i_1 + (1 - \text{off}) \cdot u_1 \end{aligned} \quad \longrightarrow \quad \begin{bmatrix} R_1 & 1 \\ \text{off} & (1 - \text{off}) \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ u_1 \end{bmatrix} = \begin{bmatrix} u_0 - u_2 \\ 0 \end{bmatrix}$$

Two possible solutions:

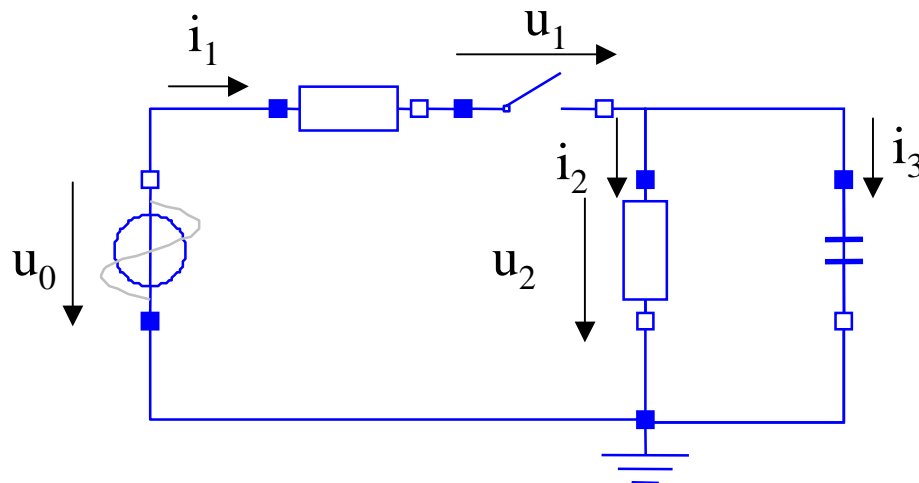
$$\text{off} = 1 : \quad i_1 = 0$$

$$u_1 = u_0 - u_2$$

$$\text{off} = 0 : \quad u_1 = 0$$

$$i_1 = (u_0 - u_2) / R_1$$

# Example: Ideal Electrical Switch



$$u_0 = A \sin(\omega t)$$

$$u_0 = R_1 \cdot i_1 + u_1 + u_2$$

$$0 = \text{off} \cdot i_1 + (1 - \text{off}) \cdot u_1$$

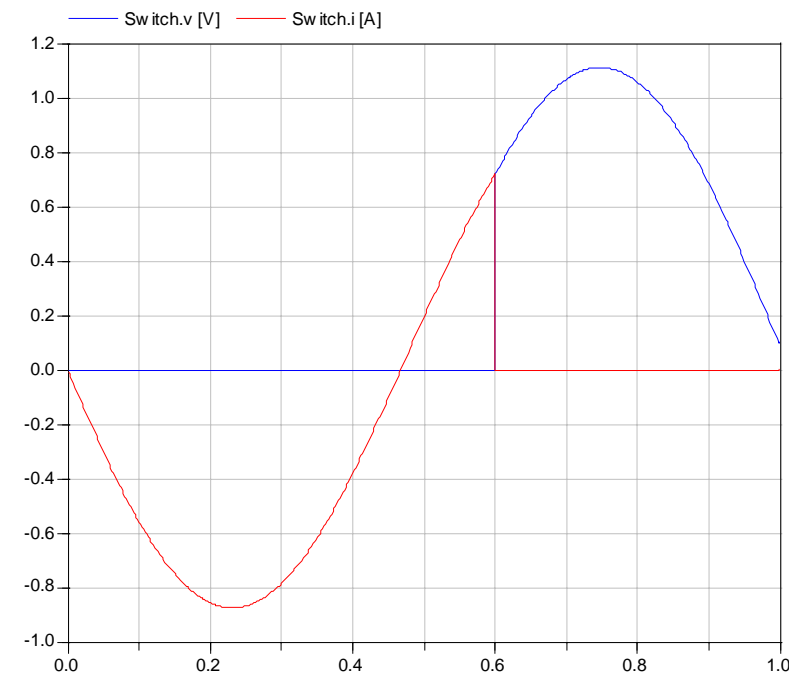
$$i_2 = R_2 / u_2$$

$$i_3 = i_1 - i_2$$

$$C \dot{u}_2 = i_3$$

## Unrealistic behavior :

- The current can not jump to zero
- Usually an arc keeps current flowing
- Current must be close to zero
- Additional logic necessary
  - needs **when**-equations  
(example will be continued)



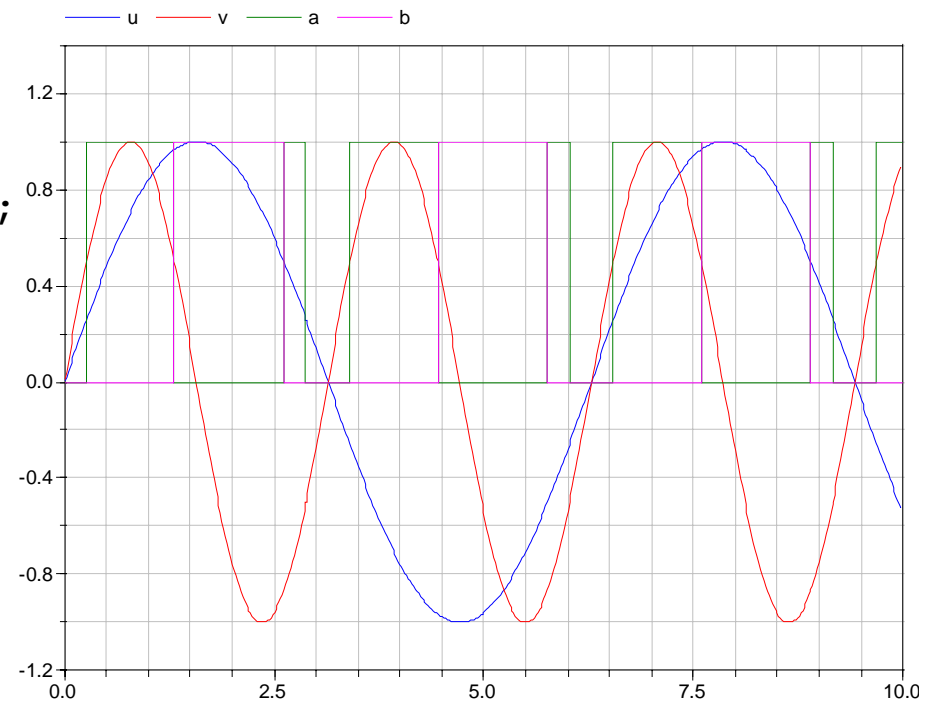
# Algebraic Loops Involving Integer and/or Boolean Variables

```

model AlgebraicLoopBoolean
  Boolean a(start=true);
  Boolean b;
  Real u;
  Real v;
equation
  u = sin(time);
  v = sin(2*time);
  b = not pre(a) and abs(u)>0.5;
  a = not b and abs(v)>0.5;
end AlgebraicLoopBoolean
  
```

Algebraic loops involving boolean variables must be handled by the user!

- Break loops using the **pre** operator



# Discrete Variables and the when-Statement

Additional equations can be declared at an event using the **when**-statement. These equations are de-activated during the continuous integration.

```
when   <condition>   then
    <equations>
end when;
```

When <condition> becomes true, <equations> are calculated.

Equivalent formulation:

```
if edge(<condition>)   then
    <equations>
end if;
```

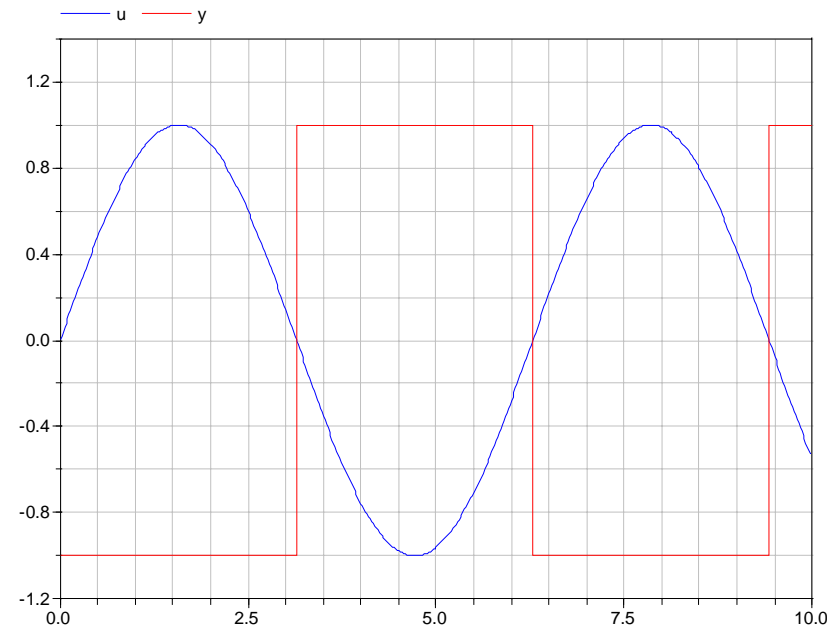
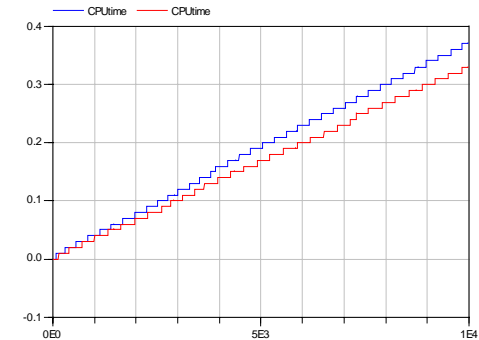
<condition> may not depend on a **noEvent**( ) Operator. Therefore, only at event points the equations within the when clause are evaluated

# Re-writing State Events to Time Events

```

model HandleEvents
  constant Real PI=Modelica.Constants.pi;
  Real u;
  Real y;
  Boolean flag(start=false);
equation
  u = Modelica.Math.sin(time);
  when sample(PI,PI) then
    flag = not pre(flag);
  end when;
  y = if flag then 1 else -1;
  //y = if u<0 then 1 else -1;
end HandleEvents

```



## Real-time efficiency:

- If possible use time-events instead of state-events

# Sorting Equations Including when-Statements

- Modelica model:

- Differential equations, algebraic equations, and if-equations

$$\underline{0} = \underline{f} \left( \underline{t}, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p} \right)$$

- Discrete equations only active at events

- **when** <cond> **then** <equations> **end when**;

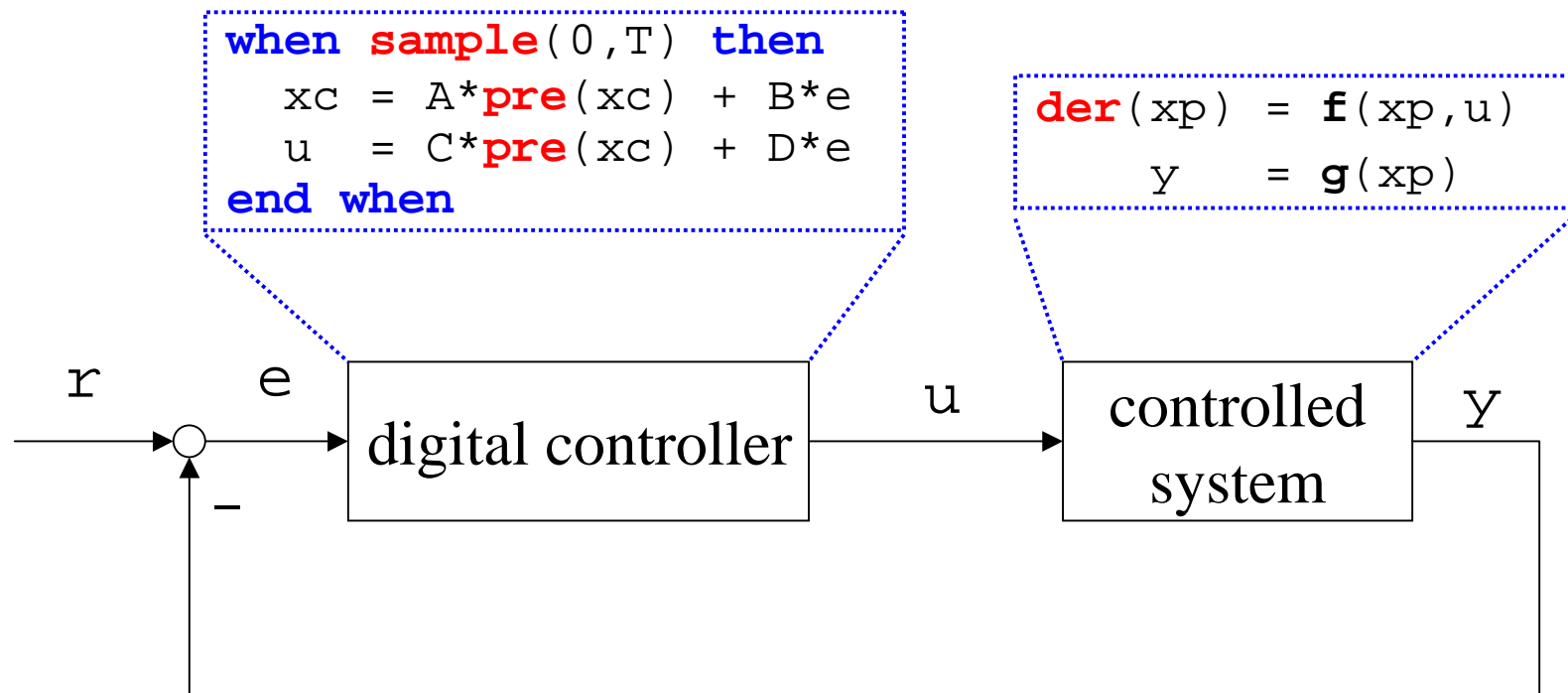
- Mathematical view:

- Same number of equations and unknowns at each time point
- Different number of equations and unknowns during event

- Algorithmic strategy:

- Sorting is based on all equations

# Example: Digital Controller



```
y := g(xp);
e := r - y;
when sample(0,T) then
  xc := A*pre(xc) + B*e;
  y  := C*pre(xc) + D*e;
end when
der(xp) := f(xp,u);
```

Order of evaluation is correct for the sample time points as well as during continued time integration!

(xp and **pre**(xc) are known)

# Sorting Equations of Hybrid Models

- **IMPORTANT:**

Each discrete variable is determined by one equation

- Example:

```
when h1>0 then
    openValve = true;
end when;
```

```
when h2<2 then
    openValve = false;
end when;
```

- Not allowed since two equations for one variable openValve
- Combine conditions  $h1>0$  and  $h2<2$

# Sorting Equations of Hybrid Models

## Re-writing event equations

```

when h1>0 or h2<2 then
  openValve = if edge(h1 > 0) then true else false;
end when;

```

## Semantical interpretation:

If one condition becomes true, (i.e.  $h1 > 0$ ), no further event will happen not even when the second condition ( $h2 < 2$ ) becomes true.

Boolean expression ( $h1 > 0$  or  $h2 < 2$ ) will not change anymore!

This is not the desired logic!

# Sorting Equations of Hybrid Models

Desired logic can be described by

```
when {h1>0, h2<2} then
  openValve = if edge(h1 > 0) then true else false;
end when;
```

or even shorter:

```
when {h1>0, h2<2} then
  openValve = edge(h1 > 0);
end when;
```

```
when {expr1, expr2, expr3, ...} then
```

Semantical interpretation:

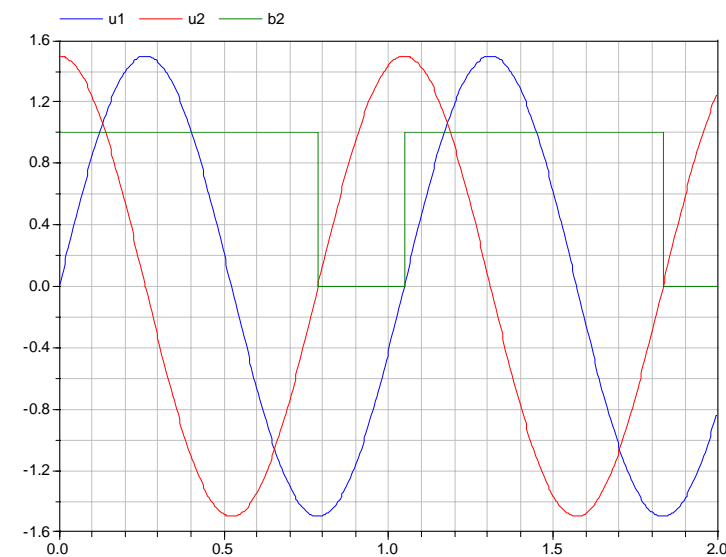
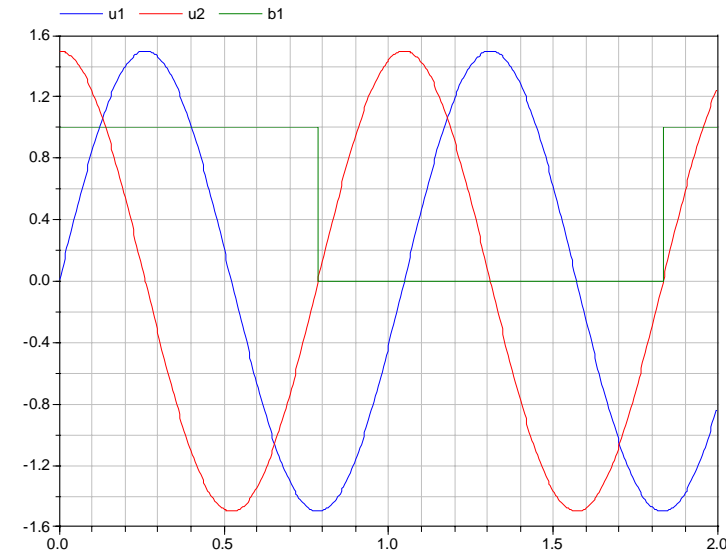
- Triggers an event, if one of the listed boolean expression becomes true

# Example: Difference of „or“ and {...,...}

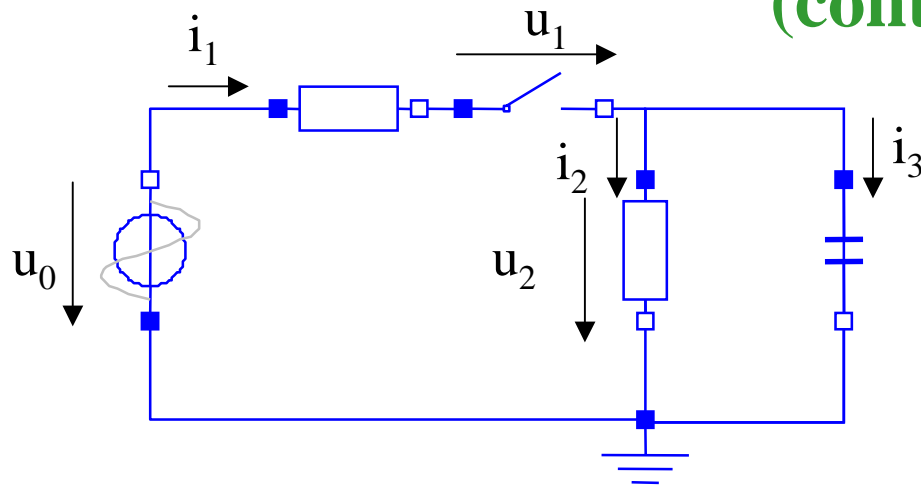
```

model WhenDemo
  parameter Real A=1.5, w=6;
  Real u1, u2;
  Boolean b1, b2;
equation
  u1 = A*sin(w*time) + 1.e-10;
  u2 = A*cos(w*time);
  when u1 > 0 or u2 > 0 then
    b1 = not pre(b1);
  end when;
  when {u1 > 0, u2 > 0} then
    b2 = not pre(b2);
  end when;
end WhenDemo;

```



# Example: Ideal Electrical Switch (continued)



$$u_0 = A \sin(\omega t)$$

$$u_0 = R_1 \cdot i_1 + u_1 + u_2$$

$$0 = \text{off} \cdot i_1 + (1 - \text{off}) \cdot u_1$$

$$i_2 = R_2 / u_2$$

$$i_3 = i_1 - i_2$$

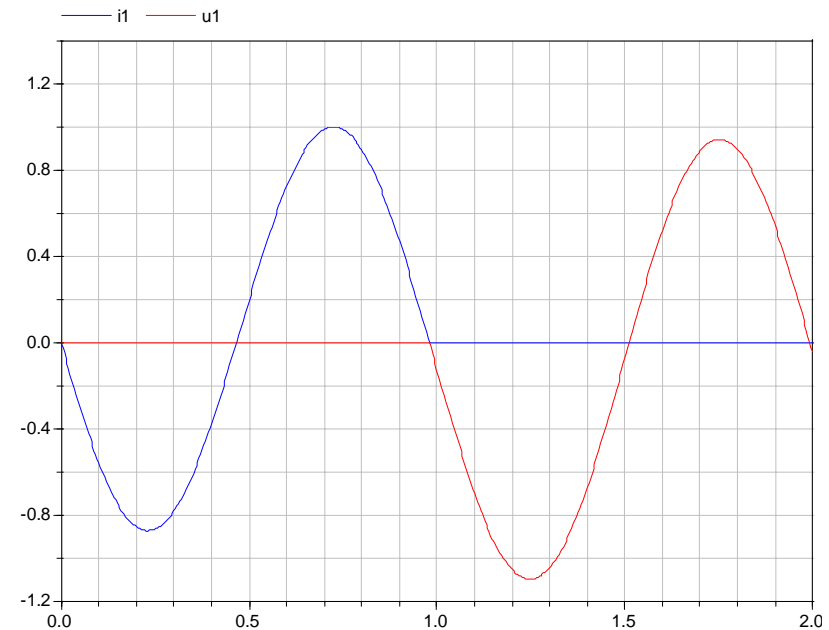
$$C \dot{u}_2 = i_3$$

Introduce additional logic :

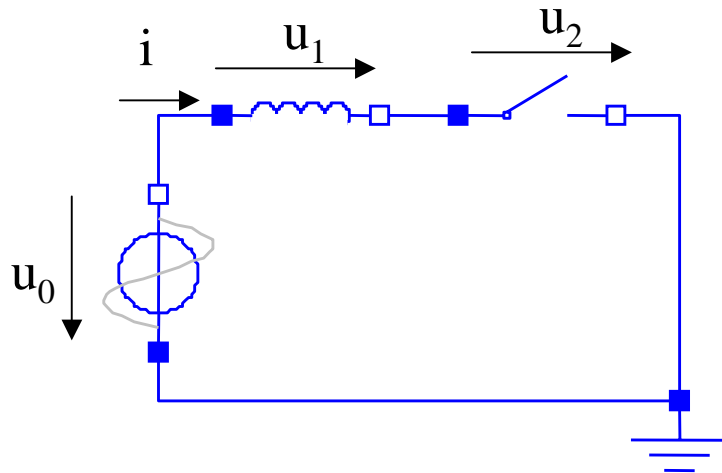
- Algebraic loop with boolean variable

```

when (time>t0) then
  sign_i = if (i1>0) then 1 else -1;
end when;
when (time>t0 and sign_i*i1<0) then
  off = true;
end when;
0 = if pre(off) then i1 else u1;
    
```



# Example: Ideal Electrical Switch (continued)



$$u_0 = A \sin(\omega t)$$

$$0 = \text{off} \cdot i + (1 - \text{off}) \cdot u_2$$

$$L \frac{di}{dt} = u_1$$

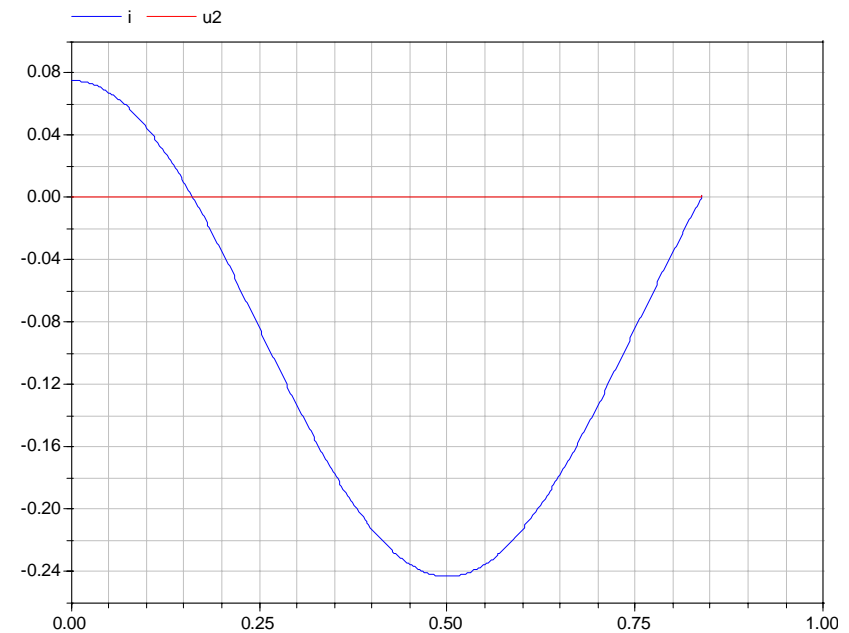
$$u_0 = u_1 + u_2$$

Causality ( $\text{off} = 0/1$ ):

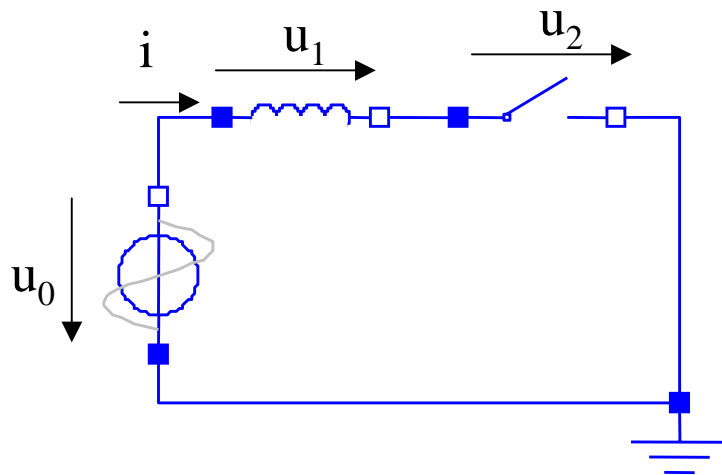
- $i$  is a state and is therefore known

$$u_2 = \frac{-\text{off} \cdot i}{1 - \text{off}}$$

Only solvable, iff  $\text{off}=0$



# Example: Ideal Electrical Switch (continued)



$$u_0 = A \sin(\omega t)$$

$$0 = \text{off} \cdot \frac{di}{dt} + (1 - \text{off}) \cdot u_2$$

$$L \frac{di}{dt} = u_1$$

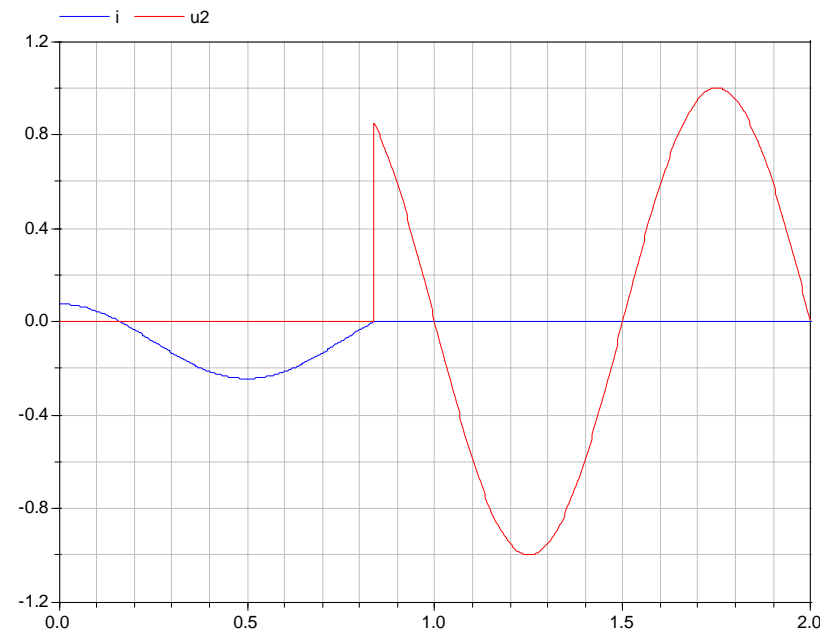
$$u_0 = u_1 + u_2$$

Varying higher-index problem

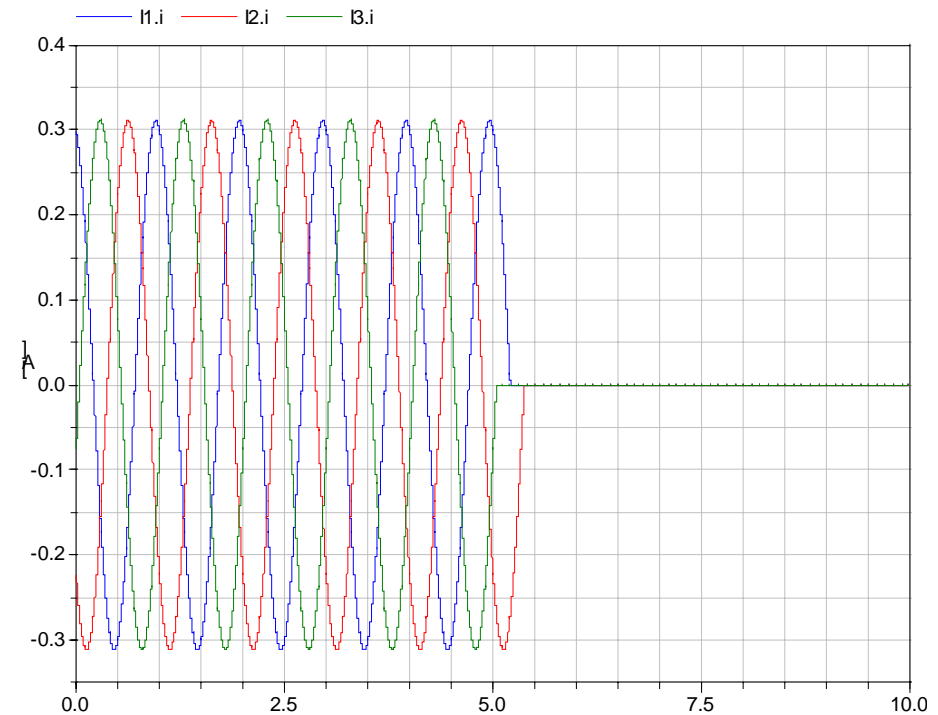
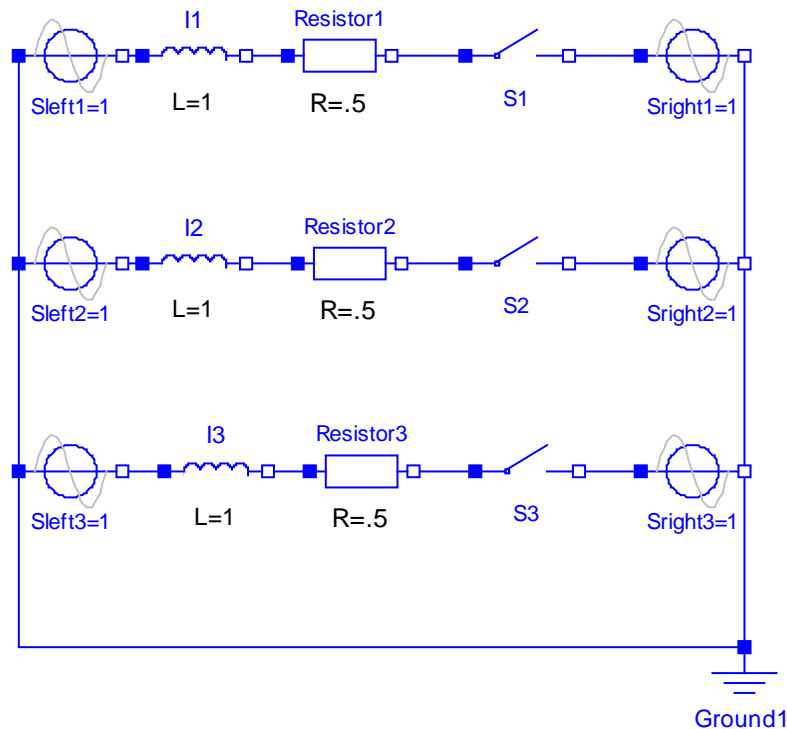
$$\text{off} = 0 : u_2 = 0 \quad \text{index}=1$$

$$\text{off} = 1 : i = 0 \quad \text{index}=2$$

$$\frac{di}{dt} = 0$$



# Example: 3-Phase Electrical System (continued)

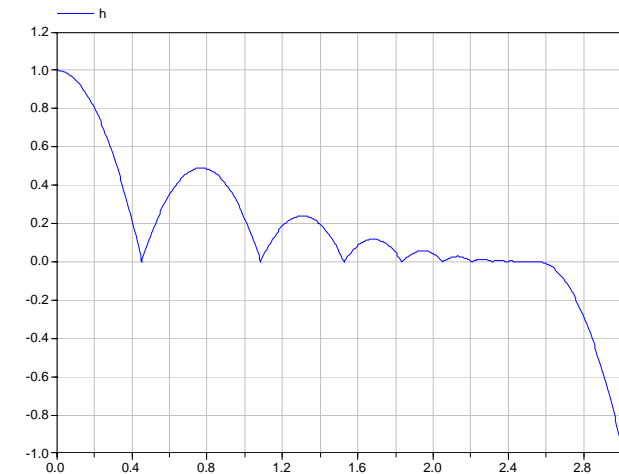
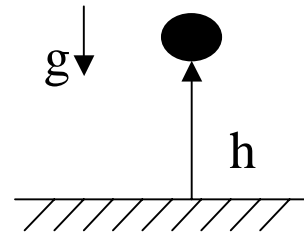


## Switch works also for the 3-Phase electrical system

- Introducing dq0-reference frame still very efficient
- $i_{dq0}$  used as state
- Initialization: **der** ( $i_{dq0}$ ) = { 0 , 0 , 0 } ;

# Numerical Issues

- Standard-solution method
  - Stop integration at event time
  - Do necessary adjustments
  - Restart integration routine
- Due to numerical errors the system can get into a non-physical state
  - Height  $h$  is negative in the bouncing ball example
- Solution to this problem
  - Changes in the settings (accuracy) for the solver routine may help
  - Different integration methods may lead to different results
  - Adapt logic to circumvent numerical problems (**Most Efficient**)

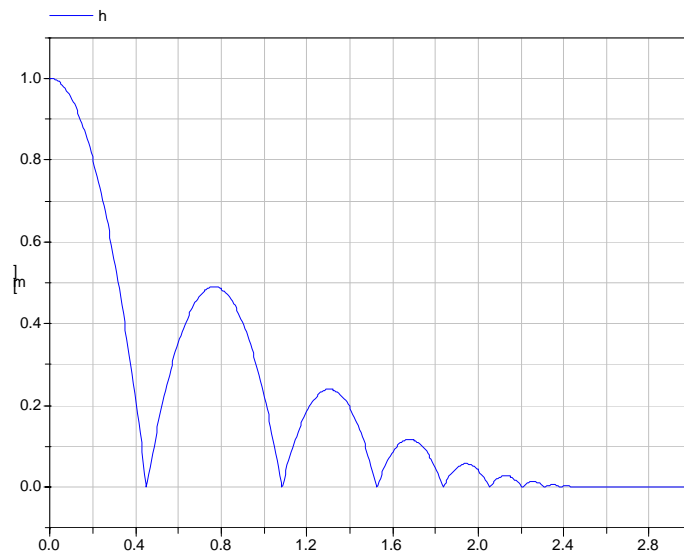
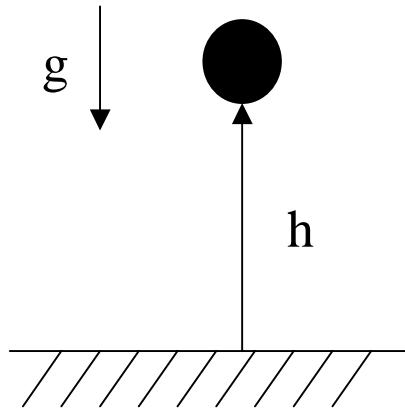


```

model BouncingBall
  parameter Real e=0.7;
  parameter Real g=9.81;
  Real h(start=1);
  Real v;
equation
  der(h) = v;
  der(v) = -g;
  when h < 0 then
    reinit(v, -e*pre(v));
  end when;
end BouncingBall;
  
```

# Numerical Issues

## Solution to the Bouncing Ball



```

model BouncingBall
  parameter Real e=0.7;
  parameter Real g=9.81;
  Real h(start=1);
  Real v;
  Boolean flying(start=true)
  Boolean impact;
  Real v_new;
equation
  der(h) = v;
  der(v) = if flying then -g else 0;
  impact = h < 0;
  when {impact, h < 0 and v < 0} then
    v_new = if edge(impact) then -e*pre(v)
            else 0;

    flying = v_new > 0;
    reinit(v, v_new);
  end when;
end BouncingBall;

```