Start Tuesday Sept 29, 2009 9.15-17.00

Course at Linköping University, Room: John von Neumann
Location: Building B, 2nd floor

Principles of Object-Oriented Modeling and Simulation of Dynamic Systems with Modelica

Hands-on exercises using OpenModelica—Bring Laptop!

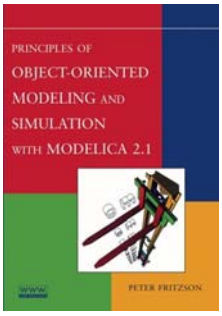
The course has the following goals

- Being easily accessible for people who do not previously have a background in modeling and simulation.
- Introducing the concepts of physical modeling, object-oriented modeling and component-based modeling and simulation.
- Demonstrating modeling examples from several application areas.
- Providing opportunity for hands-on exercises with the OpenModelica open-source implementation of Modelica and the simForge graphic user interface

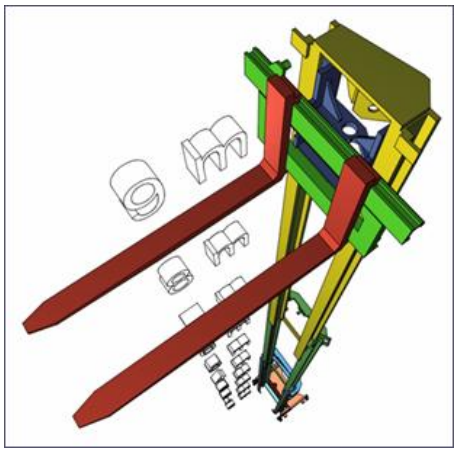
The tutorial is based on Peter Fritzson's book:

Principles of Object-Oriented Modeling and Simulation with Modelica 2.1

ISBN: 0-471-47163-1
Paperback, 940 pages
February 2004, Wiley-IEEE Press



Course Content



Object-Oriented modeling is a fast-growing area of modeling and simulation that provides a structured, computer-supported way of doing mathematical and equation-based modeling. Modelica is today the most promising modeling and simulation language in that it effectively unifies and generalizes previous object-oriented modeling languages and provides a sound basis for the basic concepts. The Modelica modeling language and technology is being warmly received by the world community in modeling and simulation with major applications in virtual prototyping. It is bringing about a revolution in this area, based on its ease of use, visual design of models with combination of lego-like predefined model building blocks, its ability to define model libraries with reusable components, its support for modeling and simulation of complex applications involving parts from several application domains, and many more useful facilities.

The course presents an object-oriented component-based approach to computer supported mathematical modeling and simulation through the powerful Modelica language and its associated technology. Modelica can be viewed as an almost universal approach to high level computational modeling and simulation, by being able to represent a range of application areas and providing general notation as well as powerful abstractions and efficient implementations.

The course gives an introduction to the Modelica language to people who are familiar with basic programming concepts. It gives a basic introduction to the concepts of modeling and simulation, as well as the basics of object-oriented component-based modeling for the novice, an overview of modeling and simulation in a number of application areas, and an introduction to meta modeling. The OpenModelica environment together with the simForge graphical user interface and MathModelica will be used for hands-on exercises.

Lecturers

Peter Fritzson is Professor and Research Director of the Programming Environment Lab (PELAB), at the Department of Computer and Information Science, Linköping University, Sweden. He has been chairman of the Scandinavian Simulation Society, secretary of the European simulation organization, EuroSim. He is vice chairman of the Modelica Association, and Director of the Open Source Modelica consortium. His main area of interest is software engineering, especially languages, programming and maintenance tools and environments, including modeling and simulation. Professor Fritzson has more than 200 publications including 10 books/proceedings.

Mohsen Torabzadeh-Tari is researcher at PELAB, with PhD from KTH. Main interest modeling and simulation with tools and applications.

Useful Links

The OpenModelica project website:
www.openmodelica.org

Peter Fritzson's book:

Principles of Object-Oriented Modeling and Simulation with Modelica 2.1
http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471471631.html
also:
www.ida.liu.se/labs/pelab/modelica/OpenModelica/Documents/ModelicaBookExcerpts.pdf
also: www.mathcore.com/drmodelica.

Graphic user interface:
https://trac.elet.polimi.it/simforge/

Schedule and Reading Instructions
Modelica Course at Linköping University

September-October 2009

Peter Fritzson
Mohsen Torabzadeh-Tari

Schedule

Day1: Tuesday, Sept 29, 9.15-17.00

Day2: Wednesday, Sept 30, 12.30-18.00 (hint: Spend morning reading the course book)

Day3: Tuesday, October 13, 9.15-17

Day4: Thursday, October 15, 9.15-17

Day5: Thursday, October, 22,.15-17

Approximately 28 hours including hands-on exercises.

Day1:

Lecture: Introduction to Modeling and Simulation with Modelica and OpenModelica

- OpenModelica OMNotebook usage

  Introduction to textual modeling

  Demo+Exercise: OMNotebook and DrModelica

  Demo+short exercise: Graphic modeling with simForge

Lecture+Exercises: classes and inheritance
Exercise01-classes-simple-textual.onb

Lecture+Exercises: Component connectors and connections, graphical modeling
Exercise02-graphical-modeling.onb

Day2:

Lecture:Equations
Exercise03-classes-textual-circuit.onb

Lecture: Algorithms and functions
Exercise04-equations-algorithms-functions.onb

Lecture: Modelica Packages

Lecture: Modelica Libraries

Day3:

Lecture: Hybrid Systems
Exercise05-hybrid-discreteevent.onb

Lecture:Simple biological models
Exercise06-pop-dynamics-and-model-design.onb

Lecture:Model Design
Exercise06-pop-dynamics-and-model-design.onb

Lecture: Romeo and Julia

Day4:

Lecture+Exercises: Building a simple Modelica library.

A whole day will be devoted to designing and building a simple modelica library from scratch, primarily using the graphical user interface.

Day5:

Lecture+Exercises:

- Introduction to the OpenModelica Eclipse plugin
- Simple simulation exercise using the Eclipse plugin.

Lecture+Exercises:

Introduction to MetaModelica

- Functional programming in MetaModelica
- Model transformations and symbolic programming
- Simple model transformation exercise in MetaModelica.

Lectures:

Introduction to the OpenModelica compiler

- Structure, information about modules, etc
- The model manipulation and information retrieval API.
- Corba connection to OMC

Advanced OpenModelica compiler development topics

- How to adapt code generator to specific needs,
- How to access the flat Modelica intermediate form,
- Programming AST transformations in the compiler
- How to add simple functionality to the compiler

Reading Instructions

The following are reading instructions for the course book Principles of Object Oriented Modeling and Simulation with Modelica 2.1.

You need to read this well enough to be able to sign a paper where you promise that you have read all the included at a level to understand approximately 95% of the included material.

There will be some sampled oral examinations to check this.

Included in the course:

Chapter 1, whole chapter.
Chapter 2, whole chapter.
Chapter 3: Sec 3.1 - 3.13.1, 3.13.3 - 3.14.7
Chapter 4, whole chapter.
Chapter 5: 5.1 - 5.4.0; 5.4.3 - 5.7.2; 5.8
Chapter 6: 6.1 - 6.8.0;
Chapter 7: 7.1 - 7.2.2
Chapter 8: 8.1 - 8.4.1.3
Chapter 9: 9.1 - 9.3.2.6
Chapter 10: whole chapter.
Chapter 11: not included.
Chapter 12: whole chapter.
Chapter 13: 13.1 - 13.2.5.5; 13.2.5.7 - 13.2.6.5; 13.3.0 - 13.3.4; 13.4.1; 13.5
Chapter 15: 15.1.0; 15.4.1; 15.5; 15.6.0-15.6.2; 15.7; 15.10.2-15.10.3
Chapter 17: 17.1.0, 17.1.4, 17.1.5, 17.1.6,
Chapter 18.1, 18.2.0, 18.2.1, 18.2.1.1

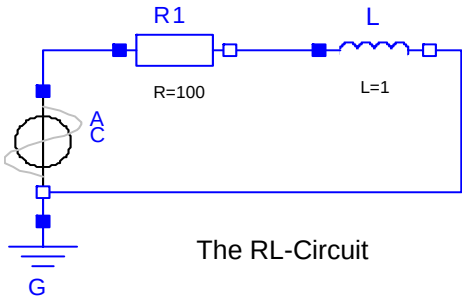
The following are reading instructions for included parts of the “Modelica Meta-Programming and Symbolic Transformations - MetaModelica programming guide”:

Chapter 1: whole chapter.
Chapter 2: Sec 2.0, 2.1, 2.2
Chapter 3: Sec 3.0, 3.1.5, 3.3

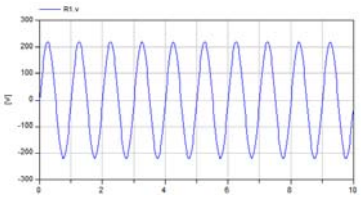
(We will have selected exercises, partly from the Appendix of the MetaModelica programming guide)

Exercises Part I – Basic Graphical Modeling

- (*See instructions on next two pages*)
- Start the simForge editor
- Draw the RL-Circuit
- Simulate



The RL-Circuit



Simulation

Exercises Part I – simForge Instructions Page 1

- Start simForge, (e.g. SimForge-0.8.4.1.jar).
- Go to **File** menu and choose **New Project**.
- Write *RL_Circuit* and click on the **Browse** button for choosing the destination folder.
- Press **OK.**
- In the navigation bar in the left, there should be three items, **Modelica, IEC61131-3** and **Simulation result**. Double-click on the **Modelica**.

• Under the **Modelica :**
  • The standard Modelica library components are listed in the **Used external package.**
  • The **Modelica classes** and **Modelica files** are the places where your models will end up under. The first folder is for the graphical models and the latter is for the texual form.

Exercises Part I – simForge Instructions Page 2

- Go to **File** menu and choose **New File.** Write *RL_circuit* and press **OK.**
- In the **Add Class** pop-up dialog box change the **Type** from **package** to **class** and press **OK.**
- Double click on the *RL_circuit* under the **Modelica classes** and the graphical window will appear.
- **Drag and Drop** components from the standard Modelica library to your model.
- For connecting components, move the cursor to the target pin and press shift+click ***once*** and just move the cursor with the mouse to the destination pin and press again shift+click.
- Start the simulation with **simulation** button.
- In the simulation pop-up you can leave out some fields like the **Stop time**, which will result in a default value of 1 sec. will be used.
- The result will appear under the **Simulation result.**

• Under the **Edit** menu -> **Advanced properties** you can tick the **visible legend** bar.

Exercises - Simple Textual

1 Simple Textual Modeling Exercises

1.1 Try DrModelica with VanDerPol

Locate the VanDerPol model in DrModelica (link from Section 2.1), run it, change it slightly, and re-run it.

1.2 HelloWorld

Simulate and plot the following example with one differential equation and one initial condition. Do a slight change in the model, re-simulate and re-plot

```
model HelloWorld "A simple equation"
  Real x(start=1);
equation
  der(x)= -x;
end HelloWorld;
```

Push shift-tab for command completion, fill in the name HelloWorld, and simulate it!

```
simul
```

Push shift-tab for command completion, fill in a variable name (x), and plot it!

```
plo
```

Take a look at the intepolated value of the variable x at time=0.5 using the val(variableName,time) function:

```
val(x,0.5)
```

Also take a look at the value at time=0.0:

```
val(x,0.)
```

1.3 A Simple Systems of Equations

Make a Modelica model that solves the equation system below with initial conditions. Hint: initial conditions are often specified using the start attribute.

$$\dot{x} = 2 * x * y - 3 * x$$
$$\dot{y} = 5 * y - 7 * x * y$$
$$x(0) = 2$$
$$y(0) = 3$$

```
model ...
```

1.4 Creating a Class

Create a class, `Multiply`, that calculates the product of two parameter variables, which are equal to `Real` numbers with given values.

1.5 Creating Instances

```
class Dog
  constant Real legs = 4;
  parameter String name = "Dummy";
end Dog;
```

Create an instance of the class `Dog` by declaring a variable.

Create another dog instance and give this dog the name "Tim".

2 Creating a Function and Making a Function Call

Write a function, `addTen`, that returns the input number plus the Integer 10.

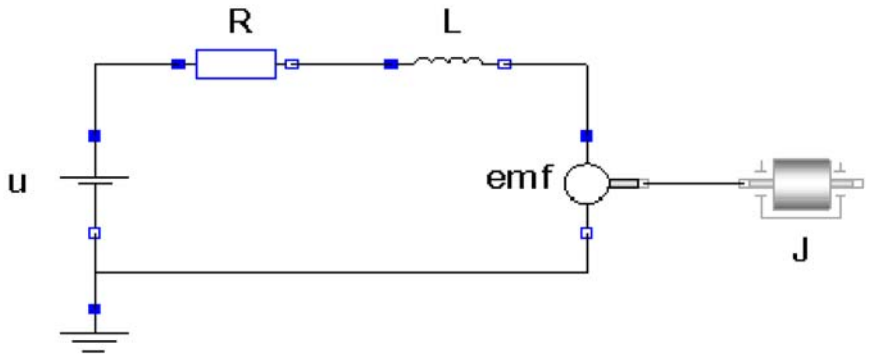
Make a function call to addTen with the input 3.5.
Also do that inside a class, and simulate the class.

Exercise - Graphical Modeling

1The DC Motor

A) DC Motor

Make a simple DC-motor using the Modelica standard library that has the following structure:




```
model ...
```

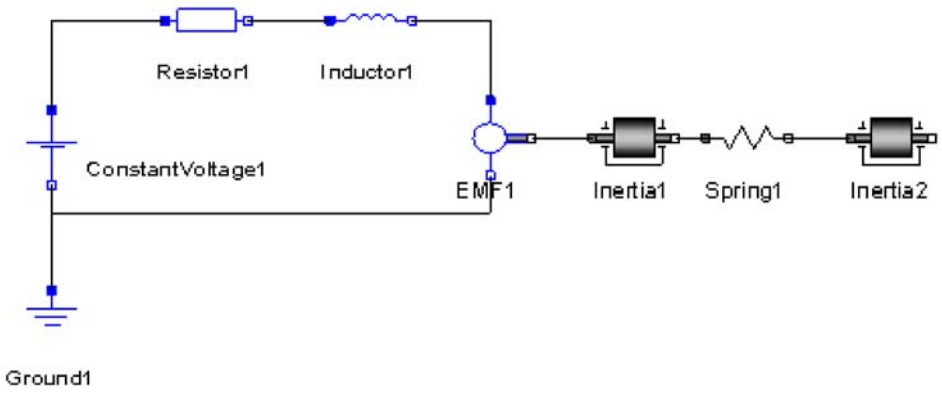
If you are using MathModelica Lite, first save the model from the graphic editor, load it (or alternatively copy paste it from the textual view in the graphic editor to OMNotebook or OMShell)

Simulate it (using OMShell or OMNotebook or MathModelica System Designer) for 15s and plot the variable the outgoing rotational speed on the inertia axis and the voltage on the voltage source (denoted u in the figure) the same plot.

Note: If you are using MathModelica System Designer you can do the plotting directly in the tool without cop the model into OMShell or OMNotebook.
Hint: if you have difficulty finding the names of the variables to plot, you can flatten the model by calling instantiateModel, which exposes all variable names

B) Spring and Inertia

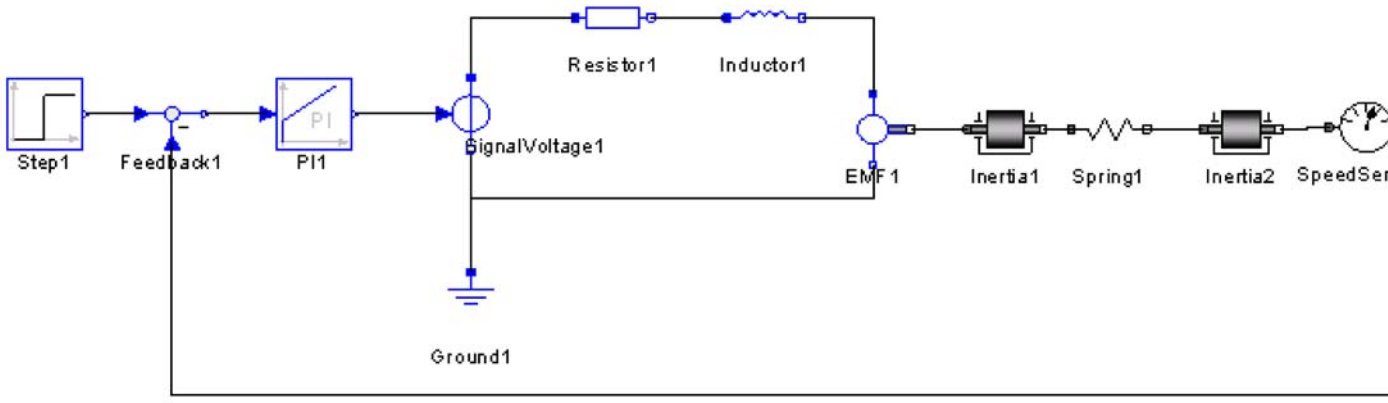
Add a torsional spring to the outgoing shaft and another inertia element. Simulate again and see the results. Ad
some parameters to make a rather stiff spring.




```
model ...
```

C) Adding controller

Add a PI controller to the system and try to control the rotational speed of the outgoing shaft. Verify the result
using a step signal for input. Tune the PI controller by changing its parameters in MathModelica System Desig
(or Lite).

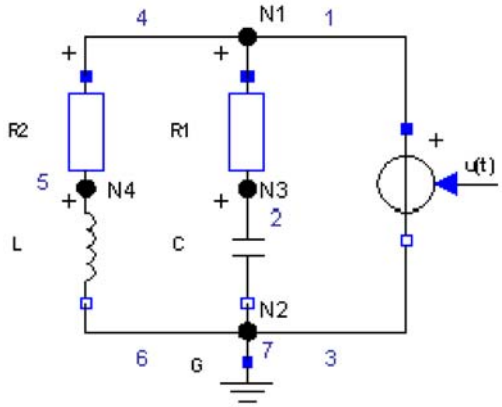



```
model ...
```

Exercise 2 - Classes and Simple Circuit

1 Add Components to SimpleCircuit

Add a capacitor between the R2 component and the R1 component and an inductor between the R1 and voltage component. Use the SimpleCicuit model and Modelica standard library components.




```
loadModel(Modelica);
```

```
model SimpleCircuit
  import Modelica.Electrical.Analog;
  Analog.Basic.Resistor  R1(R = 10);
  Analog.Basic.Capacitor C(C = 0.01);
  Analog.Basic.Resistor  R2(R = 100);
  Analog.Basic.Inductor  L(L = 0.1);
  Analog.Sources.SineVoltage AC(V = 220);
  Analog.Basic.Ground    G;
equation
  connect(AC.p, R1.p);
  connect(R1.n, C.p);
  connect(C.n, AC.n);
  connect(R1.p, R2.p);
  connect(R2.n, L.p);
  connect(L.n, C.n);
  connect(AC.n, G.p);
end SimpleCircuit;
```

You can verify the result in the model editor. This example illustrates how inconvenient it is to use textual modeling sometimes.

2 Build Electrical Components

This excercise consists of building a number of electrical componenets. Here follows the equations that describes each component. You can skip section 1.1 if you are familiar with the equations.

2.1 Equations

The ground element

$$v_p = 0$$

where v_p is the potantial of the ground element.

A resistor

$$i_p + i_n = 0$$

$$u = v_p - v_n$$

$$u = R\, i_p$$

where $i_p$and $i_n$represents the currents into the positive and negative pin (or port) of the resistor, $v_p$and v_n the corresponding potentials, u the voltage over the resistor, and R the resistance.

An inductor

$$i_p + i_n = 0$$

$$u = v_p - v_n$$

$$u = L\,\frac{di_p}{dt}$$

where $i_p$and $i_n$represents the currents into the positive and negative pin (or port) of the inductor, $v_p$and v_n the corresponding potentials, u the voltage over the inductor, and L the inductance.

A voltage source

$$i_p + i_n = 0$$

$$u = v_p - v_n$$

$$u = V$$

where $i_p$and $i_n$represents the currents into the positive and negative pin (or port) of the voltage source, $v_p$and v_n the corresponding potentials, u the voltage over the voltage source, and V the constant voltage.

Build Modelica models of the above components. A Connector component representing an electrical pin should be defined. Observe that the first two equations defining each electrical component with two pins above are equal. Utilize this observation to build a partial model, TwoPin, to be used in the definition of any electrical two pin component. Hence, totally six components (Pin, Ground, TwoPin, Resistor, Inductor, and a VoltageSource) should be built.

Use the defined components to build a model of a circuit diagram and simulate the behavior of the circuit.

2.2 Implementation

2.2.1 User-Defined Types

Define the types Voltage and Current.

```
type Voltage = Real;
type Current = Real;
```

2.2.2 Pin

The Pin has a potential, v and current variable, i. According to Kirchhoff's laws potentials are set equal and currents sum to zero at connections. Hence, v is an effort variable and i is a flow variable.

```
connector Pin
   ...
   ...
end Pin;
```

2.2.3 Ground

The Ground component has a positive Pin and a simple equation.

```
model Ground
   Pin p;
equation
   ...
end Ground;
```

2.2.4 Twopin

The TwoPin element has a positive and negative Pin, a voltage u and a current i defined (the current i does not appear in the equations above and is only introduced to simplify notation).

```
model TwoPin
   Pin p, n;
   ...
   ...
equation
   ...
   ...
   ...
end TwoPin;
```

2.2.5 Resistor

To define the resistor the partial model TwoPin is extended and we only add a declaration of the parameter R together with Ohm's law that relates voltage and current to each other.

```
model Resistor
   extends TwoPin;
   ...
equation
   ...
end Resistor;
```

An equivalent model without use of a partial model would look like

```
model Resistor
   ....
   ....
   ....
   ....
   ....
equation
   ....
   ....
   ....
   ....
end Resistor;
```

Note: The Extends command could be thought of as just copying and pasting information from the partial model.

2.2.6 Inductor

The equation relating voltage and current for an inductor together with the inductance L are added to the partial model.

```
model Inductor
   ....
   ....
equation
   ....
end Inductor;
```

2.2.7 VoltageSource

Here the partial model is extended with the trivial equation that the voltage between the positive and negative pin of the voltage source is kept constant.

```
model VoltageSource
   ....
   ....
equation
   ....
end VoltageSource;
```

2.2.8 A Circuit

An example of a simple circuit where we instantiate the parameters of the components to other values than the default.

```
model Circuit
   Resistor R1(R=0.9);
   Inductor L1(L=0.01);
   Ground G;
   VoltageSource EE(V=5);
equation
   connect(EE.p, R1.p);
   connect(R1.n, L1.p);
   connect(L1.n, G.p);
   connect(EE.n, G.p);
end Circuit;
```

Simulate the circuit

```
simulate(Circuit, startTime=0, stopTime=1)
```

The signals that can be plotted.

Plot the current through the resistor:

```
plot(R1.i)
```

Exercise 3 - Equations, Algorithms, and Functions

1 AngleTransformer Using Equations

Show that the AngleTransformer model below can be used for computing radians or degrees depending on which variable (rad or deg) a value is given. For example, do this by defining and simulating a model that contains two different instances of AngleTransformer, and use modifier equations to these to specify radians or degrees, respectively.

```
model AngleTransformer
   Real rad;
   Real deg;
protected
   constant Real Pi = Modelica.Constants.pi;
equation
   deg = 180/Pi*rad;
end AngleTransformer;
```

2 Faculty

Write a function `faculty`, using a for-loop, such that faculty(n) = 1*2*3*4*....*n.

```
function ...
```

Write a class that contains a function call to `faculty`.

```
class ...
```

3 Nested for-loop

Write a function, `matrixAddition`, for adding two two-dimensional matrices.

Perform a function call to `matrixAddition` with two matrices. Then simulate the class with the function call and plot the result.

4 Functions and Algorithm Sections

a) Write a function, mySum, which calculates the sum of Real numbers, for a vector of arbitrary size. Hint: you may declare an input vector of arbitrary size as having the type Real[:]; also, it might be convenient to use a for-loop or a while-loop to make the summation.

```
function mySum ...
```

Call sum:

```
mySum(...
```

b) Write a function, average, which calculates the average of Real numbers, in a vector of arbitrary size. The function average should make use of a function call to mySum.

```
function average ...
```

Call average:

```
average(...
```

5 Functions - LargestAverage

Write a class, `LargestAverage`, that has two vectors and calculates the average of each of them. Then it compares the averages and sets a variable to true if the first vector is larger than the second and otherwise false.

```

```

Exercise 5 - Hybrid Systems

1Hybrid Modeling with BouncingBall

Locate the BouncingBall model in one of the hybrid modeling sections of DrModelica (e.g. Section 2.9), run it, change it slightly, and re-run it.

2Square Signal

Make a square signal with a period of 1s and that starts at t = 2.5s. Note that it is possible to use either an equation or an algorithm solution. Hint: an easy way is to use sample(...) to generate events, and define a variable that switches sign at each event.

```
model ...
```

3DC Motor - Generator

What is needed if you want to make a hybrid DC motor, i.e. a DC motor that also can act like a generator for a limited time? Make it work like a DC motor for the first 20s, then apply a counteracting torque on the outgoing axis for the next 20s, and then turn off the counteracting torque, i.e. you would like to have a torque pulse starting at 20s and lasting 20s. Draw the following connection diagram in a graphic model editor, and adjust the starting times and signal height for the Step1 and Step2 signal models to get the desired torque pulse.