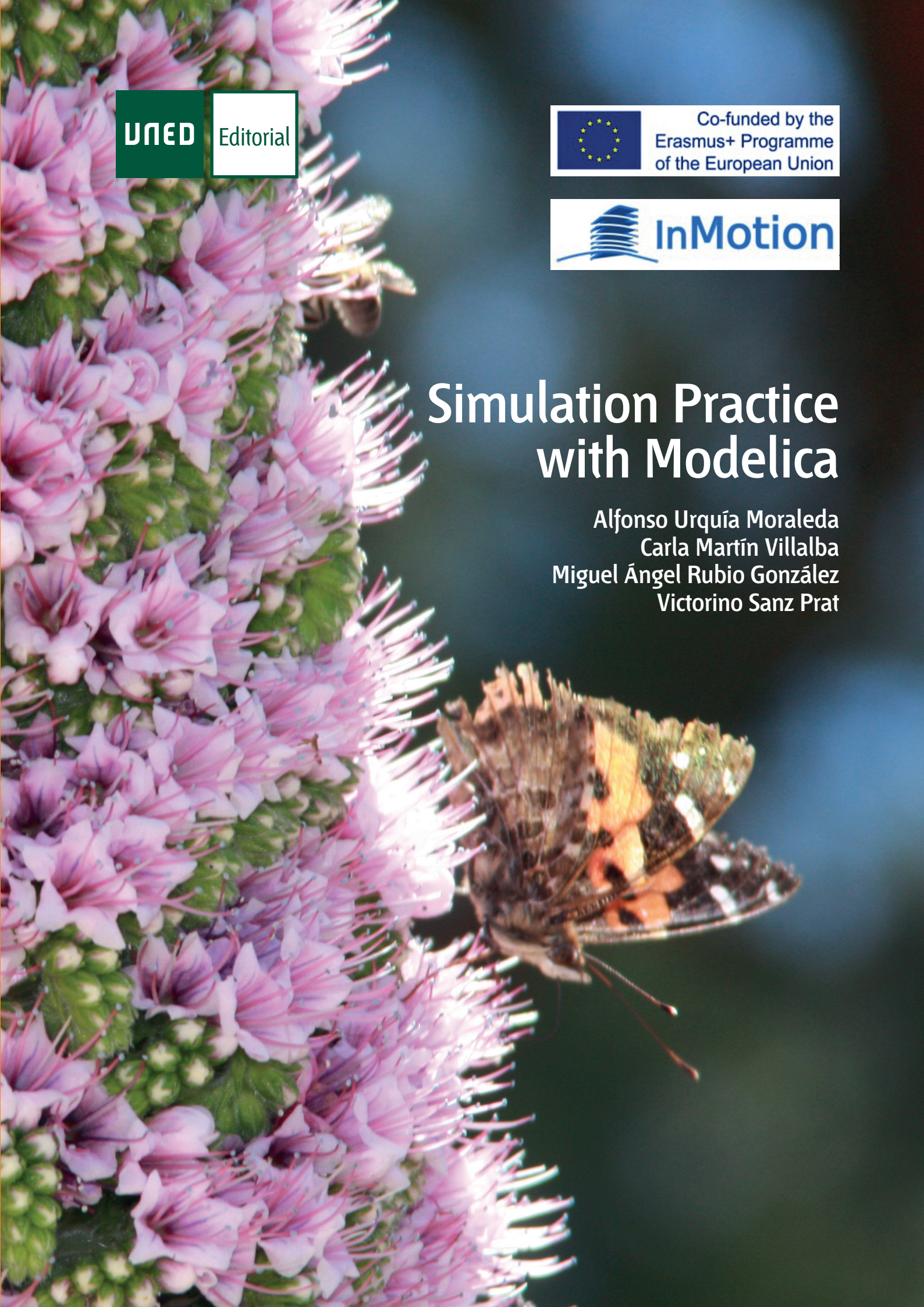


UNED Editorial

Co-funded by the
Erasmus+ Programme
of the European Union

InMotion

Simulation Practice with Modelica

Alfonso Urquía Moraleda
Carla Martín Villalba
Miguel Ángel Rubio González
Victorino Sanz Prat

Co-funded by the
Erasmus+ Programme
of the European Union

This publication was conducted within InMotion project (Innovative teaching and learning strategies in open modelling and simulation environment for student-centered engineering education (573751-EPP-1-2016-1-DE-EPPKA2-CBHE-JP)).

This project has been funded with support from the European Commission. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Simulation practice with Modelica

ALFONSO URQUÍA MORALEDA
CARLA MARTÍN VILLALBA
MIGUEL ÁNGEL RUBIO GONZÁLEZ
VICTORINO SANZ PRAT

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

Co-funded by the Erasmus+ Programme of the European Union

Innovative teaching and learning strategies in open modelling and simulation environment for student-centered engineering education
573751-EPP-1-2016-1-DE-EPPKA2-CBHE-JP

InMotion

SIMULATION PRACTICE WITH MODELICA

www.uned.es/publicaciones

ISBN: 978-84-362-7403-5

Edición digital: octubre de 2018

Contents

Modelica code

7.4. Draining of a benzene storage tank through a pipe (4/4).
8.1. Two-tank system (1/8).
8.2. Two-tank system (2/8).
8.3. Two-tank system (3/8).
8.4. Two-tank system (4/8).
8.5. Two-tank system (5/8).
8.6. Two-tank system (6/8).
8.7. Two-tank system (7/8).
8.8. Two-tank system (8/8).
9.1. Double-pipe heat exchanger (1/4).
9.2. Double-pipe heat exchanger (2/4).
9.3. Double-pipe heat exchanger (3/4).
9.4. Double-pipe heat exchanger (4/4).
10.1. The Game of Life model.
10.2. The Game of Life model with animation (1/2).
10.3. The Game of Life model with animation (2/2).
10.4. The Game of Life model using external functions (1/2).
10.5. The Game of Life model using external functions (2/2).
11.1. Gaussian dispersion model (1/3).
11.2. Gaussian dispersion model (2/3).
11.3. Gaussian dispersion model (3/3).
12.1. Simplified Tennessee Eastman model (1/8).
12.2. Simplified Tennessee Eastman model (2/8).
12.3. Simplified Tennessee Eastman model (3/8).
12.4. Simplified Tennessee Eastman model (4/8).
12.5. Simplified Tennessee Eastman model (5/8).
12.6. Simplified Tennessee Eastman model (6/8).
12.7. Simplified Tennessee Eastman model (7/8).
12.8. Simplified Tennessee Eastman model (8/8).
13.1. Gas diffusion in porous medium (1/2).
13.2. Gas diffusion in porous medium (2/2).
13.3. Diffusion of binary gas mixture and liquid in porous medium (1/4).
13.4. Diffusion of binary gas mixture and liquid in porous medium (2/4).
13.5. Diffusion of binary gas mixture and liquid in porous medium (3/4).
13.6. Diffusion of binary gas mixture and liquid in porous medium (4/4).
13.7. PEM fuel cell (1/22). Connector.
13.8. PEM fuel cell (2/22). Control volume of membrane.
13.9. PEM fuel cell (3/22). Control volume of membrane.
13.10. PEM fuel cell (4/22). Control volume of diffusion layer.
13.11. PEM fuel cell (5/22). Control volume of diffusion layer.
13.12. PEM fuel cell (6/22). Control volume of catalyst layer.
13.13. PEM fuel cell (7/22). Control volume of catalyst layer.
13.14. PEM fuel cell (8/22). Transport phenomena in the membrane.
13.15. PEM fuel cell (9/22). Transport phenomena in the diffusion layer.

Preface

Modeling and simulation of dynamical systems have many applications in Engineering, playing a fundamental role in system design, analysis, control and optimization. Support decision systems and training simulators are frequently based on mathematical modeling and computer simulation.

As simulation projects become larger and more complex, the impact of the modeling methodology and the software tools on the project cost is more evident. Adequate methodologies and tools are key success factors. Complex simulation projects typically require working in teams. Therefore, it is desirable that methodologies and tools facilitate splitting the modeling task among the team members, allowing them to work independently. Another key feature is model reusability. The *object-oriented modeling methodology* suits these requirements.

Modelica is a modeling language conceived to facilitate object-oriented modeling of cyber-physical systems, in which phenomena in different physical domains (e.g., electrical, mechanical, thermo-fluid, chemical and control systems) appear interrelated. The target models, the so-called hybrid DAE models, are dynamic mathematical models described in terms of ordinary differential equations with derivative with respect to time, algebraic equations, and events.

Modelica is a free modeling language, developed by a non-profit organization – the *Modelica Association* – with the aim of serving as a standard language for model exchange among developers and tools. The website of the Modelica Association, `www.modelica.org`, hosts documentation about the language (specifications, scientific articles, tutorials, textbooks, etc.), and links to free and commercial model libraries and software. The reader is encouraged to visit the Modelica Association website to find out more about Modelica.

Aim and structure of the book

As declared in the book's title, our aim in writing this activity book is to provide an introduction to the simulation practice with Modelica. To this end, we propose a series of independent hands-on assignments of increasing complexity.

Each assignment contains the description of a system and a mathematical model of the system's behavior. The proposed task often consists in describing this mathematical model in Modelica and simulate it. In some assignments, the system's behavior is described as an atomic model, without internal structure. Some other assignments ask to design and implement a model library, and to compose the system model by instantiating and connecting components from this model library.

The thirteen assignments collected in this book share a common structure: system description, task proposal and solution. Readers are encouraged to firstly try to solve by themselves the task, by developing and simulating the models using a Modelica modeling environment, and, next, to compare their own results with the solution.

Before start working with this activity book, it is advisable to read its companion theory book: a free e-book entitled "*Modeling and simulation in Engineering using Modelica*", written by Alfonso Urquía and Carla Martín, and published by Editorial UNED in 2018 (ISBN – PDF: 9788436273090). The theory book provides all the previous knowledge on the Modelica language required to complete the assignments.

Learning objectives

The main learning objectives of each assignment are listed below.

Assignment 1 **Springs, damper and lever**

- Practice modeling of simple mechanical systems composed of springs, dampers and levers.
- Analyze the computational causality of DAE systems.
- Describe a DAE system as an atomic model in Modelica, and simulate the model.

Assignment 2 **Springs, pulley and load**

- Practice modeling of simple mechanical systems composed of springs, pulleys and loads.

- Describe a DAE system as an atomic model in Modelica, setting the initial conditions.

Assignment 3 **Bond graph library**

- Getting started with bond graph modeling using Modelica.
- Design and implement model libraries in Modelica.

Assignment 4 **Source of liquid**

- Use if expressions, if clauses, and records.

Assignment 5 **Ideal gas in a heated container**

- Practice modeling of variable structure systems.
- Practice the use of the *unit*, *start* and *fixed* attributes.

Assignment 6 **Hysteresis controller**

- Practice modeling of finite state machines.
- Practice the declaration and use of block classes, and matrix equations.

Assignment 7 **Draining of a benzene storage tank**

- Practice developing a library, and composing a model by instantiating and connecting library components.

Assignment 8 **Heating a liquid mixture**

- Practice modeling of simple thermo-hydraulic systems, posing mass and energy balances, and describing changes in the liquid flow direction.
- Describe multi-mode models in Modelica.

Assignment 9 **Double-pipe heat exchanger**

- Practice modeling of cocurrent and countercurrent heat exchangers, with local variations in physical properties and heat transfer coefficients.
- Use array variables in Modelica.
- Facilitate the numerical solution of models with systems of simultaneous nonlinear equations.

Assignment 10 **Cellular Automata – The Game of Life**

- Modeling and simulation of discrete-event models using Modelica.

- Experience and practice with the external function interface of Modelica.
- Implement graphical animations for the simulations using visualizers of the *Modelica.Mechanics.Multibody* library, and gnuplot.
- Analyze the simulation performance.

Assignment 11 Air pollution

- Get insight into the Gaussian plume model.
- Use arrays, for loops, and enumeration types.
- Describe a DAE system as an atomic model in Modelica.

Assignment 12 Simplified Tennessee Eastman model

- Practice modeling of hydraulic components and simple chemical reactors.
- Use records to describe fluid properties.
- Use arrays of variables and for loops.
- Embed a Modelica model into a Simulink block using FMI.

Assignment 13 PEM fuel cell

- Practice modeling PEM fuel cells, which implies modeling gas and liquid diffusion in porous media, electrochemical reactions, and electric circuits.
- Use spatial discretization to solve PDEs in Modelica.

About the authors

The authors are professors in the Departamento de Informática y Automática, at the Universidad Nacional de Educación a Distancia (UNED) in Madrid, Spain; and members of the research group on Modelling & Simulation in Control Engineering of UNED. Further information is available at: `www.euclides.dia.uned.es`

Acknowledgment

This book has been written during the course of the Erasmus+ project "**InMotion - Innovative teaching and learning strategies in open modelling and simulation environment for student-centered engineering education**", Project No. 573751-EPP-1-2016-1-DE-EPPKA2-CBHE-JP, co-funded by the Erasmus+ Programme of the European Union.

Disclaimer

The InMotion project has been funded with support from the European Commission. The European Commission support for the production of this book does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Assignment 1

Springs, damper and lever

Purpose of this assignment

- Practice modeling of simple mechanical systems composed of springs, dampers and levers.
- Analyze the computational causality of DAE systems.
- Describe a DAE system as an atomic model in Modelica, and simulate the model.

1.1 System description

A mechanical system composed of two springs, a damper, a pulley and a lever is shown in Figure 1.1. The lever consists of a board that is allowed to rotate about a fulcrum.

It is assumed that the masses of the pulley and the board are negligible, and the springs and the damper behave linearly. The velocity v_1 is a known function of time. It is described by Eq. (1.1), where $V_{1,0}$ and w are known parameters.

$$v_1 = \begin{cases} V_{1,0} \cdot \sin(w \cdot t) & \text{if } 2 < t < 20 \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

The distances from the fulcrum to the points of the board where the forces are applied are L_2, L_3 and L_4 respectively. The velocities of these points are v_2, v_3 and v_4

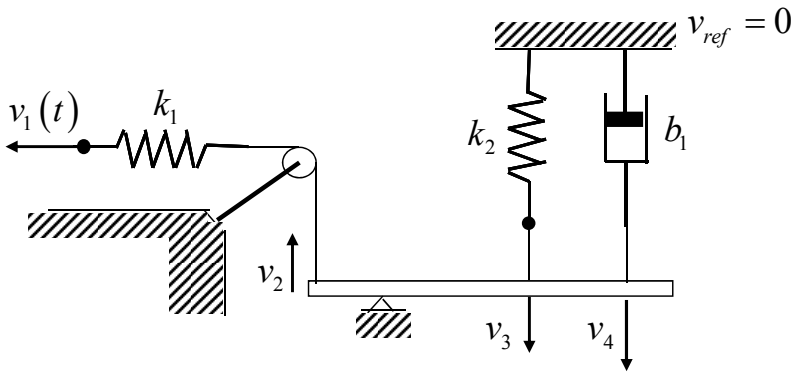


Figure 1.1: Mechanical system.

(see the figure). The forces exerted by the springs and the damper on the board are named F_2, F_3 and F_4 respectively. The sign convention for the velocities is indicated in the figure: the arrow directions represent positive velocities.

1.2 Tasks

1. Write the equations that describe the evolution in time of the spring elongations (e_1 and e_2), the velocities (v_1, v_2, v_3 and v_4) and the forces (F_2, F_3 and F_4).

2. Analyze the computational causality of the model. Write the simulation algorithm of the model, employing the explicit Euler method for solving the ordinary differential equations.

3. Describe the model in Modelica as an atomic model. When translating the model, generate the listing of translated Modelica code, and compare the computational causality calculated by the modeling environment with your solution to Task 2. Simulate the model.

1.3 Solution to Task 1

The spring elongation is the difference between the actual length of the spring and its natural length. The elongations and constants of the springs are e_1 and e_2, and k_1 and k_2, respectively. The springs are assumed to be ideal. According to Hooke's Law, the restoring force of an ideal spring is proportional to its elongation.

$$F_2 = k_1 \cdot e_1 \tag{1.2}$$
$$F_3 = -k_2 \cdot e_2 \tag{1.3}$$

The change in the spring elongation is related to the difference in the velocities of the spring terminals.

$$\frac{de_1}{dt} = v_1 - v_2 \tag{1.4}$$
$$\frac{de_2}{dt} = v_3 \tag{1.5}$$

The force exerted by the damper is proportional to the difference in the velocities of its terminals. In this system, one terminal is fixed. The other terminal moves with velocity v_4.

$$F_4 = -b_1 \cdot v_4 \tag{1.6}$$

The board of the lever moves with a certain angular velocity ϖ. The velocities v_2, v_3 and v_4 can be calculated from this angular velocity.

$$\varpi = \frac{v_2}{L_2} = \frac{v_3}{L_3} = \frac{v_4}{L_4} \tag{1.7}$$

On the other hand, as it was supposed that the mass of the board is negligible. This implies that the lever does not store energy, and the applied forces are equilibrated instantaneously.

$$F_2 \cdot L_2 + F_3 \cdot L_3 + F_4 \cdot L_4 = 0 \tag{1.8}$$

The previous 8 equations, together with Eq. (1.1), compose the model. For the readers convenience, the complete model is written below. It has the following nine time-dependent variables: e_1, e_2, v_1, v_2, v_3, v_4, F_2, F_3 and F_4.

$$v_1 = \begin{cases} V_{1,0} \cdot \sin(w \cdot t) & \text{if } 2 < t < 20 \\ 0 & \text{otherwise} \end{cases} \tag{1.9}$$

$$F_2 = k_1 \cdot e_1 \tag{1.10}$$

$$\frac{de_1}{dt} = v_1 - v_2 \tag{1.11}$$

$$F_2 \cdot L_2 + F_3 \cdot L_3 + F_4 \cdot L_4 = 0 \tag{1.12}$$

$$\frac{v_2}{L_2} = \frac{v_3}{L_3} \tag{1.13}$$

$$\frac{v_2}{L_2} = \frac{v_4}{L_4} \tag{1.14}$$

$$F_3 = -k_2 \cdot e_2 \tag{1.15}$$

$$\frac{de_2}{dt} = v_3 \tag{1.16}$$

$$F_4 = -b_1 \cdot v_4 \tag{1.17}$$

1.4 Solution to Task 2

Let's analyze the computational causality of the model, selecting e_1 and e_2 as state variables. Firstly, the derivatives are replaced by dummy variables:

$$\frac{de_1}{dt} \rightarrow dere_1 \qquad\qquad \frac{de_2}{dt} \rightarrow dere_2 \tag{1.18}$$

The unknown variables to evaluate from the model equations are $dere_1$, $dere_2$, v_1, v_2, v_3, v_4, F_2, F_3 and F_4. The original incidence matrix is shown below. The Eq. (1.9) is represented as $v_1 = f(t)$ (see the label beside the first row of the incidence matrix) to denote that v_1 is a function of time.

$$
\begin{array}{c}
 \\
v_1{=}f(t) \\
F_2{=}k_1{\cdot}e_1 \\
dere_1{=}v_1{-}v_2 \\
F_2{\cdot}L_2{+}F_3{\cdot}L_3{+}F_4{\cdot}L_4{=}0 \\
\frac{v_2}{L_2}{=}\frac{v_3}{L_3} \\
\frac{v_2}{L_2}{=}\frac{v_4}{L_4} \\
F_3{=}{-}k_2{\cdot}e_2 \\
dere_2{=}v_3 \\
F_4{=}{-}b_1{\cdot}v_4
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
dere_1 & dere_2 & v_1 & v_2 & v_3 & v_4 & F_2 & F_3 & F_4
\end{array} \\
\left(\begin{array}{ccccccccc}
0 & 0 & \mathrm{X} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathrm{X} & 0 & 0 \\
\mathrm{X} & 0 & \mathrm{X} & \mathrm{X} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathrm{X} & \mathrm{X} & \mathrm{X} \\
0 & 0 & 0 & \mathrm{X} & \mathrm{X} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \mathrm{X} & 0 & \mathrm{X} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathrm{X} & 0 \\
0 & \mathrm{X} & 0 & 0 & \mathrm{X} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathrm{X} & 0 & 0 & \mathrm{X}
\end{array}\right)
\end{array}
\tag{1.19}
$$

The next step in the analysis of the computational causality is to check whether the model is structurally singular. This model satisfies that:

1. The number of unknown variables and equations are equal (= 9).

2. There exists a sequence of permutations of the incidence matrix columns that allows to obtain a permuted matrix with all the elements on the main diagonal different from zero. This is demonstrated below.

$$
\begin{array}{c}
 \\
v_1{=}f(t) \\
F_2{=}k_1{\cdot}e_1 \\
dere_1{=}v_1{-}v_2 \\
F_2{\cdot}L_2{+}F_3{\cdot}L_3{+}F_4{\cdot}L_4{=}0 \\
\frac{v_2}{L_2}{=}\frac{v_3}{L_3} \\
\frac{v_2}{L_2}{=}\frac{v_4}{L_4} \\
F_3{=}{-}k_2{\cdot}e_2 \\
dere_2{=}v_3 \\
F_4{=}{-}b_1{\cdot}v_4
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
v_1 & F_2 & dere_1 & F_4 & v_3 & v_2 & F_3 & dere_2 & v_4
\end{array} \\
\left(\begin{array}{ccccccccc}
\mathrm{X} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \mathrm{X} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathrm{X} & 0 & \mathrm{X} & 0 & 0 & \mathrm{X} & 0 & 0 & 0 \\
0 & \mathrm{X} & 0 & \mathrm{X} & 0 & 0 & \mathrm{X} & 0 & 0 \\
0 & 0 & 0 & 0 & \mathrm{X} & \mathrm{X} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathrm{X} & 0 & 0 & \mathrm{X} \\
0 & 0 & 0 & 0 & 0 & 0 & \mathrm{X} & 0 & 0 \\
0 & 0 & 0 & 0 & \mathrm{X} & 0 & 0 & \mathrm{X} & 0 \\
0 & 0 & 0 & \mathrm{X} & 0 & 0 & 0 & 0 & \mathrm{X}
\end{array}\right)
\end{array}
\tag{1.20}
$$

Therefore, the model is not structurally singular. The computational causality is assigned reasoning as follows.

1. v_1 is the only unknown variable that intervenes in Eq. (1.9). The same condition is satisfied by F_2 in Eq. (1.10), and by F_3 in Eq. (1.15). Therefore, these variables have to be calculated from these equations. As these unknown variables intervenes in other equations, they are moved to the first columns and the three equations to the first rows.

The $dere_1$ variable only intervenes in the Eq. (1.11), and $dere_2$ in Eq. (1.16). For this reason, these variables have to be calculated from these equations. As these variables do not intervene in any other equation, they are moved to the last columns, and the equations to the last rows.

$$
\begin{array}{r}
 \\
v_1{=}f(t) \\
F_2{=}k_1{\cdot}e_1 \\
F_3{=}{-}k_2{\cdot}e_2 \\
F_2{\cdot}L_2{+}F_3{\cdot}L_3{+}F_4{\cdot}L_4{=}0 \\
\frac{v_2}{L_2}{=}\frac{v_3}{L_3} \\
\frac{v_2}{L_2}{=}\frac{v_4}{L_4} \\
F_4{=}{-}b_1{\cdot}v_4 \\
dere_1{=}v_1{-}v_2 \\
dere_2{=}v_3
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
v_1 & F_2 & F_3 & v_2 & v_3 & v_4 & F_4 & dere_1 & dere_2
\end{array} \\
\left(\begin{array}{ccccccccc}
\boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \mathrm{X} & \mathrm{X} & 0 & 0 & 0 & \mathrm{X} & 0 & 0 \\
0 & 0 & 0 & \mathrm{X} & \mathrm{X} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \mathrm{X} & 0 & \mathrm{X} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathrm{X} & \mathrm{X} & 0 & 0 \\
\mathrm{X} & 0 & 0 & \mathrm{X} & 0 & 0 & 0 & \boxed{\mathrm{X}} & 0 \\
0 & 0 & 0 & 0 & \mathrm{X} & 0 & 0 & 0 & \boxed{\mathrm{X}}
\end{array}\right)
\end{array}
\tag{1.21}
$$

2. Assuming that F_2 and F_3 are calculated from Eqs. (1.10) and (1.15) respectively, the Eq. (1.12) only contains one unknown variable that has not been calculated yet: the F_4 variable. Therefore, F_4 is calculated from Eq. (1.12). As this variable intervenes in other equations, it is moved to the fourth columns, and the equation is not moved from the fourth row.

$$
\begin{array}{r}
 \\
v_1{=}f(t) \\
F_2{=}k_1{\cdot}e_1 \\
F_3{=}{-}k_2{\cdot}e_2 \\
F_2{\cdot}L_2{+}F_3{\cdot}L_3{+}F_4{\cdot}L_4{=}0 \\
\frac{v_2}{L_2}{=}\frac{v_3}{L_3} \\
\frac{v_2}{L_2}{=}\frac{v_4}{L_4} \\
F_4{=}{-}b_1{\cdot}v_4 \\
dere_1{=}v_1{-}v_2 \\
dere_2{=}v_3
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
v_1 & F_2 & F_3 & F_4 & v_2 & v_3 & v_4 & dere_1 & dere_2
\end{array} \\
\left(\begin{array}{ccccccccc}
\boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \mathrm{X} & \mathrm{X} & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \mathrm{X} & \mathrm{X} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \mathrm{X} & 0 & \mathrm{X} & 0 & 0 \\
0 & 0 & 0 & \mathrm{X} & 0 & 0 & \mathrm{X} & 0 & 0 \\
\mathrm{X} & 0 & 0 & 0 & \mathrm{X} & 0 & 0 & \boxed{\mathrm{X}} & 0 \\
0 & 0 & 0 & 0 & 0 & \mathrm{X} & 0 & 0 & \boxed{\mathrm{X}}
\end{array}\right)
\end{array}
\tag{1.22}
$$

3. Assuming that F_4 is calculated from Eq. (1.12), Eq. (1.17) only contains one unknown variable that has not been calculated yet: the v_4 variable. Therefore, v_4 is calculated from Eq. (1.17).

$$
\begin{array}{c}
\\
v_1{=}f(t) \\
F_2{=}k_1{\cdot}e_1 \\
F_3{=}{-}k_2{\cdot}e_2 \\
F_2{\cdot}L_2{+}F_3{\cdot}L_3{+}F_4{\cdot}L_4{=}0 \\
F_4{=}{-}b_1{\cdot}v_4 \\
\frac{v_2}{L_2}{=}\frac{v_3}{L_3} \\
\frac{v_2}{L_2}{=}\frac{v_4}{L_4} \\
dere_1{=}v_1{-}v_2 \\
dere_2{=}v_3
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc} v_1 & F_2 & F_3 & F_4 & v_4 & v_2 & v_3 & dere_1 & dere_2 \end{array} \\
\left(\begin{array}{ccccccccc}
\boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \mathrm{X} & \mathrm{X} & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \mathrm{X} & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathrm{X} & \mathrm{X} & 0 & 0 \\
0 & 0 & 0 & 0 & \mathrm{X} & \mathrm{X} & 0 & 0 & 0 \\
\mathrm{X} & 0 & 0 & 0 & 0 & \mathrm{X} & 0 & \boxed{\mathrm{X}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathrm{X} & 0 & \boxed{\mathrm{X}}
\end{array}\right)
\end{array}
\tag{1.23}
$$

4. Assuming that v_4 is calculated from Eq. (1.17), Eq. (1.14) only contains one unknown variable that has not been calculated yet: the v_2 variable. Therefore, v_2 is calculated from Eq. (1.14). The incidence matrix in BLT form is shown below.

$$
\begin{array}{c}
\\
v_1{=}f(t) \\
F_2{=}k_1{\cdot}e_1 \\
F_3{=}{-}k_2{\cdot}e_2 \\
F_2{\cdot}L_2{+}F_3{\cdot}L_3{+}F_4{\cdot}L_4{=}0 \\
F_4{=}{-}b_1{\cdot}v_4 \\
\frac{v_2}{L_2}{=}\frac{v_4}{L_4} \\
\frac{v_2}{L_2}{=}\frac{v_3}{L_3} \\
dere_1{=}v_1{-}v_2 \\
dere_2{=}v_3
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc} v_1 & F_2 & F_3 & F_4 & v_4 & v_2 & v_3 & dere_1 & dere_2 \end{array} \\
\left(\begin{array}{ccccccccc}
\boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \mathrm{X} & \mathrm{X} & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \mathrm{X} & \boxed{\mathrm{X}} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \mathrm{X} & \boxed{\mathrm{X}} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathrm{X} & \boxed{\mathrm{X}} & 0 & 0 \\
\mathrm{X} & 0 & 0 & 0 & 0 & \mathrm{X} & 0 & \boxed{\mathrm{X}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathrm{X} & 0 & \boxed{\mathrm{X}}
\end{array}\right)
\end{array}
\tag{1.24}
$$

Observe that all the diagonal blocks of the BLT incidence matrix are scalar. The model variables can be solved sequentially, this is, there are not algebraic loops. The sorted model, with the computational causality annotated, is shown below.

$$
[v_1] = \begin{cases} V_{1,0} \cdot \sin(w \cdot t) & \text{if } 2 < t < 20 \\ 0 & \text{otherwise} \end{cases} \tag{1.25}
$$

$$
[F_2] = k_1 \cdot e_1 \tag{1.26}
$$

$$
[F_3] = -k_2 \cdot e_2 \tag{1.27}
$$

$$
\begin{aligned}
F_2 \cdot L_2 + F_3 \cdot L_3 + [F_4] \cdot L_4 &= 0 && (1.28)\\
F_4 &= -b_1 \cdot [v_4] && (1.29)\\
\frac{[v_2]}{L_2} &= \frac{v_4}{L_4} && (1.30)\\
\frac{v_2}{L_2} &= \frac{[v_3]}{L_3} && (1.31)\\
[dere_1] &= v_1 - v_2 && (1.32)\\
[dere_2] &= v_3 && (1.33)
\end{aligned}
$$

As the unknown variables intervene linearly in the equations, the sorted and solved model can be obtained manipulating symbolically the equations. It is shown below.

$$
\begin{aligned}
[v_1] &= \begin{cases} V_{1,0} \cdot \sin(w \cdot t) & \text{if } 2 < t < 20 \\ 0 & \text{otherwise} \end{cases} && (1.34)\\
[F_2] &= k_1 \cdot e_1 && (1.35)\\
[F_3] &= -k_2 \cdot e_2 && (1.36)\\
[F_4] &= \frac{-1}{L_4} \cdot (F_2 \cdot L_2 + F_3 \cdot L_3) && (1.37)\\
[v_4] &= \frac{-F_4}{b_1} && (1.38)\\
[v_2] &= \frac{L_2}{L_4} \cdot v_4 && (1.39)\\
[v_3] &= \frac{L_3}{L_2} \cdot v_2 && (1.40)\\
[dere_1] &= v_1 - v_2 && (1.41)\\
[dere_2] &= v_3 && (1.42)
\end{aligned}
$$

The simulation algorithm is shown in Figure 1.2.

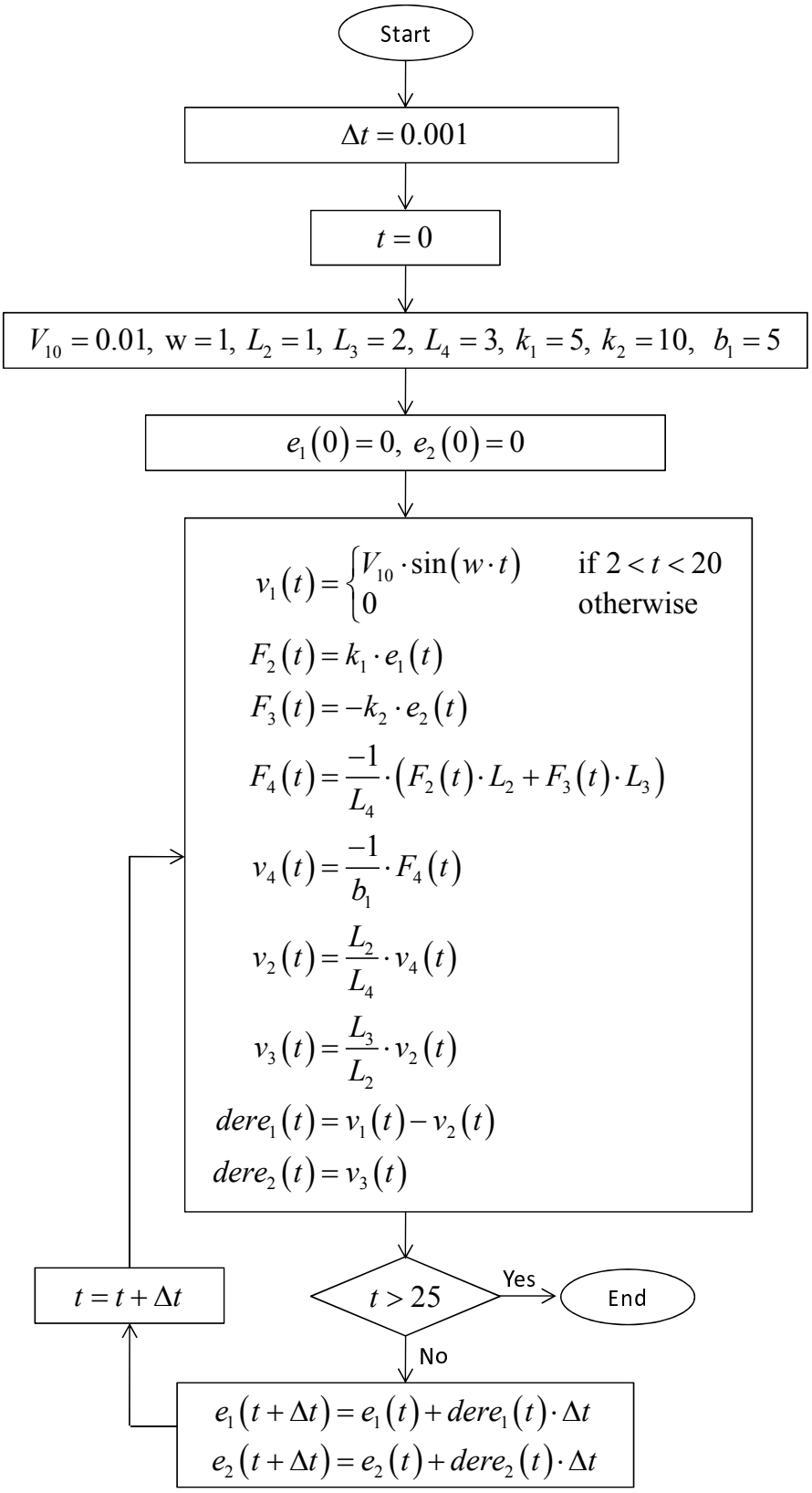


Figure 1.2: Simulation algorithm of the mechanical system.

1.5 Solution to Task 3

The model can be described in Modelica as shown in Modelica Code 1.1. Observe that the state variables are initialized to zero using the *start* and *fixed* attributes.

```
model SpringsDamperLever
  import SI = Modelica.SIunits;
  // Parameters of the v1 velocity
  parameter SI.Velocity V10=0.01;
  parameter SI.AngularFrequency w=1;
  // Perpendicular distances between the forces and the fulcrum
  parameter SI.Length L2=1;
  parameter SI.Length L3=2;
  parameter SI.Length L4=3;
  // Springs and damper
  parameter SI.TranslationalSpringConstant k1=5;
  parameter SI.TranslationalSpringConstant k2=10;
  parameter SI.TranslationalDampingConstant b1=5;
  SI.Length e1(start=0, fixed=true);
  SI.Length e2(start=0, fixed=true);
  SI.Velocity v1;
  SI.Velocity v2;
  SI.Velocity v3;
  SI.Velocity v4;
  SI.Force F2;
  SI.Force F3;
  SI.Force F4;
equation
  // Velocity imposed to the left terminal of spring 1
  v1 = if time > 2 and time < 20 then V10*sin(w*time) else 0;
  // Spring 1
  F2 = k1*e1;
  der(e1) = v1 - v2;
  // Lever
  F2*L2 + F3*L3 + F4*L4 = 0;
  v2/L2 = v3/L3;
  v2/L2 = v4/L4;
  // Spring 2
  F3 = -k2*e2;
  der(e2) = v3;
  // Damper
  F4 = -b1*v4;
  annotation (uses(Modelica(version="3.2.2")));
end SpringsDamperLever;
```

Modelica Code 1.1: Springs, damper and lever system.

The model is simulated using Dymola. The sorted and solved model is saved to file during the translation if the following checkbox is checked before launching the translation: *Simulation Setup > Translation > Generate listing of translated Modelica code in dsmodel.mof*. The content of the *dsmodel.mof* file is listed below.

```
// Translated Modelica model generated by Dymola from Modelica model
//  SpringsDamperLever

// --------------------------------------------------------------------------

// Initial Section

// --------------------------------------------------------------------------

// Dynamics Section
  v1 := (if time > 2 and time < 20 then V10*sin(w*time) else 0);
  F2 := k1*e1;
  F3 :=  -k2*e2;

  // Linear system of equations
    // Symbolic solution
      /*  Original equation
      L4*F4 =  -(F2*L2+F3*L3);
      */
      F4 :=  -(F2*L2+F3*L3)/L4;
  // End of linear system of equations


  // Linear system of equations
    // Symbolic solution
      /*  Original equation
      b1*v4 =  -F4;
      */
      v4 :=  -F4/b1;
  // End of linear system of equations

  // Linear system of equations
    // Symbolic solution
      /*  Original equation
      v2/L2 = v4/L4;
      */
      v2 := v4*L2/L4;
  // End of linear system of equations

  der(e1) := v1-v2;

  // Linear system of equations
    // Symbolic solution
      /*  Original equation
       -der(e2)/L3 =  -v2/L2;
      */
      der(e2) := v2*L3/L2;
  // End of linear system of equations

// --------------------------------------------------------------------------

// Eliminated alias variables
  v3 = der(e2);
```

The computational causality calculated by Dymola coincides with the solution to Task 2. The simulation result is shown in Figure 1.3.

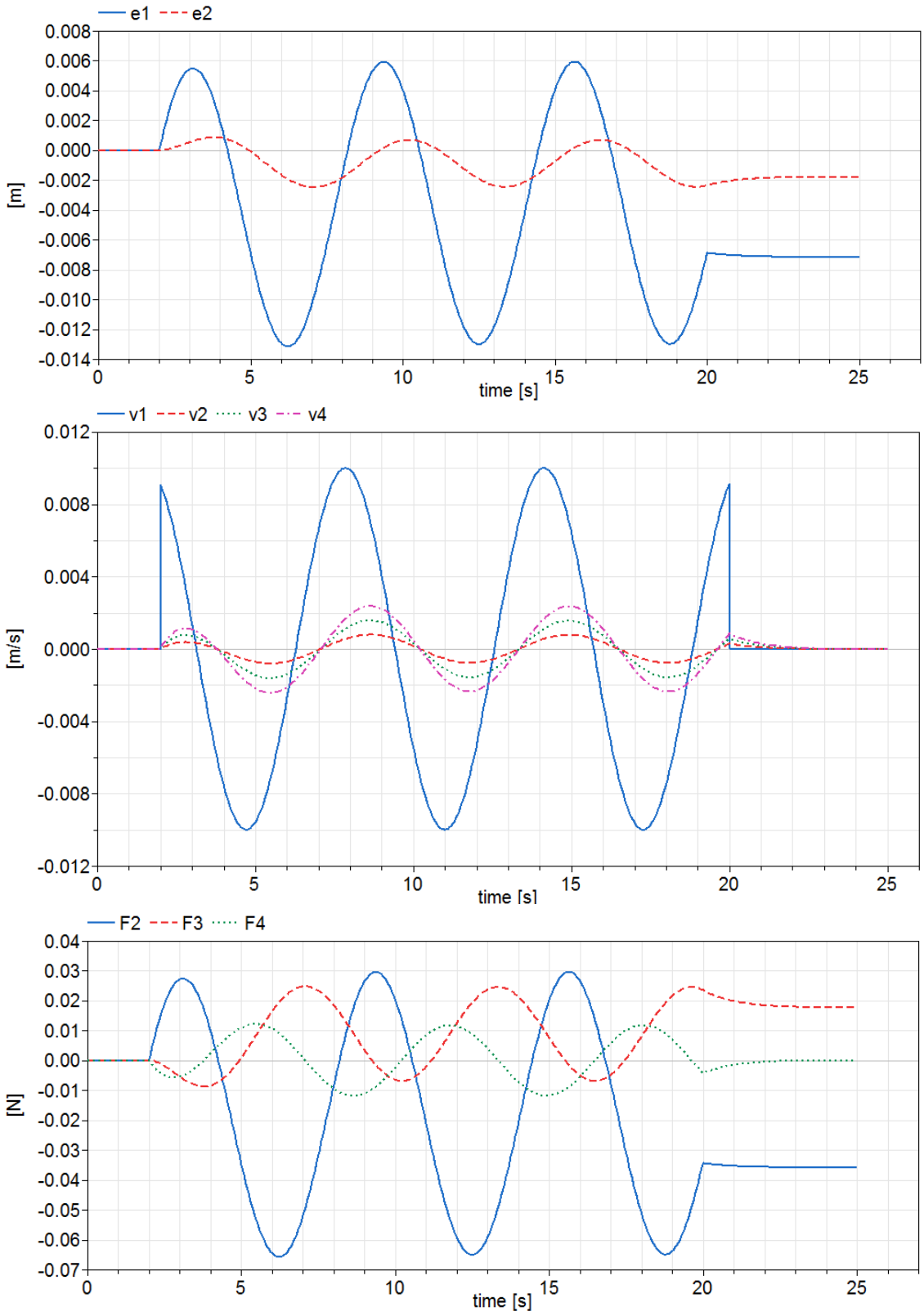


Figure 1.3: Simulation of the mechanical system.

Assignment 2

Springs, pulley and load

Purpose of this assignment

- Practice modeling of simple mechanical systems composed of springs, pulleys and loads.
- Describe a DAE system as an atomic model in Modelica, setting the initial conditions.

2.1 System description

A mechanical system composed of two springs, a pulley and a hanging load is shown in Figure 2.1. The masses of the load and the pulley are m_L and m_p. Assuming that the pulley behaves as a solid circular disk of radius R_p, its moment of inertia J is calculated from Eq. (2.1).

$$J = \frac{1}{2} \cdot m_p \cdot R_p^2 \tag{2.1}$$

The two springs are supposed to be ideal, with spring constants k_1 and k_2 respectively. The gravitational acceleration is denoted as g ($= 9.81$ m/s^2).

The velocities of the spring free terminals are v_1 and v_2. The vertical velocity of the pulley is v_3, and its angular velocity is w. The arrows in Figure 2.1 indicate the positive directions of the velocities (v_1, v_2 and v_3) and the angular velocity (w).

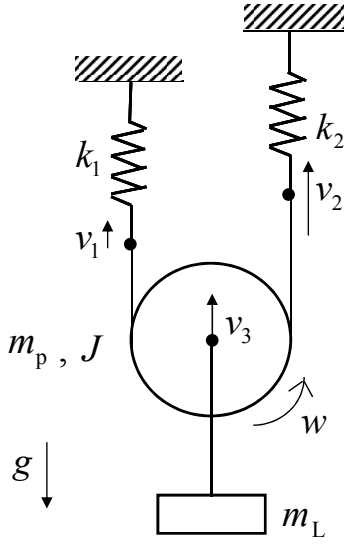


Figure 2.1: Mass hanging from a pulley.

2.2 Tasks

1. Write the equations that describe the evolution in time of the spring elongations (e_1 and e_2), the velocities (v_1, v_2 and v_3), the angular velocity (w), and the forces exerted by the springs (F_1 and F_2).

2. The initial conditions of the simulation are the following: the spring elongations, the vertical velocity of the pulley and its angular velocity are zero. The parameters have the following values: $k_1 = 5$ N/m, $k_2 = 10$ N/m, $m_p = 0.5$ kg, $m_L = 10$ kg and $R_p = 0.3$ m. Pose and solve by hand the initialization problem. Describe the model in Modelica as an atomic model, simulate it and compare your solution with the simulation results.

3. The model is modified to describe the following experiment. The model is initially at steady state. The mass of the load is a time-dependent variable: initially it is equal to 10 kg, but it becomes zero at time 5 s. The spring and pulley parameters have the values indicated in the description of the previous task. Pose and solve by hand the initialization problem. Describe the model in Modelica as an atomic model, simulate it and compare your solution with the simulation results.

4. The previous model is modified, so that the spring constants are calculated at the initialization. Initially the model is in steady state, and the spring elongations are initially 3 m and 1.5 m. Pose and solve by hand the initialization problem, simulate the model and check your solution.

2.3 Solution to Task 1

The following convention is employed: forces pointing upward have a positive magnitude, and pointing downward have a negative magnitude.

The behavior of the ideal springs is described by Eqs. (2.2) – (2.5), where F_1 and F_2 are the forces exerted by the springs, and e_1 and e_2 their elongations (difference between the actual length and the natural length of the spring).

$$F_1 = k_1 \cdot e_1 \tag{2.2}$$
$$\frac{de_1}{dt} = -v_1 \tag{2.3}$$
$$F_2 = k_2 \cdot e_2 \tag{2.4}$$
$$\frac{de_2}{dt} = -v_2 \tag{2.5}$$

The vertical movement of the pulley and the load is described by Eq. (2.6).

$$(m_p + m_L) \cdot \frac{dv_3}{dt} = F_1 + F_2 - g \cdot (m_p + m_L) \tag{2.6}$$

The angular velocity of the pulley is related with the forces exerted by the springs, as described by Eq. (2.7).

$$J \cdot \frac{dw}{dt} = R_p \cdot (F_2 - F_1) \tag{2.7}$$

The displacement of the spring free terminals (x_1 and x_2), the vertical displacement of the pulley (x_3), and the angle (α) rotated by the pulley are related as described by Eqs. (2.8) and (2.9).

$$x_1 = x_3 - R_p \cdot \alpha \tag{2.8}$$
$$x_2 = x_3 + R_p \cdot \alpha \tag{2.9}$$

Differentiating with respect to time these two equations, it is obtained:

$$v_1 = v_3 - R_p \cdot w \tag{2.10}$$
$$v_2 = v_3 + R_p \cdot w \tag{2.11}$$

The model is composed of Eqs. (2.1) – (2.7), (2.10) and (2.11).

2.4 Solution to Task 2

The description in Modelica of the model is shown below. The simulation results are represented in Figure 2.2.

```
model SpringPulleyLoad
  import SI = Modelica.SIunits;
  constant SI.Acceleration g=9.81;
  // Springs
  parameter SI.TranslationalSpringConstant k1=5;
  parameter SI.TranslationalSpringConstant k2=10;
  // Pulley
  parameter SI.Mass Mp=0.5;
  parameter SI.Radius Rp=0.3;
  parameter SI.MomentOfInertia J=0.5*Mp*Rp^2;
  // Load
  parameter SI.Mass ML=10;

  SI.Length e1(start=0, fixed=true);
  SI.Length e2(start=0, fixed=true);
  SI.Velocity v1;
  SI.Velocity v2;
  SI.Velocity v3(start=0, fixed=true);
  SI.Force F1;
  SI.Force F2;
  SI.AngularVelocity w(start=0, fixed=true);
equation
  // Spring 1
  F1 = k1*e1;
  der(e1) = -v1;
  // Spring 2
  F2 = k2*e2;
  der(e2) = -v2;
  // Pulley and load
  (Mp + ML)*der(v3) = F1 + F2 - g*(Mp + ML);
  v1 = v3 - Rp*w;
  v2 = v3 + Rp*w;
  J*der(w) = Rp*(F2 - F1);
  annotation (uses(Modelica(version="3.2.2")));
end SpringPulleyLoad;
```

Modelica Code 2.1: Springs, pulley and load system - Task 2.

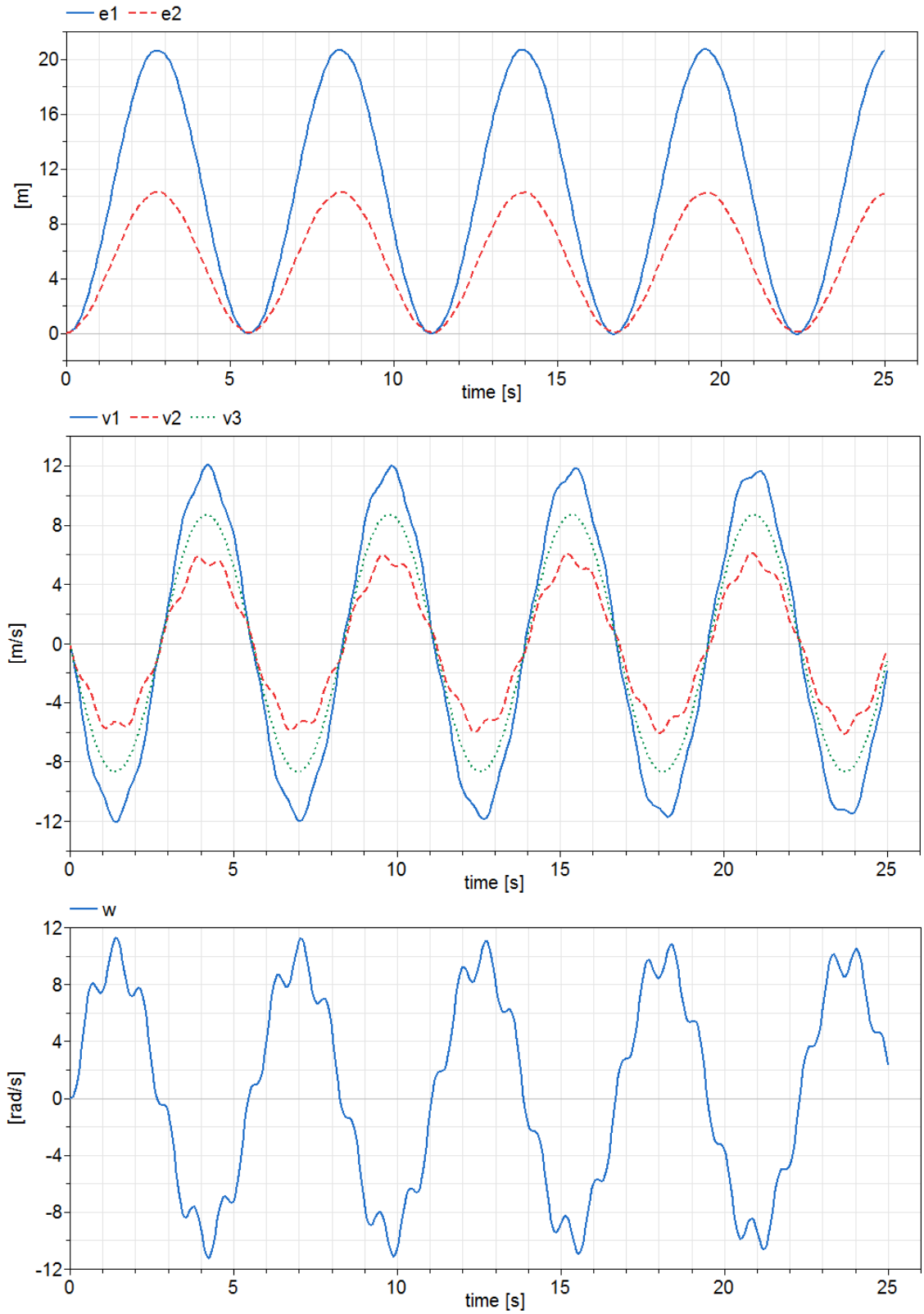


Figure 2.2: Simulation results of Task 2.

The initialization problem consists in solving the variables $\{k_1,\ k_2,\ m_p,\ m_L,\ R_p,\ e_1,\ e_2,\ v_1,\ v_2,\ v_3,\ w,\ F_1,\ F_2,\ J,\ dere_1,\ dere_2,\ derv_3,\ derw\}$ from the following set of equations. The computational causality is annotated.

$$
\begin{aligned}
[k_1] &= 5 & [J] &= \frac{1}{2} \cdot m_p \cdot R_p^2 \\
[k_2] &= 10 & [F_1] &= k_1 \cdot e_1 \\
[m_p] &= 0.5 & [dere_1] &= -v_1 \\
[m_L] &= 10 & [F_2] &= k_2 \cdot e_2 \\
[R_p] &= 0.3 & [dere_2] &= -v_2 \\
[e_1] &= 0 & (m_p + m_L) \cdot [derv_3] &= F_1 + F_2 - g \cdot (m_p + m_L) \\
[e_2] &= 0 & [v_1] &= v_3 - R_p \cdot w \\
[v_3] &= 0 & [v_2] &= v_3 + R_p \cdot w \\
[w] &= 0 & J \cdot [derw] &= R_p \cdot (F_2 - F_1)
\end{aligned}
\tag{2.12}
$$

The initial values of the variables are calculated solving the previous equations, giving:

$$
\begin{array}{ll|ll}
k_1 & = 5 & J & = \frac{1}{2} \cdot 0.5 \cdot 0.3^2 = 0.0225 \\
k_2 & = 10 & F_1 & = 5 \cdot 0 = 0 \\
m_p & = 0.5 & dere_1 & = -v_1 = 0 \\
m_L & = 10 & F_2 & = 10 \cdot 0 = 0 \\
R_p & = 0.3 & dere_2 & = 0 \\
e_1 & = 0 & derv_3 & = \frac{F_1 + F_2 - g \cdot (m_p + m_L)}{m_p + m_L} \\
e_2 & = 0 & & = \frac{0 + 0 - 9.81 \cdot (0.5 + 10)}{0.5 + 10} = -9.81 \\
v_3 & = 0 & v_1 & = v_3 - R_p \cdot w = 0 - 0.3 \cdot 0 = 0 \\
w & = 0 & v_2 & = v_3 + R_p \cdot w = 0 - 0.3 \cdot 0 = 0 \\
 & & derw & = \frac{R_p \cdot (F_2 - F_1)}{J} = \frac{0.3 \cdot (0 - 0)}{0.0225} = 0
\end{array}
\tag{2.13}
$$

2.5 Solution to Task 3

The description in Modelica of the model is shown below. The simulation results are represented in Figure 2.3.

```
model SpringPulleyLoad1
  import SI = Modelica.SIunits;
  constant SI.Acceleration g=9.81;
  // Springs
  parameter SI.TranslationalSpringConstant k1=5;
  parameter SI.TranslationalSpringConstant k2=10;
  // Pulley
  parameter SI.Mass Mp=0.5;
  parameter SI.Radius Rp=0.3;
  parameter SI.MomentOfInertia J=0.5*Mp*Rp^2;
  // Load
  parameter SI.Mass ML0=10;
  SI.Mass ML;

  SI.Length e1;
  SI.Length e2;
  SI.Velocity v1;
  SI.Velocity v2;
  SI.Velocity v3;
  SI.Force F1;
  SI.Force F2;
  SI.AngularVelocity w;
equation
  // Spring 1
  F1 = k1*e1;
  der(e1) = -v1;
  // Spring 2
  F2 = k2*e2;
  der(e2) = -v2;
  // Pulley and load
  ML = if time < 5 then ML0 else 0;
  (Mp + ML)*der(v3) = F1 + F2 - g*(Mp + ML);
  v1 = v3 - Rp*w;
  v2 = v3 + Rp*w;
  J*der(w) = Rp*(F2 - F1);
initial equation
  der(e1) = 0;
  der(e2) = 0;
  der(v3) = 0;
  der(w) = 0;
  annotation (uses(Modelica(version="3.2.2")));
end SpringPulleyLoad1;
```

Modelica Code 2.2: Springs, pulley and load system - Task 3.

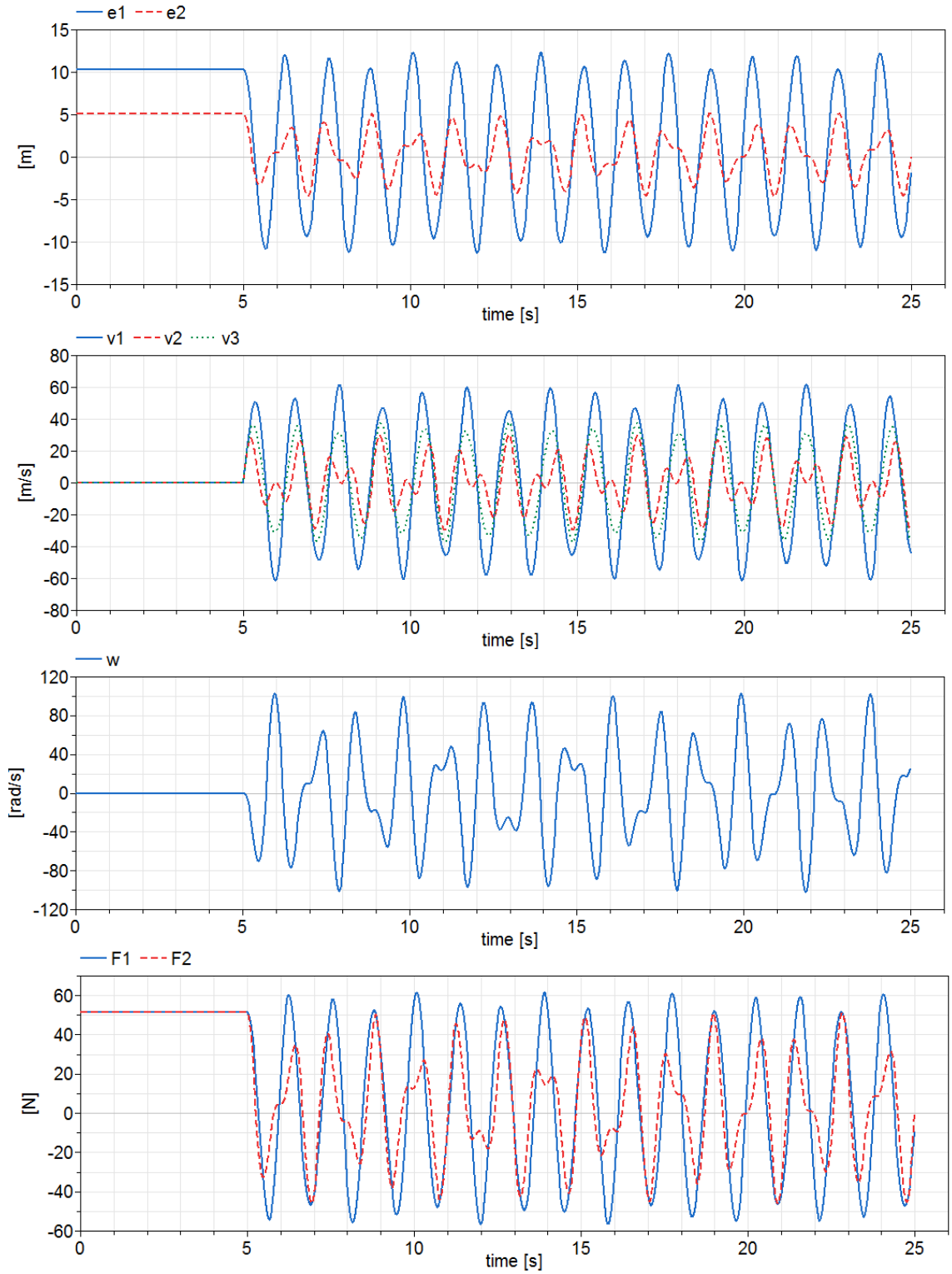


Figure 2.3: Simulation results of Task 3.

The initialization problem consists in solving the variables $\{k_1,\ k_2,\ m_p,\ m_L,\ R_p,\ e_1,\ e_2,\ v_1,\ v_2,\ v_3,\ w,\ F_1,\ F_2,\ J,\ dere_1,\ dere_2,\ derv_3,\ derw\}$ from the following set of equations. The sorted model, with the computational causality annotated, is shown below.

$$
\begin{array}{l}
[k_1] = 5 \\
[k_2] = 10 \\
[m_p] = 0.5 \\
[m_L] = 10 \\
[R_p] = 0.3 \\
[dere_1] = 0 \\
[dere_2] = 0 \\
[derv_3] = 0 \\
[derw] = 0
\end{array}
\qquad
\begin{array}{l}
[J] = \dfrac{1}{2} \cdot m_p \cdot R_p^2 \\
dere_1 = -\left[v_1\right] \\
dere_2 = -\left[v_2\right] \\
\left.\begin{array}{ll} v_1 & = v_3 - R_p \cdot w \\ v_2 & = v_3 + R_p \cdot w \end{array}\right\| \rightarrow [v_3], [w] \\
\left.\begin{array}{ll} (m_p + m_L) \cdot derv_3 & = F_1 + F_2 - g \cdot (m_p + m_L) \\ J \cdot derw & = R_p \cdot (F_2 - F_1) \end{array}\right\| \rightarrow [F_1], [F_2] \\
F_1 = k_1 \cdot [e_1] \\
F_2 = k_2 \cdot [e_2]
\end{array}
\tag{2.14}
$$

The initial values of the variables are calculated solving the previous equations, giving:

$$
\begin{array}{l|l}
 & J = \dfrac{1}{2} \cdot 0.5 \cdot 0.3^2 = 0.0225 \\
\begin{array}{ll} k_1 & = 5 \\ k_2 & = 10 \\ m_p & = 0.5 \\ m_L & = 10 \\ R_p & = 0.3 \\ dere_1 & = 0 \\ dere_2 & = 0 \\ derv_3 & = 0 \\ derw & = 0 \end{array} &
\begin{array}{l}
v_1 = -0 = 0 \\
v_2 = -0 = 0 \\
\left.\begin{array}{ll} 0 & = v_3 - 0.3 \cdot w \\ 0 & = v_3 + 0.3 \cdot w \end{array}\right\| \rightarrow v_3 = 0, w = 0 \\
\left.\begin{array}{ll} (0.5 + 10) \cdot 0 & = F_1 + F_2 - 9.81 \cdot (0.5 + 10) \\ 0.0225 \cdot 0 & = 0.3 \cdot (F_2 - F_1) \end{array}\right\| \rightarrow \begin{array}{l} F_1 = 51.5025 \\ F_2 = 51.5025 \end{array} \\
e_1 = \dfrac{51.5025}{5} = 10.3005 \\
e_2 = \dfrac{51.5025}{10} = 5.15025
\end{array}
\end{array}
\tag{2.15}
$$

2.6 Solution to Task 4

The description in Modelica of the model is shown below. The simulation results are represented in Figure 2.4.

```
model SpringPulleyLoad2
  import SI = Modelica.SIunits;
  constant SI.Acceleration g=9.81;
  // Springs
  parameter SI.TranslationalSpringConstant k1(start=1,fixed=false);
  parameter SI.TranslationalSpringConstant k2(start=1,fixed=false);
  // Pulley
  parameter SI.Mass Mp=0.5;
  parameter SI.Radius Rp=0.3;
  parameter SI.MomentOfInertia J=0.5*Mp*Rp^2;
  // Load
  parameter SI.Mass ML0=10;
  SI.Mass ML;

  SI.Length e1;
  SI.Length e2;
  SI.Velocity v1;
  SI.Velocity v2;
  SI.Velocity v3;
  SI.Force F1;
  SI.Force F2;
  SI.AngularVelocity w;
equation
  // Spring 1
  F1 = k1*e1;
  der(e1) = -v1;
  // Spring 2
  F2 = k2*e2;
  der(e2) = -v2;
  // Pulley and load
  ML = if time < 5 then ML0 else 0;
  (Mp + ML)*der(v3) = F1 + F2 - g*(Mp + ML);
  v1 = v3 - Rp*w;
  v2 = v3 + Rp*w;
  J*der(w) = Rp*(F2 - F1);
initial equation
  der(e1) = 0;
  der(e2) = 0;
  der(v3) = 0;
  der(w) = 0;
  e1 = 3;
  e2 = 1.5;
  annotation (uses(Modelica(version="3.2.2")));
end SpringPulleyLoad2;
```

Modelica Code 2.3: Springs, pulley and load system - Task 4.

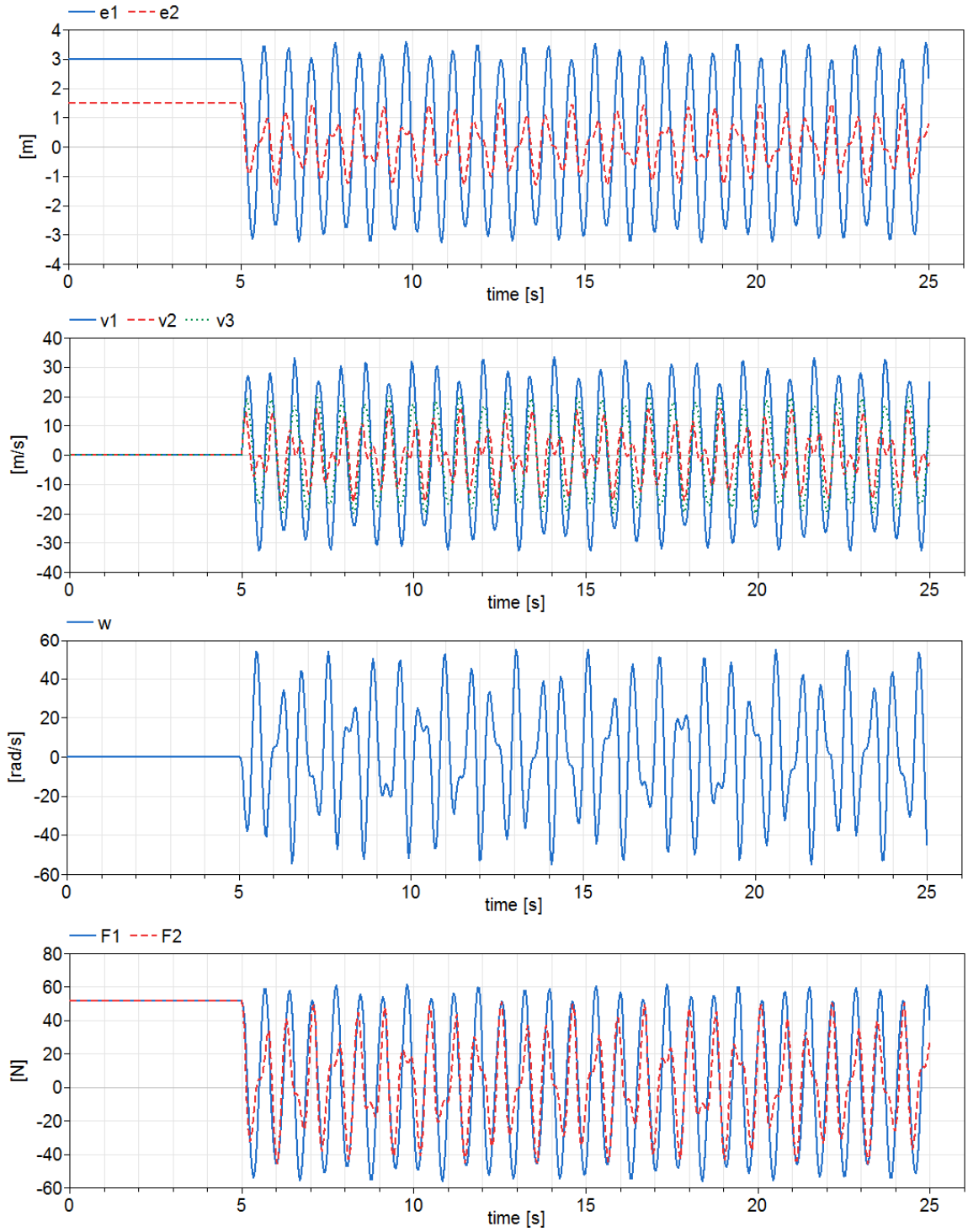


Figure 2.4: Simulation results of Task 4.

The initialization problem consists in solving the variables $\{k_1,\, k_2,\, m_p,\, m_L,\, R_p,\, e_1,\, e_2,\, v_1,\, v_2,\, v_3,\, w,\, F_1,\, F_2,\, J,\, dere_1,\, dere_2,\, derv_3,\, derw\}$ from the following set of equations. The sorted model, with the computational causality annotated, is shown below.

$$
\begin{array}{l}
[e_1] = 3 \\
[e_2] = 1.5 \\
[m_p] = 0.5 \\
[m_L] = 10 \\
[R_p] = 0.3 \\
[dere_1] = 0 \\
[dere_2] = 0 \\
[derv_3] = 0 \\
[derw] = 0
\end{array}
\qquad
\begin{array}{l}
[J] = \dfrac{1}{2} \cdot m_p \cdot R_p^2 \\
dere_1 = -\,[v_1] \\
dere_2 = -\,[v_2] \\
\left.\begin{array}{ll} v_1 & = v_3 - R_p \cdot w \\ v_2 & = v_3 + R_p \cdot w \end{array}\right\| \rightarrow [v_3]\,, [w] \\
\left.\begin{array}{ll} (m_p + m_L) \cdot derv_3 & = F_1 + F_2 - g \cdot (m_p + m_L) \\ J \cdot derw & = R_p \cdot (F_2 - F_1) \end{array}\right\| \rightarrow [F_1]\,, [F_2] \\
F_1 = [k_1] \cdot e_1 \\
F_2 = [k_2] \cdot e_2
\end{array}
\tag{2.16}
$$

The initial values of the variables are calculated solving the previous equations, giving:

$$
\begin{array}{l}
\\
e_1 \quad = 3 \\
e_2 \quad = 1.5 \\
m_p \quad = 0.5 \\
m_L \quad = 10 \\
R_p \quad = 0.3 \\
dere_1 \;= 0 \\
dere_2 \;= 0 \\
derv_3 \;= 0 \\
derw \quad = 0
\end{array}
\qquad
\begin{array}{l}
J = \dfrac{1}{2} \cdot 0.5 \cdot 0.3^2 = 0.0225 \\
v_1 = -0 = 0 \\
v_2 = -0 = 0 \\
\left.\begin{array}{ll} 0 & = v_3 - 0.3 \cdot w \\ 0 & = v_3 + 0.3 \cdot w \end{array}\right\| \rightarrow v_3 = 0, w = 0 \\
\left.\begin{array}{ll} (0.5 + 10) \cdot 0 & = F_1 + F_2 - 9.81 \cdot (0.5 + 10) \\ 0.0225 \cdot 0 & = 0.3 \cdot (F_2 - F_1) \end{array}\right\| \rightarrow \begin{array}{l} F_1 = 51.5025 \\ F_2 = 51.5025 \end{array} \\
k_1 = \dfrac{51.5025}{3} = 17.1675 \\
k_2 = \dfrac{51.5025}{1.5} = 34.335
\end{array}
\tag{2.17}
$$

Assignment 3

Bond graph library

Purpose of this assignment

- Getting started with bond graph modeling using Modelica.
- Design and implement model libraries in Modelica.

3.1 System description

Three mechanical systems are shown in Figures 3.1 – 3.3. The models are posed assuming that the friction force between two contacting surfaces is proportional to their relative velocity, the springs and dampers are massless and behave linearly, and the body masses are known constants. The model of the system depicted in Figure 3.3 was discussed in Assignment 1.

3.2 Tasks

The objective is to model these three systems applying the bond graph modeling methodology, and describe and simulate the models using Modelica.

1. A clear and concise introduction to bond graph modeling can be found in (Broenink 1999). Read this document. Write the bond graph models of the three systems represented in Figures 3.1, 3.2 and 3.3. Perform the causal analysis, writing the causal bond graphs.

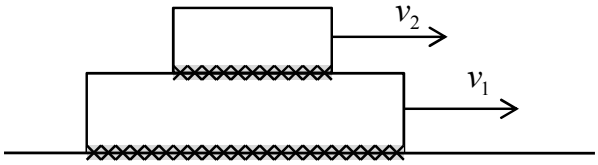


Figure 3.1: Movement of two bodies with friction.

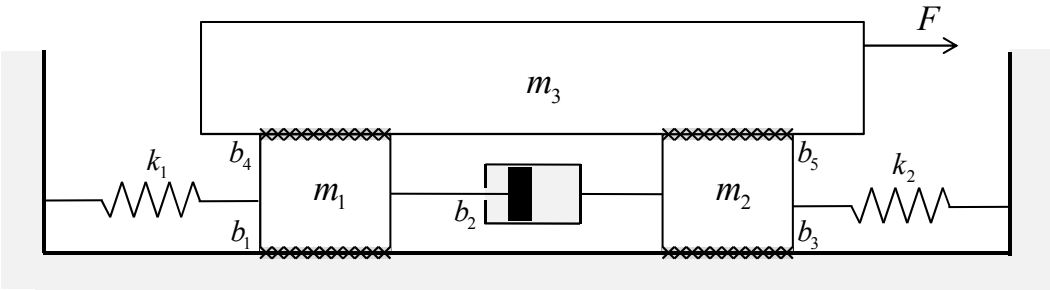


Figure 3.2: Movement of three bodies with friction, connected by damper and springs.

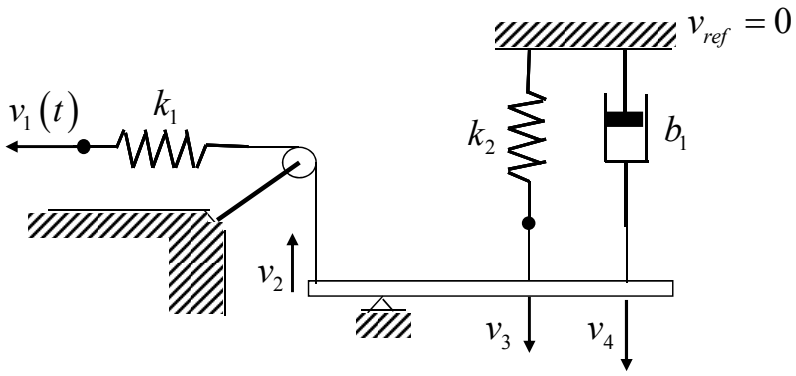


Figure 3.3: Springs, damper and lever.

2. Develop a Modelica library containing the following bond graph elements: port, bond, 0-junction, 1-junction, Se and Sf sources, C and I storage elements, resistor, transformer and gyrator. Using these elements, compose the models of the three mechanical systems shown in Figures 3.1 – 3.3.

3.3 Solution to Task 1

The bond graph model of the first system can be constructed as explained in Figure 3.4. The causal analysis is explained in Figure 3.5. The bond graph of the second and third systems are shown in Figures 3.6 and 3.7 respectively.

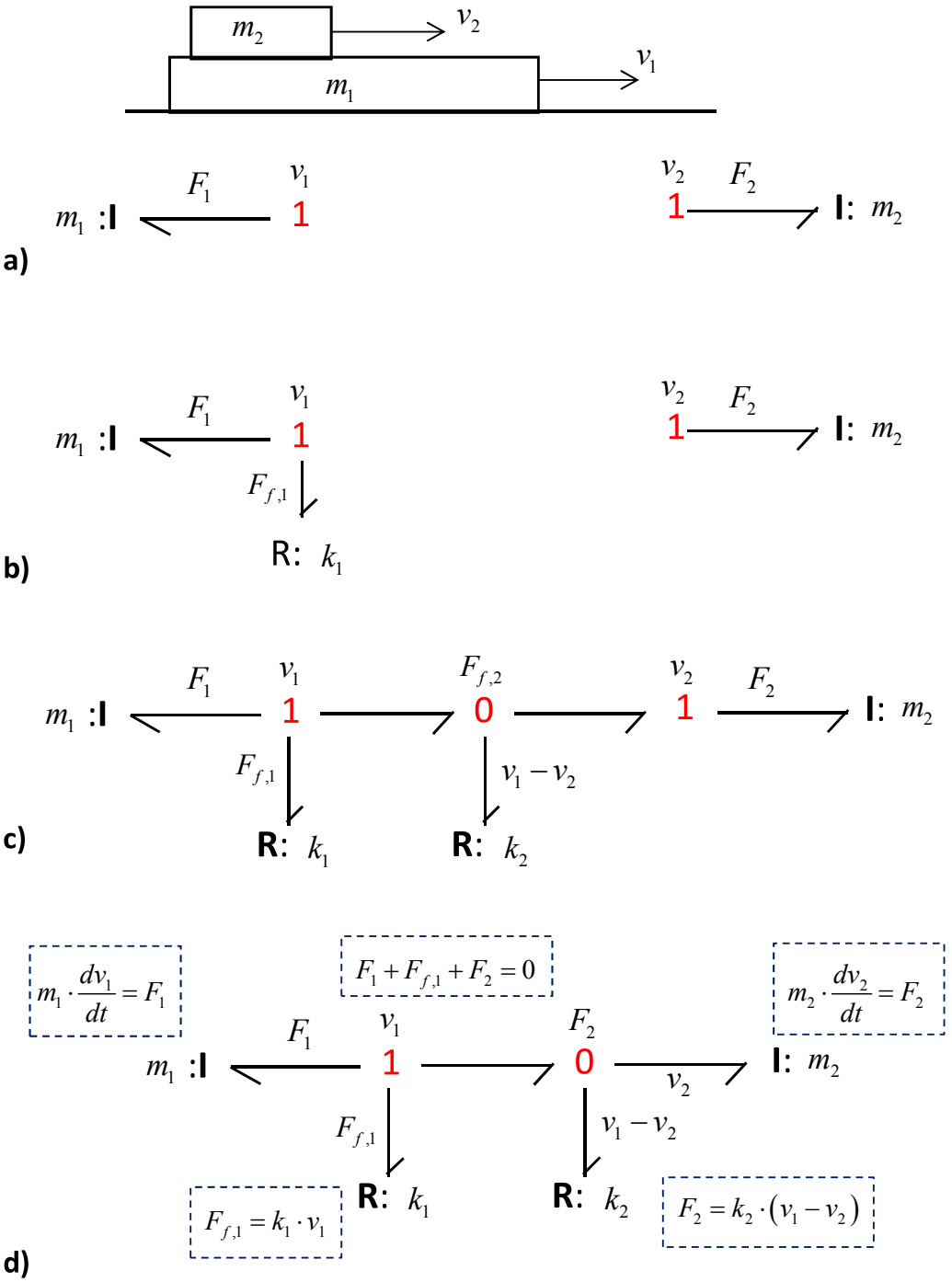


Figure 3.4: Mechanical system and bond graph model, which is derived as follows: a) body velocities are represented by 1-junctions and the body inertias by I-elements; b) energy is dissipated by friction between the first body and ground; c) and by friction between the two bodies; and d) the bond graph is simplified.

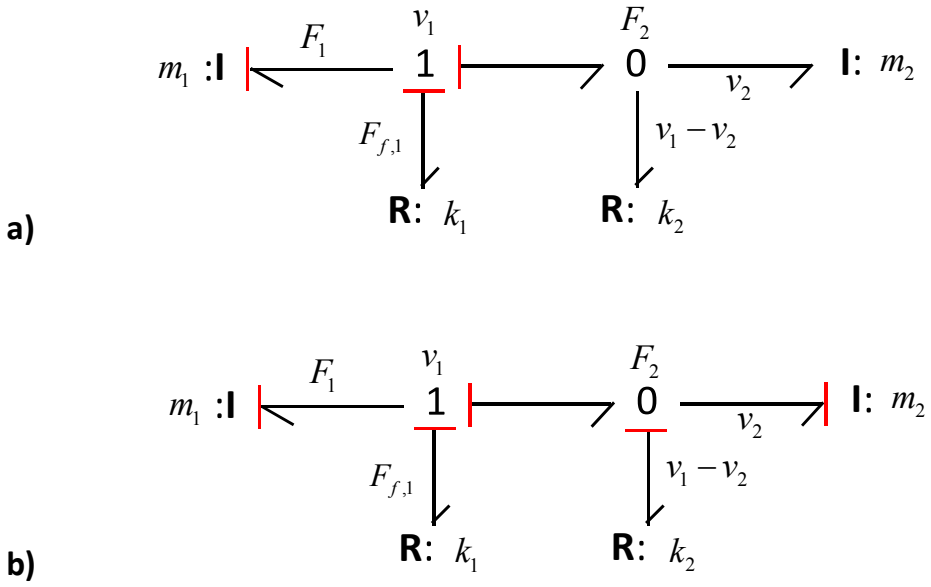


Figure 3.5: The causal analysis of the bond graph model is performed as follows: a) integrating causality is selected for the I-element that describes the inertia of the first body, which imposes the 1-junction causality; and b) integrating causality is selected for the other I-element, which determines the causality of the 0-junction.

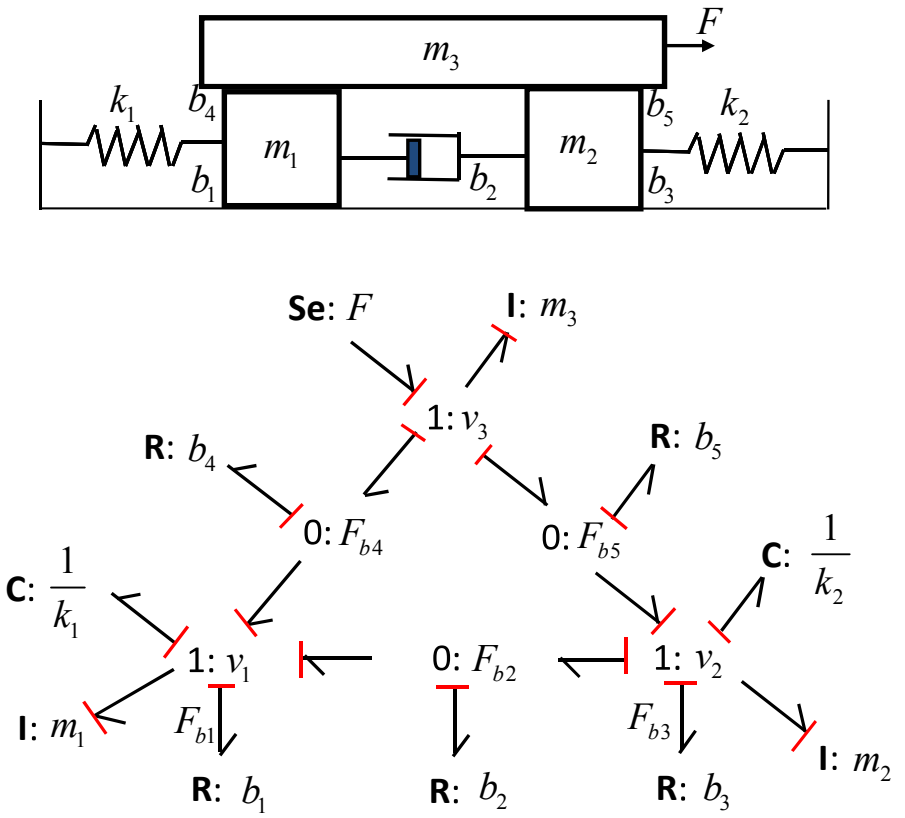


Figure 3.6: Mechanical system and causal bond graph.

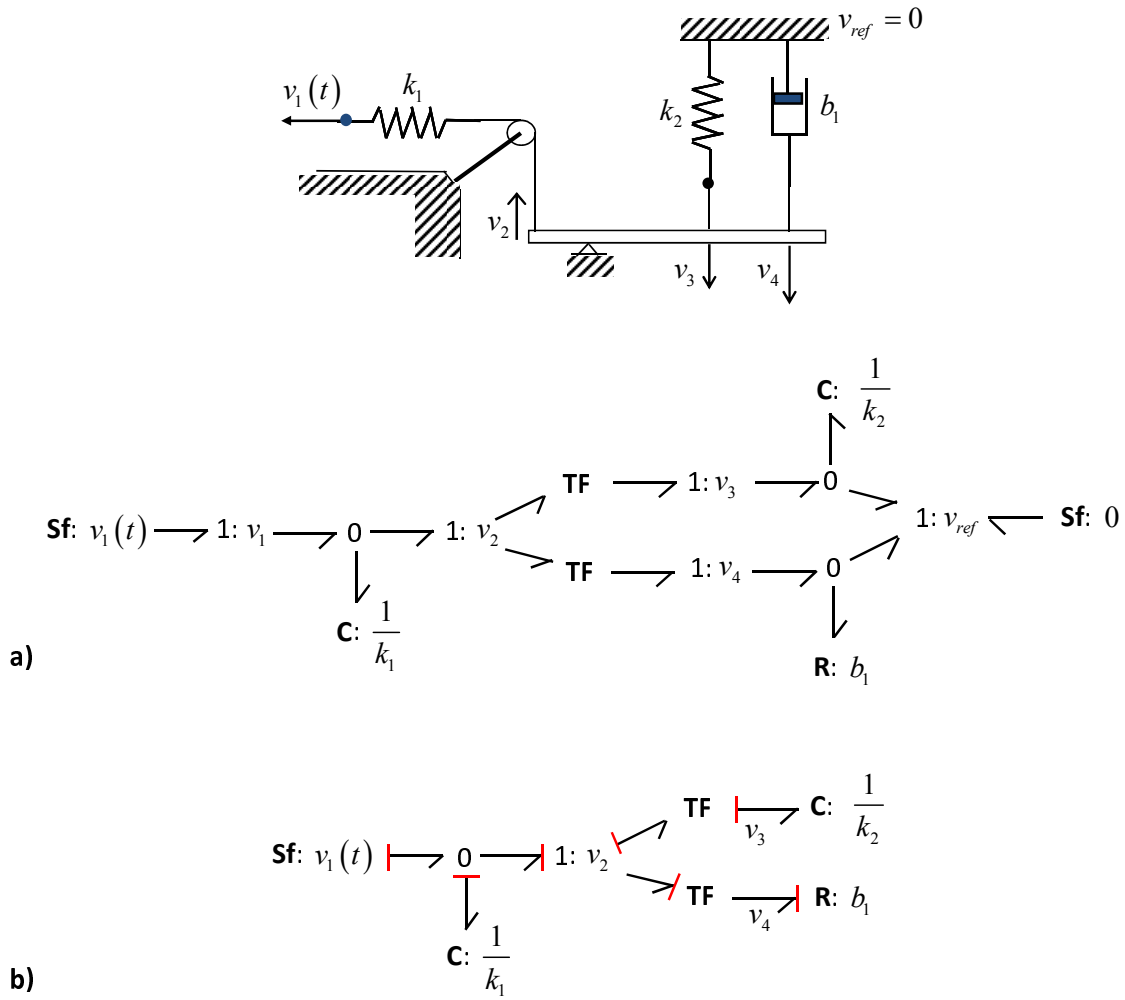


Figure 3.7: Mechanical system (above). Below: a) bond graph; and b) simplified, causal bond graph.

3.4 Solution to Task 2

The architecture of the library is shown in Figure 3.8(a). The library is composed of five packages. The `Interfaces` package contains the declaration of the port, and declarations of interfaces composed of different number of ports. The `Junction0` and `Junction1` packages contain the declaration of junctions with 2, 3, 4, 5, and 8 ports. The `Components` package contains the declaration of the bond graph elements. The `Examples` package contains the models of the three mechanical systems.

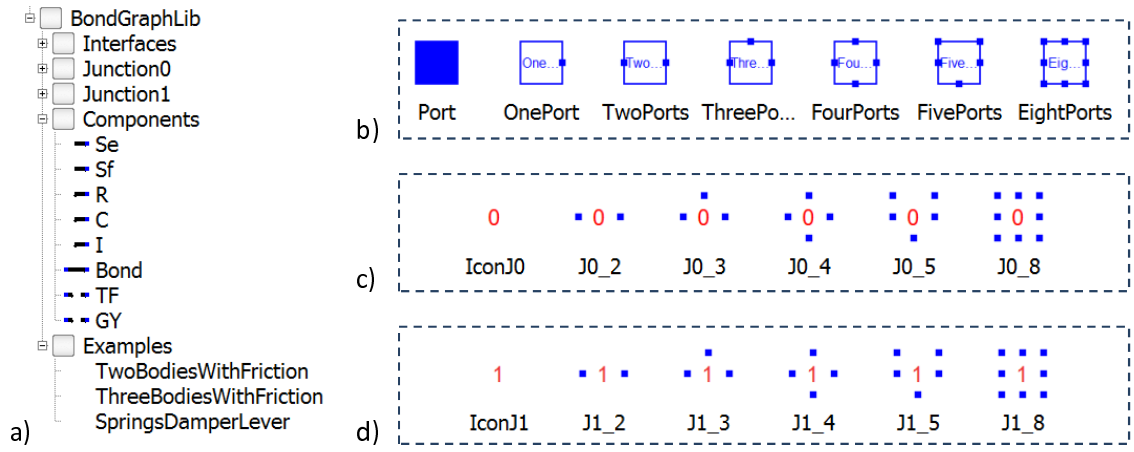
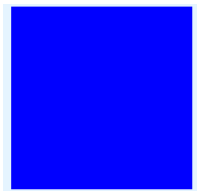


Figure 3.8: Bond graph library: a) architecture; b) *Interfaces* package; c) *Junction0* package; and d) *Junction1* package.

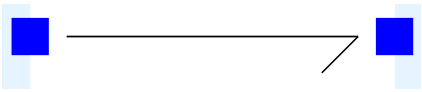
The `port` class is a connector that contains the declaration of three real variables: `e`, `f` and `sign`. The `e` and `f` variables are the effort and flow variables of the port, respectively. The `sign` variable has two possible values: $\{+1, -1\}$. It is used to indicate the orientation of the harpoon: $+1$ indicates the harpoon's head and -1 the harpoon's tail. The value of this variable is set in the elements (Se, Sf, R, C, I, Bond, TF and GY), and employed in the junctions. This imposses the following rule of use: the connection between two components or between two junctions is not allowed.

The icons and the code (with the annotations hidden) of some selected classes are shown in Figures 3.9 – 3.11. The complete code is shown in Modelica Code 3.1 – 3.19. The library has been developed using Dymola version 2017.

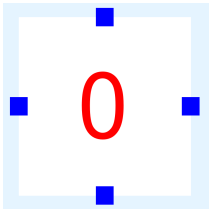


```
connector Port
   Real  e;
   Real  f;
   Real  sign;

end Port;
```

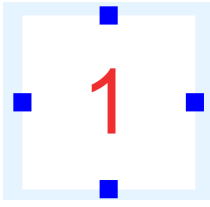
```
partial class TwoPorts

   Port port1
     annotation (Placement(transformation(
           extent={{-110,-10},{-90,10}})));
   Port port2
     annotation (Placement(transformation(
           extent={{110,-10},{90,10}})));
end TwoPorts;
```



```
model Bond
   extends Interfaces.TwoPorts;
equation
   port1.e = port2.e;
   port1.f = port2.f;
   port1.sign = -1;
   port2.sign = 1;

end Bond;
```

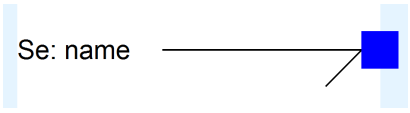



```
model J0_4
   extends Interfaces.FourPorts;
   extends IconJ0;
equation
   port1.e = port2.e;
   port1.e = port3.e;
   port1.e = port4.e;
   port1.sign*port1.f + port2.sign*port2.f +
   port3.sign*port3.f + port4.sign*port4.f = 0;
end J0_4;
```

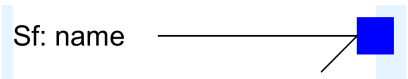



```
model J1_4
   extends Interfaces.FourPorts;
   extends IconJ1;
equation
   port1.f = port2.f;
   port1.f = port3.f;
   port1.f = port4.f;
   port1.sign*port1.e + port2.sign*port2.e +
   port3.sign*port3.e + port4.sign*port4.e = 0;
end J1_4;
```

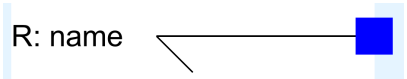
Figure 3.9: Icon and code without annotations of selected classes (1/3).



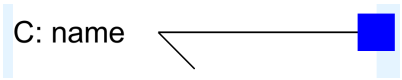

```
model Se
   extends Interfaces.OnePort;
   Real s;

equation
   port.e = s;
   port.sign = 1;

end Se;
```



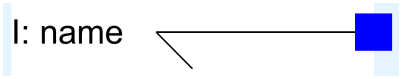

```
model Sf
   extends Interfaces.OnePort;
   Real s;
equation
   port.f = s;
   port.sign = 1;

end Sf;
```




```
model R
   extends Interfaces.OnePort;
   parameter Real R = 1;
equation
   port.e = R*port.f;
   port.sign = -1;

end R;
```




```
model C
   extends Interfaces.OnePort;
   parameter Real C = 1;
equation
   C*der(port.e) = port.f;
   port.sign = -1;

end C;
```




```
model I
   extends Interfaces.OnePort;
   parameter Real I = 1;
equation
   I*der(port.f) = port.e;
   port.sign = -1;

end I;
```

Figure 3.10: Icon and code without annotations of selected classes (2/3).


```
model TF
   extends Interfaces.TwoPorts;
   parameter Real m = 1;
equation
   m*port1.e = port2.e;
   port1.f = m*port2.f;
   port1.sign = -1;
   port2.sign = 1;

end TF;
```


```
model GY
   extends Interfaces.TwoPorts;
   parameter Real r = 1;
equation
   port1.e = r*port2.f;
   r*port1.f = port2.e;
   port1.sign = -1;
   port2.sign = 1;

end GY;
```

Figure 3.11: Icon and code without annotations of selected classes (3/3).

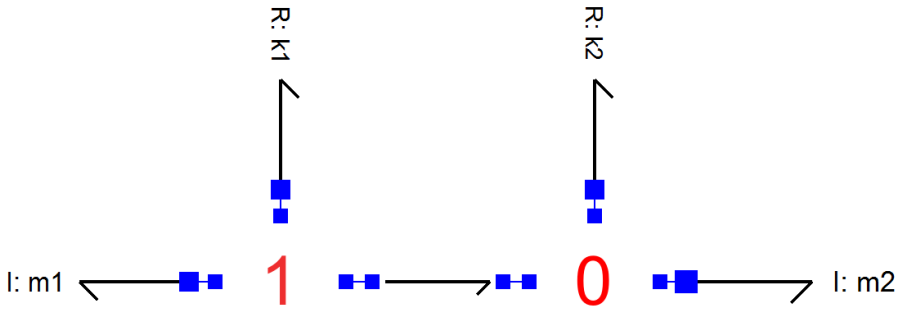


Figure 3.12: Diagram of the *BondGraphLib.Examples.TwoBodiesWithFriction* model.

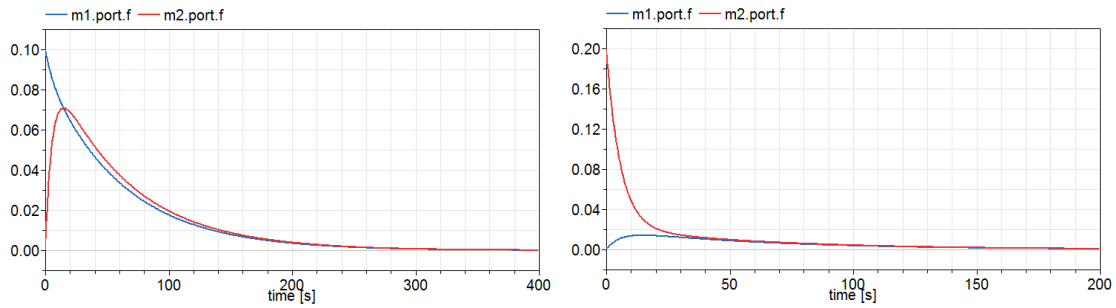


Figure 3.13: Simulation of the *TwoBodiesWithFriction* model (see Figure 3.12) with two different initial conditions: $v_1(0) = 0.1$ m/s, $v_2(0) = 0$ (left); $v_1(0) = 0$, $v_2(0) = 0.2$ m/s (right).

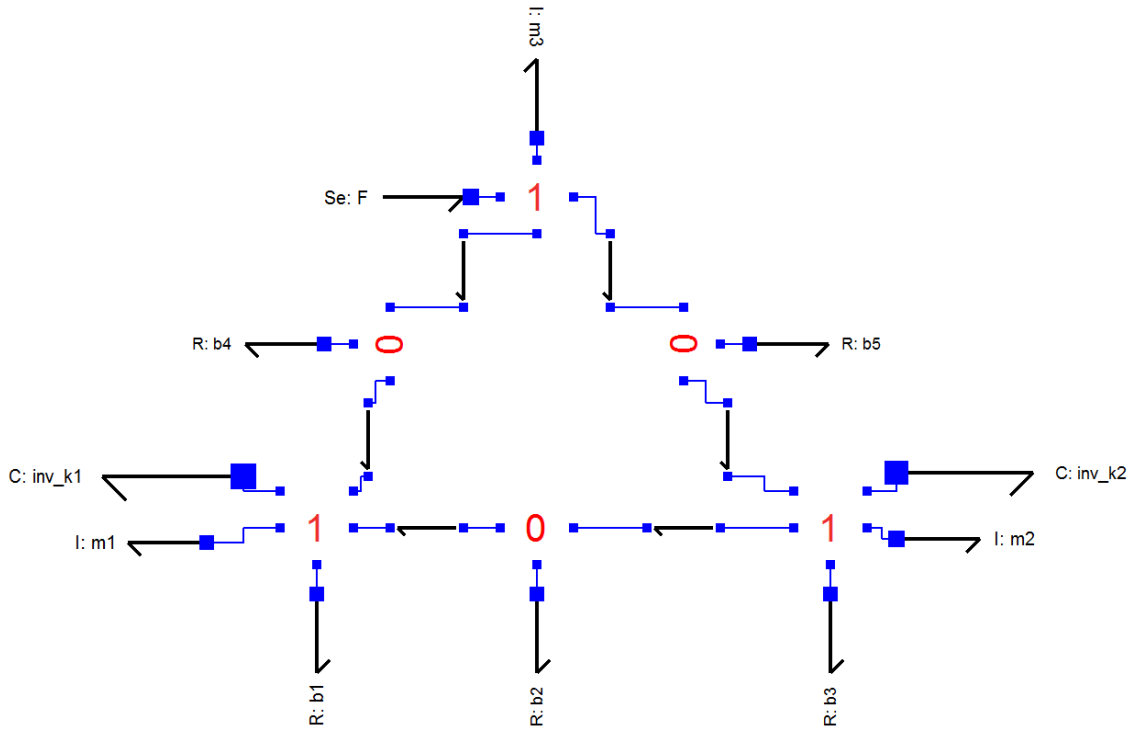


Figure 3.14: Diagram of the *BondGraphLib.Examples.ThreeBodiesWithFriction* model.

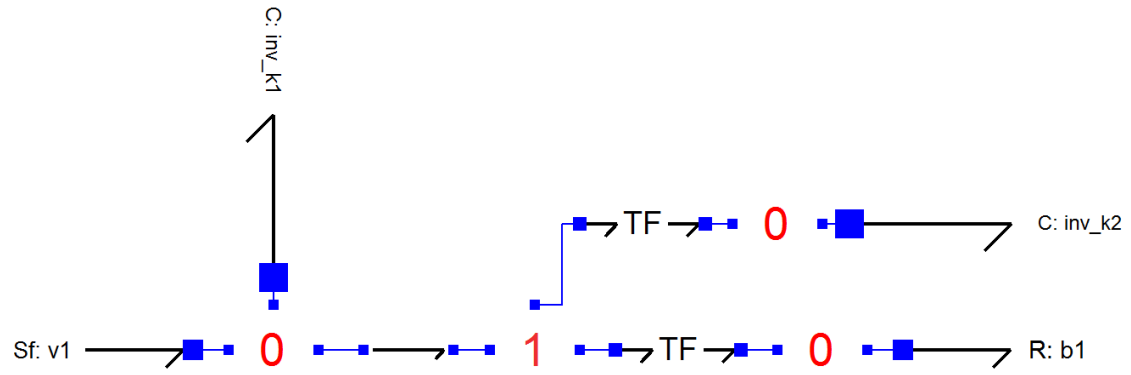


Figure 3.15: Diagram of the *BondGraphLib.Examples.SpringsDamperLever* model.

```
package BondGraphLib

  import SI = Modelica.SIunits;

  package Interfaces

    connector Port
      Real e;
      Real f;
      Real sign;

      annotation (Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,
                -100},{100,100}}), graphics={Rectangle(
              extent={{-100,100},{100,-100}},
              lineColor={0,0,255},
              fillColor={0,0,255},
              fillPattern=FillPattern.Solid)}), Diagram(coordinateSystem(
              preserveAspectRatio=true, extent={{-100,-100},{100,100}}),
            graphics={Rectangle(
              extent={{-40,40},{40,-40}},
              lineColor={0,0,255},
              fillColor={0,0,255},
              fillPattern=FillPattern.Solid), Text(
              extent={{-160,110},{40,50}},
              lineColor={0,0,255},
              textString="%name")}));
    end Port;

    partial class OnePort
      Port port
        annotation (Placement(transformation(extent={{110,-10},{90,10}})));
    end OnePort;

    partial class TwoPorts
      Port port1
        annotation (Placement(transformation(extent={{-110,-10},{-90,10}})));
      Port port2
        annotation (Placement(transformation(extent={{110,-10},{90,10}})));
    end TwoPorts;

    partial class ThreePorts
      Port port1
        annotation (Placement(transformation(extent={{-110,10},{-90,-10}})));
      Port port2
        annotation (Placement(transformation(extent={{-10,110},{10,90}})));
      Port port3
        annotation (Placement(transformation(extent={{90,10},{110,-10}})));
    end ThreePorts;
```

Modelica Code 3.1: The *BondGraphLib.Interfaces* package (1/2).

```
  partial class FourPorts
    Port port1
      annotation (Placement(transformation(extent={{-110,10},{-90,-10}})));
    Port port2
      annotation (Placement(transformation(extent={{-10,110},{10,90}})));
    Port port3
      annotation (Placement(transformation(extent={{90,10},{110,-10}})));
    Port port4
      annotation (Placement(transformation(extent={{-10,-90},{10,-110}})));
  end FourPorts;

  partial class FivePorts
    Port port1
      annotation (Placement(transformation(extent={{-110,10},{-90,-10}})));
    Port port2
      annotation (Placement(transformation(extent={{-110,110},{-90,90}})));
    Port port3
      annotation (Placement(transformation(extent={{90,110},{110,90}})));
    Port port4
      annotation (Placement(transformation(extent={{90,10},{110,-10}})));
    Port port5
      annotation (Placement(transformation(extent={{-10,-90},{10,-110}})));
  end FivePorts;

  partial class EightPorts
    Port port1
      annotation (Placement(transformation(extent={{-110,10},{-90,-10}})));
    Port port2
      annotation (Placement(transformation(extent={{-110,110},{-90,90}})));
    Port port3
      annotation (Placement(transformation(extent={{-10,110},{10,90}})));
    Port port4
      annotation (Placement(transformation(extent={{90,110},{110,90}})));
    Port port5
      annotation (Placement(transformation(extent={{90,10},{110,-10}})));
    Port port6
      annotation (Placement(transformation(extent={{90,-90},{110,-110}})));
    Port port7
      annotation (Placement(transformation(extent={{-10,-90},{10,-110}})));
    Port port8
      annotation (Placement(transformation(extent={{-110,-90},{-90,-110}})));
  end EightPorts;


end Interfaces;
```

Modelica Code 3.2: The *BondGraphLib.Interfaces* package (2/2).

```
package Junction0

  partial model IconJ0
    annotation (Icon(graphics={Text(
                extent={{-62,60},{58,-60}},
                lineColor={255,0,0},
                lineThickness=0.5,
                fillColor={0,0,255},
                fillPattern=FillPattern.Solid,
                textString="0")}), Diagram(graphics={Text(
                extent={{-62,60},{58,-60}},
                lineColor={255,0,0},
                lineThickness=0.5,
                fillColor={0,0,255},
                fillPattern=FillPattern.Solid,
                textString="0")}));
  end IconJ0;

  model J0_2
    extends Interfaces.TwoPorts;
    extends IconJ0;
  equation
    port1.e = port2.e;
    port1.sign*port1.f + port2.sign*port2.f = 0;
  end J0_2;

  model J0_3
    extends Interfaces.ThreePorts;
    extends IconJ0;
  equation
    port1.e = port2.e;
    port1.e = port3.e;
    port1.sign*port1.f + port2.sign*port2.f + port3.sign*port3.f = 0;
  end J0_3;

  model J0_4
    extends Interfaces.FourPorts;
    extends IconJ0;
  equation
    port1.e = port2.e;
    port1.e = port3.e;
    port1.e = port4.e;
    port1.sign*port1.f + port2.sign*port2.f + port3.sign*port3.f + port4.sign
      *port4.f = 0;
  end J0_4;
```

Modelica Code 3.3: The *BondGraphLib.Junction0* package (1/2).

```
  model J0_5
    extends Interfaces.FivePorts;
    extends IconJ0;
  equation
    port1.e = port2.e;
    port1.e = port3.e;
    port1.e = port4.e;
    port1.e = port5.e;
    port1.sign*port1.f + port2.sign*port2.f + port3.sign*port3.f + port4.sign
      *port4.f + port5.sign*port5.f = 0;
  end J0_5;

  model J0_8
    extends Interfaces.EightPorts;
    extends IconJ0;
  equation
    port1.e = port2.e;
    port1.e = port3.e;
    port1.e = port4.e;
    port1.e = port5.e;
    port1.e = port6.e;
    port1.e = port7.e;
    port1.e = port8.e;
    port1.sign*port1.f + port2.sign*port2.f + port3.sign*port3.f + port4.sign
      *port4.f + port5.sign*port5.f + port5.sign*port6.f + port7.sign*port7.f
       + port8.sign*port8.f = 0;
  end J0_8;

end Junction0;
```

Modelica Code 3.4: The *BondGraphLib.Junction0* package (2/2).

```
package Junction1

  partial model IconJ1
    annotation (Icon(graphics={Text(
                extent={{-62,60},{58,-60}},
                lineColor={238,46,47},
                lineThickness=0.5,
                fillColor={0,0,255},
                fillPattern=FillPattern.Solid,
                textString="1")}), Diagram(graphics={Text(
                extent={{-62,60},{58,-60}},
                lineColor={238,46,47},
                lineThickness=0.5,
                fillColor={0,0,255},
                fillPattern=FillPattern.Solid,
                textString="1")}));
  end IconJ1;

  model J1_2
    extends Interfaces.TwoPorts;
    extends IconJ1;
  equation
    port1.f = port2.f;
    port1.sign*port1.e + port2.sign*port2.e = 0;
  end J1_2;

  model J1_3
    extends Interfaces.ThreePorts;
    extends IconJ1;
  equation
    port1.f = port2.f;
    port1.f = port3.f;
    port1.sign*port1.e + port2.sign*port2.e + port3.sign*port3.e = 0;
  end J1_3;

  model J1_4
    extends Interfaces.FourPorts;
    extends IconJ1;
  equation
    port1.f = port2.f;
    port1.f = port3.f;
    port1.f = port4.f;
    port1.sign*port1.e + port2.sign*port2.e + port3.sign*port3.e + port4.sign
      *port4.e = 0;
  end J1_4;
```

Modelica Code 3.5: The *BondGraphLib.Junction1* package (1/2).

```
  model J1_5
    extends Interfaces.FivePorts;
    extends IconJ1;
  equation
    port1.f = port2.f;
    port1.f = port3.f;
    port1.f = port4.f;
    port1.f = port5.f;
    port1.sign*port1.e + port2.sign*port2.e + port3.sign*port3.e + port4.sign
      *port4.e + port5.sign*port5.e = 0;
  end J1_5;

  model J1_8
    extends Interfaces.EightPorts;
    extends IconJ1;
  equation
    port1.f = port2.f;
    port1.f = port3.f;
    port1.f = port4.f;
    port1.f = port5.f;
    port1.f = port6.f;
    port1.f = port7.f;
    port1.f = port8.f;
    port1.sign*port1.e + port2.sign*port2.e + port3.sign*port3.e + port4.sign
      *port4.e + port5.sign*port5.e + port6.sign*port6.e + port7.sign*port7.e
       + port8.sign*port8.e = 0;
  end J1_8;

end Junction1;
```

Modelica Code 3.6: The *BondGraphLib.Junction1* package (2/2).

```
package Components

  model Se
    extends Interfaces.OnePort;
    Real s;
  equation
    port.e = s;
    port.sign = 1;

    annotation (Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,
              -100},{100,100}}), graphics={Line(
                points={{90,0},{70,-20}},
                color={0,0,0},
                thickness=0.5),Line(
                points={{-20,0},{96,0}},
                color={0,0,0},
                thickness=0.5),Text(
                extent={{-100,60},{-40,-60}},
                lineColor={0,0,0},
                textString="Se:%name",
                horizontalAlignment=TextAlignment.Left)}), Diagram(
          coordinateSystem(preserveAspectRatio=true, extent={{-100,-100},{100,
              100}}), graphics={Line(
                points={{96,0},{76,-20}},
                color={0,0,0},
                thickness=0.5),Line(
                points={{-20,0},{96,0}},
                color={0,0,0},
                thickness=0.5),Text(
                extent={{-260,60},{-40,-60}},
                lineColor={0,0,0},
                horizontalAlignment=TextAlignment.Left,
                textString="Se:%name")}));
  end Se;
```

Modelica Code 3.7: The *BondGraphLib.Components.Se* model.

```
model Sf
  extends Interfaces.OnePort;
  Real s;
equation
  port.f = s;
  port.sign = 1;

  annotation (Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,
            -100},{100,100}}), graphics={Line(
              points={{90,0},{70,-20}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{-20,0},{96,0}},
              color={0,0,0},
              thickness=0.5),Text(
              extent={{-100,60},{-40,-60}},
              lineColor={0,0,0},
              textString="Sf:%name",
              horizontalAlignment=TextAlignment.Left)}), Diagram(
          coordinateSystem(preserveAspectRatio=true, extent={{-100,-100},{100,
              100}}), graphics={Line(
              points={{96,0},{76,-20}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{-20,0},{96,0}},
              color={0,0,0},
              thickness=0.5),Text(
              extent={{-260,60},{-40,-60}},
              lineColor={0,0,0},
              horizontalAlignment=TextAlignment.Left,
              textString="Sf:%name")}));
end Sf;
```

Modelica Code 3.8: The *BondGraphLib.Components.Sf* model.

```
model R
  extends Interfaces.OnePort;
  parameter Real R=1;
equation
  port.e = R*port.f;
  port.sign = -1;

  annotation (Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,
            -100},{100,100}}), graphics={Line(
              points={{-20,0},{0,-20}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{-20,0},{96,0}},
              color={0,0,0},
              thickness=0.5),Text(
              extent={{-100,60},{-40,-60}},
              lineColor={0,0,0},
              textString="R: %name",
              horizontalAlignment=TextAlignment.Left)}), Diagram(
        coordinateSystem(preserveAspectRatio=true, extent={{-100,-100},{100,
            100}}), graphics={Line(
              points={{-20,0},{0,-20}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{-20,0},{96,0}},
              color={0,0,0},
              thickness=0.5),Text(
              extent={{-180,60},{-40,-60}},
              lineColor={0,0,0},
              horizontalAlignment=TextAlignment.Left,
              textString="R: %name")}));
end R;
```

Modelica Code 3.9: The *BondGraphLib.Components.R* model.

```
model C
  extends Interfaces.OnePort;
  parameter Real C=1;
equation
  C*der(port.e) = port.f;
  port.sign = -1;

  annotation (Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,
            -100},{100,100}}), graphics={Line(
              points={{-20,0},{0,-20}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{-20,0},{96,0}},
              color={0,0,0},
              thickness=0.5),Text(
              extent={{-100,60},{-40,-60}},
              lineColor={0,0,0},
              textString="C: %name",
              horizontalAlignment=TextAlignment.Left)}), Diagram(
          coordinateSystem(preserveAspectRatio=true, extent={{-100,-100},{100,
              100}}), graphics={Line(
              points={{-20,0},{0,-20}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{-20,0},{96,0}},
              color={0,0,0},
              thickness=0.5),Text(
              extent={{-180,60},{-40,-60}},
              lineColor={0,0,0},
              horizontalAlignment=TextAlignment.Left,
              textString="C: %name")}));
end C;
```

Modelica Code 3.10: The *BondGraphLib.Components.C* model.

```
model I
  extends Interfaces.OnePort;
  parameter Real I=1;
equation
  I*der(port.f) = port.e;
  port.sign = -1;

  annotation (Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,
            -100},{100,100}}), graphics={Line(
              points={{-20,0},{0,-20}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{-20,0},{96,0}},
              color={0,0,0},
              thickness=0.5),Text(
              extent={{-100,60},{-40,-60}},
              lineColor={0,0,0},
              textString="I: %name",
              horizontalAlignment=TextAlignment.Left)}), Diagram(
        coordinateSystem(preserveAspectRatio=true, extent={{-100,-100},{100,
            100}}), graphics={Line(
              points={{-20,0},{0,-20}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{-20,0},{96,0}},
              color={0,0,0},
              thickness=0.5),Text(
              extent={{-180,80},{-40,-80}},
              lineColor={0,0,0},
              horizontalAlignment=TextAlignment.Left,
              textString="I: %name")}));
end I;
```

Modelica Code 3.11: The *BondGraphLib.Components.I* model.

```
model Bond
  extends Interfaces.TwoPorts;
equation
  port1.e = port2.e;
  port1.f = port2.f;
  port1.sign = -1;
  port2.sign = 1;

  annotation (Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,
            -100},{100,100}}), graphics={Line(
              points={{-80,0},{80,0}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{60,-20},{80,0}},
              color={0,0,0},
              thickness=0.5)}), Diagram(coordinateSystem(
          preserveAspectRatio=true, extent={{-100,-100},{100,100}}),
        graphics={Line(
              points={{-80,0},{80,0}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{60,-20},{80,0}},
              color={0,0,0},
              thickness=0.5)}));
end Bond;
```

Modelica Code 3.12: The *BondGraphLib.Components.Bond* model.

```
model TF
  extends Interfaces.TwoPorts;
  parameter Real m=1;
equation
  m*port1.e = port2.e;
  port1.f = m*port2.f;
  port1.sign = -1;
  port2.sign = 1;

  annotation (Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,
            -100},{100,100}}), graphics={Line(
              points={{-60,-20},{-40,0}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{40,0},{96,0}},
              color={0,0,0},
              thickness=0.5),Text(
              extent={{-30,60},{30,-60}},
              lineColor={0,0,0},
              textString="TF",
              horizontalAlignment=TextAlignment.Left),Line(
              points={{-96,0},{-40,0}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{70,-20},{90,0}},
              color={0,0,0},
              thickness=0.5)}), Diagram(coordinateSystem(
          preserveAspectRatio=true, extent={{-100,-100},{100,100}}),
        graphics={Line(
              points={{-60,-20},{-40,0}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{40,0},{96,0}},
              color={0,0,0},
              thickness=0.5),Text(
              extent={{-30,60},{30,-60}},
              lineColor={0,0,0},
              textString="TF",
              horizontalAlignment=TextAlignment.Left),Line(
              points={{-96,0},{-40,0}},
              color={0,0,0},
              thickness=0.5),Line(
              points={{76,-20},{96,0}},
              color={0,0,0},
              thickness=0.5)}));
end TF;
```

Modelica Code 3.13: The *BondGraphLib.Components.TF* model.

```
  model GY
    extends Interfaces.TwoPorts;
    parameter Real r=1;
  equation
    port1.e = r*port2.f;
    r*port1.f = port2.e;
    port1.sign = -1;
    port2.sign = 1;

    annotation (Icon(coordinateSystem(preserveAspectRatio=true, extent={{-100,
              -100},{100,100}}), graphics={Line(
                points={{-60,-20},{-40,0}},
                color={0,0,0},
                thickness=0.5),Line(
                points={{40,0},{96,0}},
                color={0,0,0},
                thickness=0.5),Text(
                extent={{-30,60},{30,-60}},
                lineColor={0,0,0},
                textString="GY",
                horizontalAlignment=TextAlignment.Left),Line(
                points={{-96,0},{-40,0}},
                color={0,0,0},
                thickness=0.5),Line(
                points={{70,-20},{90,0}},
                color={0,0,0},
                thickness=0.5)}), Diagram(coordinateSystem(
            preserveAspectRatio=true, extent={{-100,-100},{100,100}}),
          graphics={Line(
                points={{-60,-20},{-40,0}},
                color={0,0,0},
                thickness=0.5),Line(
                points={{40,0},{96,0}},
                color={0,0,0},
                thickness=0.5),Text(
                extent={{-30,60},{30,-60}},
                lineColor={0,0,0},
                textString="GY",
                horizontalAlignment=TextAlignment.Left),Line(
                points={{-96,0},{-40,0}},
                color={0,0,0},
                thickness=0.5),Line(
                points={{76,-20},{96,0}},
                color={0,0,0},
                thickness=0.5)}));
  end GY;


end Components;
```

Modelica Code 3.14: The *BondGraphLib.Components.GY* model.

```
package Examples

  model TwoBodiesWithFriction

    Components.I m1(I=100)
      annotation (Placement(transformation(extent={{-64,-4},{-36,24}})));
    Junction1.J1_3 junction1_1
      annotation (Placement(transformation(extent={{-32,0},{-12,20}})));
    Components.R k1(R=1.8) annotation (Placement(transformation(
          extent={{14,-14},{-14,14}},
          rotation=90,
          origin={-22,38})));
    Junction0.J0_3 junction0_1
      annotation (Placement(transformation(extent={{16,0},{36,20}})));
    Components.R k2(R=1.6) annotation (Placement(transformation(
          extent={{14,-14},{-14,14}},
          rotation=90,
          origin={26,38})));
    Components.Bond bond
      annotation (Placement(transformation(extent={{-8,0},{12,20}})));
    Components.I m2(I=10)
      annotation (Placement(transformation(extent={{72,-6},{40,26}})));

  equation
    connect(m1.port, junction1_1.port1) annotation (Line(points={{-36,10},{-34,
            10},{-32,10}}, color={0,0,255}));
    connect(junction1_1.port3, bond.port1)
      annotation (Line(points={{-12,10},{-8,10}}, color={0,0,255}));
    connect(bond.port2, junction0_1.port1)
      annotation (Line(points={{12,10},{12,10},{16,10}}, color={0,0,255}));
    connect(junction0_1.port3, m2.port)
      annotation (Line(points={{36,10},{38,10},{40,10}}, color={0,0,255}));
    connect(junction0_1.port2, k2.port)
      annotation (Line(points={{26,20},{26,22},{26,24}}, color={0,0,255}));
    connect(junction1_1.port2, k1.port) annotation (Line(points={{-22,20},{-22,
            22},{-22,24}}, color={0,0,255}));
    annotation (Icon(coordinateSystem(preserveAspectRatio=false)), Diagram(
          coordinateSystem(preserveAspectRatio=false)));
  end TwoBodiesWithFriction;
```

Modelica Code 3.15: The *TwoBodiesWithFriction* model.

```
model ThreeBodiesWithFriction

  BondGraphLib.Junction1.J1_4 j1_4_1
    annotation (Placement(transformation(extent={{-16,40},{4,60}})));
  BondGraphLib.Junction0.J0_3 j0_3_1 annotation (Placement(transformation(
        extent={{-10,-10},{10,10}}, rotation=-90, origin={34,10})));
  BondGraphLib.Junction1.J1_5 j1_5_1
    annotation (Placement(transformation(extent={{64,-50},{84,-30}})));
  BondGraphLib.Junction0.J0_3 j0_3_2
    annotation (Placement(transformation(extent={{-16,-30},{4,-50}})));
  BondGraphLib.Junction1.J1_5 j1_5_2
    annotation (Placement(transformation(extent={{-76,-50},{-56,-30}})));
  BondGraphLib.Junction0.J0_3 j0_3_3 annotation (Placement(transformation(
        extent={{-10,-10},{10,10}}, rotation=90, origin={-46,10})));
  BondGraphLib.Components.Se F
    annotation (Placement(transformation(extent={{-64,30},{-24,70}})));
  BondGraphLib.Components.I m3(I=15) annotation (Placement(transformation(
        extent={{-18,-18},{18,18}}, rotation=-90, origin={-6,84})));
  BondGraphLib.Components.I m2(I=10) annotation (Placement(transformation(
        extent={{-19,19},{19,-19}}, rotation=180, origin={111,-43})));
  BondGraphLib.Components.I m1(I=100) annotation (Placement(transformation(
        extent={{18,18},{-18,-18}}, rotation=180, origin={-114,-44})));
  BondGraphLib.Components.C inv_k1(C=1/1.2)
    annotation (Placement(transformation(extent={{-150,-58},{-86,6}})));
  BondGraphLib.Components.C inv_k2(C=1/1.2)
    annotation (Placement(transformation(extent={{154,-56},{92,6}})));
  BondGraphLib.Components.R b1(R=1.8) annotation (Placement(transformation(
        extent={{-18,-18},{18,18}}, rotation=90, origin={-66,-76})));
  BondGraphLib.Components.R b3(R=1.8) annotation (Placement(transformation(
        extent={{-18,-18},{18,18}}, rotation=90, origin={74,-76})));
  BondGraphLib.Components.R b2(R=3.2) annotation (Placement(transformation(
        extent={{-18,-18},{18,18}}, rotation=90, origin={-6,-76})));
  BondGraphLib.Components.R b5(R=1.6) annotation (Placement(transformation(
        extent={{-18,18},{18,-18}}, rotation=180, origin={70,10})));
  BondGraphLib.Components.R b4(R=1.6) annotation (Placement(transformation(
        extent={{18,18},{-18,-18}}, rotation=180, origin={-82,10})));
  BondGraphLib.Components.Bond bond3 annotation (Placement(transformation(
        extent={{-10,-10},{10,10}}, rotation=-90, origin={-52,-16})));
  BondGraphLib.Components.Bond bond5 annotation (Placement(transformation(
        extent={{-10,-10},{10,10}}, rotation=-90, origin={-26,30})));
  BondGraphLib.Components.Bond bond1 annotation (Placement(transformation(
        extent={{-10,-10},{10,10}}, rotation=-90, origin={14,30})));
  BondGraphLib.Components.Bond bond2 annotation (Placement(transformation(
        extent={{-10,-10},{10,10}}, rotation=-90, origin={46,-16})));
  BondGraphLib.Components.Bond bond4 annotation (Placement(transformation(
        extent={{-10,10},{10,-10}}, rotation=180, origin={-36,-40})));
  BondGraphLib.Components.Bond bond6 annotation (Placement(transformation(
        extent={{-10,10},{10,-10}}, rotation=180, origin={34,-40})));
  parameter SI.Force F0=10 "Amplitude of the external force";
  parameter SI.AngularFrequency w=0.1 "Frequency of the external force";
```

Modelica Code 3.16: The *ThreeBodiesWithFriction* model (1/2).

```
equation
  // External force
  F.s = if time < 200 then F0*sin(w*time) else 0;
  connect(F.port, j1_4_1.port1) annotation (Line(points={{-24,50},{-20,50},
          {-16,50}}, color={0,0,255}));
  connect(m3.port, j1_4_1.port2)
    annotation (Line(points={{-6,66},{-6,66},{-6,60}}, color={0,0,255}));
  connect(inv_k1.port, j1_5_2.port2) annotation (Line(points={{-86,-26},{-86,
          -30},{-76,-30}}, color={0,0,255}));
  connect(j1_5_1.port3, inv_k2.port) annotation (Line(points={{84,-30},{92,
          -30},{92,-25}}, color={0,0,255}));
  connect(j1_5_1.port4, m2.port) annotation (Line(points={{84,-40},{88,-40},
          {88,-43},{92,-43}}, color={0,0,255}));
  connect(m1.port, j1_5_2.port1) annotation (Line(points={{-96,-44},{-86,-44},
          {-86,-40},{-76,-40}}, color={0,0,255}));
  connect(j1_4_1.port4, bond5.port1)
    annotation (Line(points={{-6,40},{-16,40},{-26,40}}, color={0,0,255}));
  connect(j1_4_1.port3, bond1.port1) annotation (Line(points={{4,50},{10,50},
          {10,40},{14,40}}, color={0,0,255}));
  connect(bond1.port2, j0_3_1.port1)
    annotation (Line(points={{14,20},{24,20},{34,20}}, color={0,0,255}));
  connect(j0_3_1.port3, bond2.port1) annotation (Line(points={{34,0},{40,0},
          {40,-6},{46,-6}}, color={0,0,255}));
  connect(bond2.port2, j1_5_1.port2) annotation (Line(points={{46,-26},{56,
          -26},{56,-30},{64,-30}}, color={0,0,255}));
  connect(bond6.port1, j1_5_1.port1) annotation (Line(points={{44,-40},{54,
          -40},{64,-40}}, color={0,0,255}));
  connect(j1_5_1.port5, b3.port)
    annotation (Line(points={{74,-50},{74,-58}}, color={0,0,255}));
  connect(j0_3_2.port3, bond6.port2)
    annotation (Line(points={{4,-40},{14,-40},{24,-40}}, color={0,0,255}));
  connect(j0_3_2.port2, b2.port) annotation (Line(points={{-6,-50},{-6,-54},
          {-6,-58}}, color={0,0,255}));
  connect(bond4.port1, j0_3_2.port1) annotation (Line(points={{-26,-40},{-22,
          -40},{-16,-40}}, color={0,0,255}));
  connect(j1_5_2.port4, bond4.port2) annotation (Line(points={{-56,-40},{-51,
          -40},{-46,-40}}, color={0,0,255}));
  connect(j1_5_2.port5, b1.port) annotation (Line(points={{-66,-50},{-66,-54},
          {-66,-58}}, color={0,0,255}));
  connect(j1_5_2.port3, bond3.port2) annotation (Line(points={{-56,-30},{-54,
          -30},{-54,-26},{-52,-26}}, color={0,0,255}));
  connect(bond3.port1, j0_3_3.port1) annotation (Line(points={{-52,-6},{-50,
          -6},{-50,0},{-46,0}}, color={0,0,255}));
  connect(b4.port, j0_3_3.port2) annotation (Line(points={{-64,10},{-60,10},
          {-56,10}}, color={0,0,255}));
  connect(j0_3_3.port3, bond5.port2) annotation (Line(points={{-46,20},{-36,
          20},{-26,20}}, color={0,0,255}));
  connect(j0_3_1.port2, b5.port)
    annotation (Line(points={{44,10},{48,10},{52,10}}, color={0,0,255}));
  annotation (Icon(coordinateSystem(preserveAspectRatio=false)), Diagram(
        coordinateSystem(preserveAspectRatio=false, extent={{-160,-160},{
          160,160}})));
end ThreeBodiesWithFriction;
```

Modelica Code 3.17: The *ThreeBodiesWithFriction* model (2/2).

```
model SpringsDamperLever

  // Parameters of the v1 velocity
  parameter SI.Velocity V10=0.01;
  parameter SI.AngularFrequency w=1;
  // Perpendicular distances between the forces and the fulcrum
  parameter SI.Length L2=1;
  parameter SI.Length L3=2;
  parameter SI.Length L4=3;
  Components.Sf v1
    annotation (Placement(transformation(extent={{-120,-40},{-80,0}})));
  Junction0.J0_3 j0_3_1
    annotation (Placement(transformation(extent={{-72,-30},{-52,-10}})));
  Junction1.J1_3 j1_3_1
    annotation (Placement(transformation(extent={{-14,-30},{6,-10}})));
  Components.TF tF(m=L2/L3)
    annotation (Placement(transformation(extent={{6,-6},{34,22}})));
  Components.TF tF1(m=L2/L4)
    annotation (Placement(transformation(extent={{14,-34},{42,-6}})));
  Components.C inv_k2(C=1/10)
    annotation (Placement(transformation(extent={{126,-22},{66,38}})));
  Components.R b1(R=5)
    annotation (Placement(transformation(extent={{118,-40},{78,0}})));
  Components.C inv_k1(C=1/5) annotation (Placement(transformation(
        extent={{-30,-30},{30,30}}, rotation=-90, origin={-62,26})));
  Components.Bond bond
    annotation (Placement(transformation(extent={{-42,-30},{-22,-10}})));
  Junction0.J0_2 j0_2_1
    annotation (Placement(transformation(extent={{40,-2},{60,18}})));
  Junction0.J0_2 j0_2_2
    annotation (Placement(transformation(extent={{50,-30},{70,-10}})));
```

Modelica Code 3.18: The *SpringsDamperLever* model (1/2).

```
  equation
    // Velocity imposed to the left terminal of spring 1
    v1.s = if time > 2 and time < 20 then V10*sin(w*time) else 0;
    connect(v1.port, j0_3_1.port1) annotation (Line(points={{-80,-20},{-76,-20},
            {-72,-20}}, color={0,0,255}));
    connect(inv_k1.port, j0_3_1.port2) annotation (Line(points={{-62,-4},{-62,
            -7},{-62,-10}}, color={0,0,255}));
    connect(j0_3_1.port3, bond.port1) annotation (Line(points={{-52,-20},{-47,
            -20},{-42,-20}}, color={0,0,255}));
    connect(bond.port2, j1_3_1.port1) annotation (Line(points={{-22,-20},{-18,
            -20},{-14,-20}}, color={0,0,255}));
    connect(j1_3_1.port2, tF.port1) annotation (Line(points={{-4,-10},{2,-10},
            {2,8},{6,8}}, color={0,0,255}));
    connect(j1_3_1.port3, tF1.port1)
      annotation (Line(points={{6,-20},{10,-20},{14,-20}}, color={0,0,255}));
    connect(tF.port2, j0_2_1.port1)
      annotation (Line(points={{34,8},{37,8},{40,8}}, color={0,0,255}));
    connect(j0_2_1.port2, inv_k2.port)
      annotation (Line(points={{60,8},{66,8}}, color={0,0,255}));
    connect(tF1.port2, j0_2_2.port1) annotation (Line(points={{42,-20},{46,-20},
            {50,-20}}, color={0,0,255}));
    connect(j0_2_2.port2, b1.port) annotation (Line(points={{70,-20},{74,-20},
            {78,-20}}, color={0,0,255}));
    annotation (Icon(coordinateSystem(preserveAspectRatio=false)), Diagram(
          coordinateSystem(preserveAspectRatio=false, extent={{-160,-160},{
              160,160}})));
  end SpringsDamperLever;

end Examples;



end BondGraphLib;
```

Modelica Code 3.19: The *SpringsDamperLever* model (2/2).

Source of liquid

Purpose of this assignment

– Use if expressions, if clauses, and records.

4.1 System description

The characteristic curve of a source of liquid is shown in Figure 4.1, where F^{SP} and F represent the setpoint and actual mass flow rates, respectively. The mass flow rate is positive while the liquid flows into the source and negative while flows out from the source. The load pressure, denoted as p, is the pressure exerted by the environment on the source connection point. For instance, suppose that the liquid source is connected to a liquid storage tank. The load pressure is the liquid pressure in the tank at the connection point. The characteristic curve is defined by the following four parameters (see again Figure 4.1):

p_{eps}	Below this load pressure, the source is not able to extract liquid from its environment.
p_{min}	Below this load pressure, the mass flow rate extracted by the source decreases linearly.
p_{elbow}	Above this load pressure, the mass flow rate provided by the source decreases linearly.
p_{max}	Above this load pressure, the source is not able to inject flow into its environment.

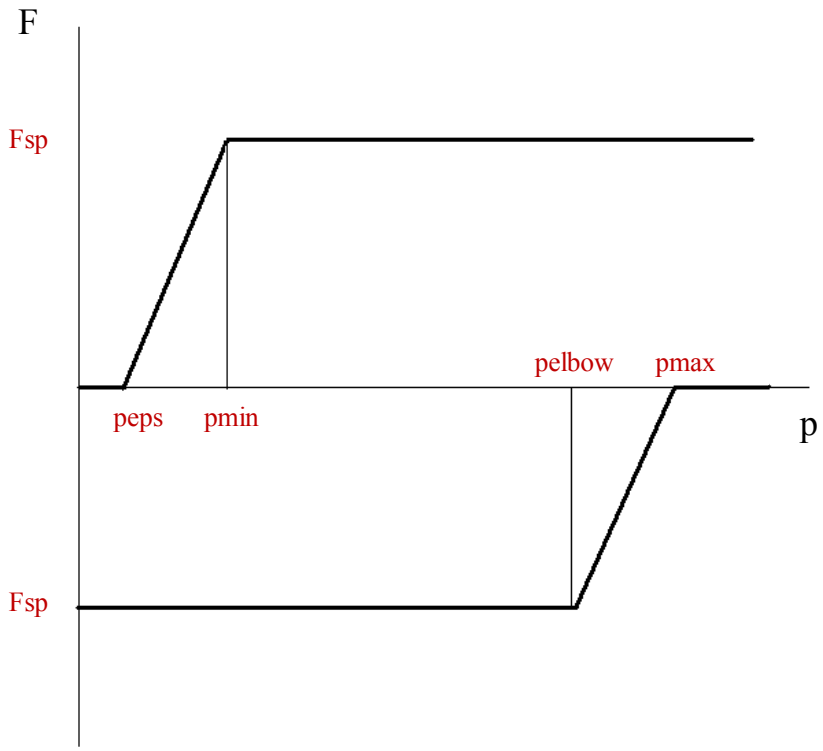


Figure 4.1: Characteristic curve of a source of liquid.

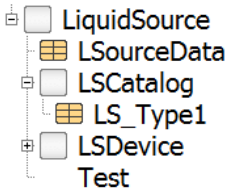


Figure 4.2: Architecture of the package to be developed in Task 2.

4.2 Tasks

1. Write the equations that represent the characteristic curve shown in Figure 4.1.

2. Program a Modelica package as shown in Figure 4.2, where `LSourceData` is a record that contains the declaration of the source parameters; `LSCatalog` is a package that contains the declaration of records of the `LSourceData` type; `LSDevice` is a package that contains the liquid source model; and `Test` is a model where the liquid source of the `LS_Type1` type is instantiated, and values are given to the load pressure and the setpoint in order to obtain the characteristic curve. The `LS_Type1` type of liquid source has the following parameter values: $p_{eps} = 10^5$ Pa, $p_{min} = 3 \cdot 10^5$ Pa, $p_{elbow} = 8 \cdot 10^5$ Pa and $p_{max} = 10^6$ Pa.

4.3 Solution to Task 1

The characteristic curve can be represented by means of the following equations. While the liquid flow exits the source (i.e., while $F^{SP} \leq 0$):

$$F = \begin{cases} F^{SP} & \text{if} \quad p < p_{elbow} \\ \frac{F^{SP}}{p_{max} - p_{elbow}} (p_{max} - p) & \text{if} \quad p_{elbow} \leq p < p_{max} \\ 0 & \text{if} \quad p \geq p_{max} \end{cases} \tag{4.1}$$

and while the liquid flow goes into the source (i.e., while $F^{SP} > 0$):

$$F = \begin{cases} F^{SP} & \text{if} \quad p > p_{min} \\ \frac{F^{SP}}{p_{min} - p_{eps}} (p - p_{eps}) & \text{if} \quad p_{eps} < p \leq p_{min} \\ 0 & \text{if} \quad p \leq p_{eps} \end{cases} \tag{4.2}$$

4.4 Solution to Task 2

Modelica Codes 4.1 – 4.3 describe the `LSourceData` record, and three equivalent models of the liquid source (`LiquidS1`, `LiquidS2` and `LiquidS3`). The `Test` model allows to experiment with them. The evolution of the load pressure and mass flow rate of the first of these models is shown in Figure 4.3.

```
package LiquidSource

record LSourceData
   import SI = Modelica.SIunits;
   parameter SI.Pressure peps;
   parameter SI.Pressure pmin;
   parameter SI.Pressure pelbow;
   parameter SI.Pressure pmax;
end LSourceData;

package LSCatalog
   record LS_Type1 = LSourceData(
      peps   = 1E5,
      pmin   = 3E5,
      pelbow = 8E5,
      pmax   = 10E5 );
end LSCatalog;
```

Modelica Code 4.1: Source of liquid (1/3).

```
package LSDevice

partial model LiquidS
   import SI = Modelica.SIunits;
   LSourceData LSdata;
   SI.MassFlowRate Fsp "Setpoint of the input flow to the source";
   SI.MassFlowRate F   "Input flow to the source";
   SI.Pressure p       "Load pressure";
end LiquidS;

model LiquidS1
   extends LiquidS;
equation
   if Fsp < 0 then
      F = if p < LSdata.pelbow then
             Fsp
          elseif p >= LSdata.pelbow and p < LSdata.pmax then
             Fsp*(LSdata.pmax-p)/(LSdata.pmax-LSdata.pelbow)
          else
             0;
   else
      F = if p > LSdata.pmin then
             Fsp
          elseif p > LSdata.peps and p <= LSdata.pmin then
             Fsp*(p-LSdata.peps)/(LSdata.pmin-LSdata.peps)
          else
             0;
   end if;
end LiquidS1;

model LiquidS2
   extends LiquidS;
equation
   F = if (Fsp < 0 and p < LSdata.pelbow) or (Fsp >= 0 and p > LSdata.pmin) then
          Fsp
       elseif Fsp < 0 and p >= LSdata.pelbow and p < LSdata.pmax then
          Fsp*(LSdata.pmax-p)/(LSdata.pmax-LSdata.pelbow)
       elseif Fsp >= 0 and p > LSdata.peps and p <= LSdata.pmin then
          Fsp*(p-LSdata.peps)/(LSdata.pmin-LSdata.peps)
       else
          0;
end LiquidS2;

model LiquidS3
   extends LiquidS;
equation
   F = if Fsp < 0 then
          if p < LSdata.pelbow then
             Fsp
          elseif p >= LSdata.pelbow and p < LSdata.pmax then
             Fsp*(LSdata.pmax-p)/(LSdata.pmax-LSdata.pelbow)
          else
             0
       else
          if p > LSdata.pmin then
             Fsp
          elseif p > LSdata.peps and p <= LSdata.pmin then
             Fsp*(p-LSdata.peps)/(LSdata.pmin-LSdata.peps)
          else
             0;
end LiquidS3;

end LSDevice;
```

Modelica Code 4.2: Source of liquid (2/3).

```
model Test
    import SI = Modelica.SIunits;
    LSDevice.LiquidS1 LS1( LSdata=LSCatalog.LS_Type1() );
    LSDevice.LiquidS2 LS2( LSdata=LSCatalog.LS_Type1() );
    LSDevice.LiquidS3 LS3( LSdata=LSCatalog.LS_Type1() );
    Boolean FlowGoingOutFromSource;
    SI.Pressure p(start=0, fixed=true) "Load pressure";
    parameter SI.MassFlowRate Fsp = 10 "Setpoint of input mass flow";
equation
    FlowGoingOutFromSource = sin(time) > 0;
    der(p) = 4E5;
    when { FlowGoingOutFromSource, not FlowGoingOutFromSource } then
       reinit(p,0);
    end when;
    LS1.p = p;
    LS1.Fsp = if FlowGoingOutFromSource then -Fsp else Fsp;
    LS2.p = p;
    LS2.Fsp = if FlowGoingOutFromSource then -Fsp else Fsp;
    LS3.p = p;
    LS3.Fsp = if FlowGoingOutFromSource then -Fsp else Fsp;
end Test;

end LiquidSource;
```

Modelica Code 4.3: Source of liquid (3/3).

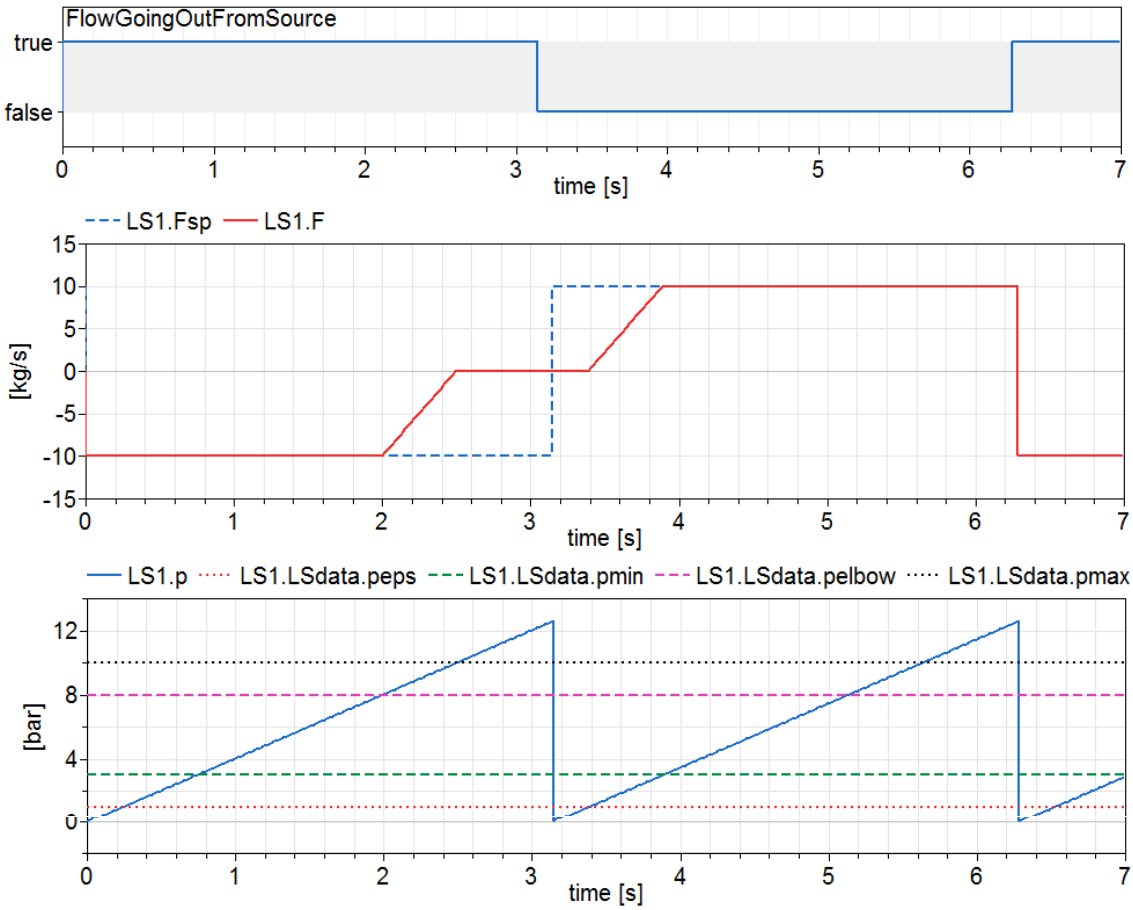


Figure 4.3: Simulation of the *Test* model.

Assignment 5

Ideal gas in a heated container

Purpose of this assignment

- Practice modeling of variable structure systems.
- Practice the use of the *unit*, *start* and *fixed* attributes.

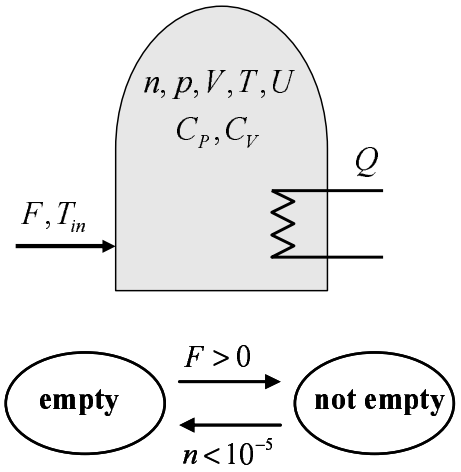
5.1 System description

A model of an ideal gas in a heated container is shown in Figure 5.1. It has two modes: *empty* and *not empty*. The volume of the container, V, is constant. The quantities n, p, T, U represent the number of moles, pressure, temperature and internal energy of the gas stored inside the container. The values of the specific heat capacities (C_P and C_V) are set assuming that the ideal gas is monatomic.

The gas is injected in the container at a molar flow rate F. A negative value of F indicates that the gas is being extracted from the container. The input gas has a temperature T_{in}. The heat power Q, and F and T_{in} are known functions of time. The meaning of the model quantities is described in Table 5.1.

5.2 Task

Describe the model in Modelica as an atomic model. Use the *unit* attribute to specify the units of parameters and variables. Initially, there are 20 moles of gas inside the container, at 300 K. Experiment with the model by setting the evolution in time of Q, F and T_{in}.



$$p \cdot V = n \cdot R \cdot T$$

$$\frac{dn}{dt} = \begin{cases} 0 & \text{empty} \\ F & \text{not empty} \end{cases}$$

$$\frac{dU}{dt} = \begin{cases} 0 & \text{empty} \\ F \cdot C_P \cdot T_{in} + Q & \text{not empty and } F > 0 \\ F \cdot C_P \cdot T + Q & \text{not empty and } F \leq 0 \end{cases}$$

$$U = n \cdot C_V \cdot T$$

$$C_P - C_V = R$$

Figure 5.1: Model of an ideal gas in a heated container.

Table 5.1: Quantities of the model shown in Figure 5.1.

Quantity	**Meaning**	**Units**
n	Number of moles	mol
p	Gas pressure	Pa
V	Container volume	1 m^3
T	Gas temperature	K
U	Gas internal energy	J
C_P	Specific heat capacity at constant pressure	J/(mol·K)
C_V	Specific heat capacity at constant volume	J/(mol·K)
F	Molar flow rate of gas	mol/s
T_{in}	Input gas temperature	K
Q	Heat power	W
R	Ideal gas constant	8.3145 J/(mol·K)

5.3 Solution

The model is described in Modelica Code 5.1. The `IdealGas` model describes the ideal gas. This model is inherited by the `Test` model, where the evolution in time of the boundary conditions is set. The result of the simulation is shown in Figure 5.2.

```
model IdealGas

   constant  Real R( unit="J/(mol.K)") = 8.3145  "Ideal gas constant";

   parameter Real V( unit="m3") = 1              "Container volume";

   // Specific heat capacities
   parameter Real Cp( unit="J/(mol.K)") = 5*R/2  "Monatomic ideal gas";
   parameter Real Cv( unit="J/(mol.K)") = Cp - R "Mayer's Law";

   Real n   (unit="mol", start=20, fixed=true) "Number of moles";
   Real p   (unit="N.m-2")                     "Gas pressure";
   Real T   (unit="K", start=300, fixed=true)  "Gas temperature";
   Real F   (unit="mol.s-1")                   "Input molar flow rate";
   Real Tin (unit="K")                         "Input temperature";
   Real Q   (unit="J.s-1")                     "Heat power";
   Real U   (unit="J")                         "Internal energy";

   // Modes and transition condition
   Boolean empty;
   parameter Real molEps = 1E-5;

equation

   // State equation of ideal gases
   p * V = n * R * T;

   // Balance of moles
   der(n) = if empty then 0 else F;

   // Balance of energy
   der(U) = if empty then 0 else if F>0 then F*Cp*Tin+Q else F*Cp*T+Q;

   // Internal energy
   U = n * Cv * T;

   // Mode transition
   when F > 0 and pre(empty) or n < molEps and not pre(empty) then
      empty = not pre(empty);
   end when;

initial equation
   empty = ( n < molEps );

end IdealGas;

model Test
   extends IdealGas;
equation
   Tin = if time < 10 then 275 else 350;
   F = if time < 10 then 2
       else if time < 30 then -4
       else if time < 50 then 3 else 0;
   Q = if time > 60 and time < 80 then 1E3 else 0;
end Test;
```

Modelica Code 5.1: Monatomic ideal gas in a heated container.

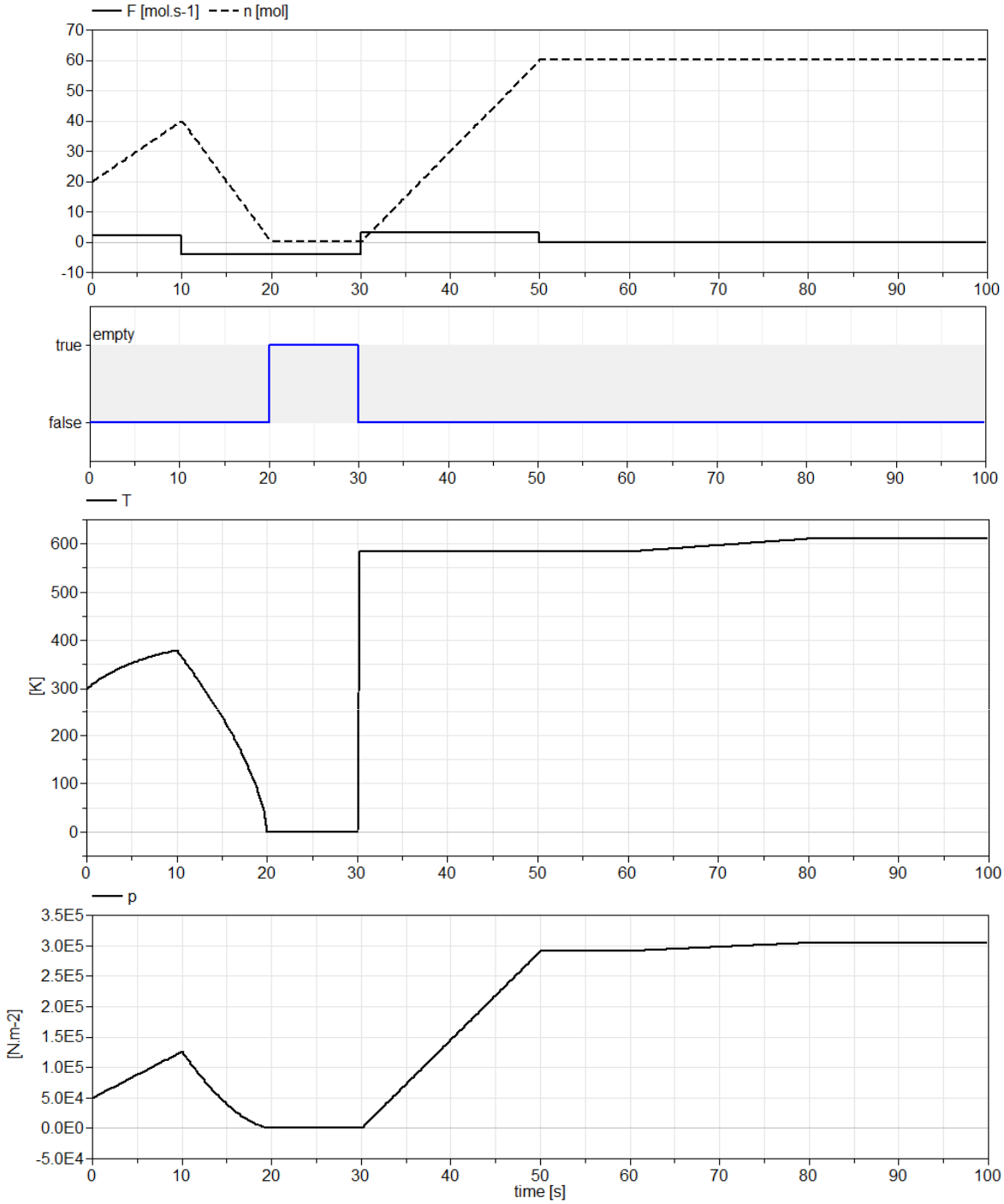


Figure 5.2: Result obtained simulating Modelica Code 5.1.

Assignment 6

Hysteresis controller

Purpose of this assignment

- Practice modeling of finite state machines.
- Practice the declaration and use of block classes, and matrix equations.

6.1 System description

A SISO plant (*Single-Input and Single-Output*) controlled by a hysteresis controller is represented in Figure 6.1. The following variables appear in the figure: r, e, u and y. The r variable is the reference. The error variable (e) is calculated subtracting the plant output (y) from the reference (r). The error variable is the controller input. The controller output (u) is the plant input.

The characteristic curve in the first quadrant of the hysteresis controller is plotted in Figure 6.2. It determines the value of the controller output (u), calculated from the controller input (e) and the controller mode. The controller operates in four different modes, named $\{s_1, s_2, s_3, s_4\}$, which correspond to four different parts of the characteristic curve (see again the figure). The characteristic curve depends on the position of the points $\{a, b, c, d, e, f\}$, and is symmetric in the first and third quadrants, i.e., the following relationship is satisfied: $u(e) = -u(-e)$.

The plant model is described in Figure 6.1, where $\mathbf{A}$ is a $N \times N$ matrix of constant components; $\mathbf{B}$ and $\mathbf{C}$ are vectors of N constant components; and the state variable vector $\mathbf{x}$ has N components. The plant input (u) and output (y) are scalar variables.

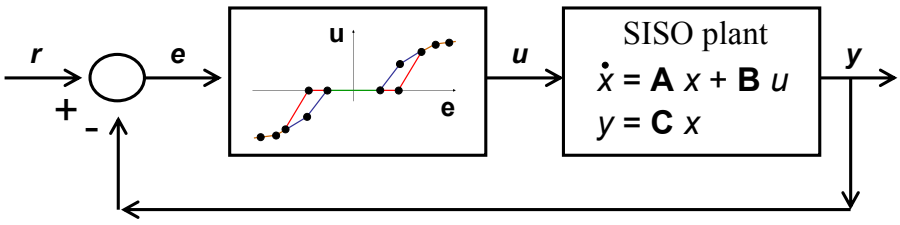


Figure 6.1: Control loop.

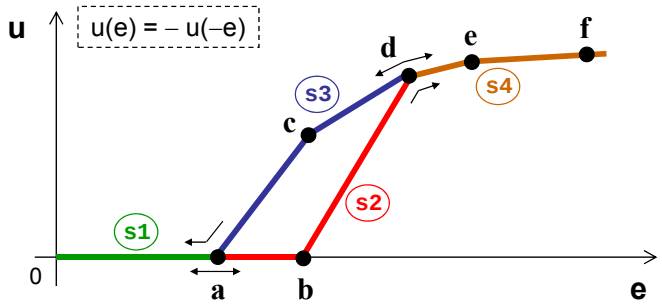


Figure 6.2: Characteristic curve in the first quadrant of the hysteresis controller.

6.2 Task

Describe the plant model, which is composed of Eqs. (6.1) and (6.2), as a Modelica block class. Declare **A** as a parameter, specifying its dimension, but not its size. Describe the plant model so that the size of **A**, **B** and **C** is determined by the modeling environment from the value assigned to **A** when the model is inherited or instantiated.

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot u \tag{6.1}$$

$$y = \mathbf{C} \cdot \mathbf{x} \tag{6.2}$$

Describe the hysteresis controller as a block class. Represent its behavior using a finite-state machine with four modes: $\{s_1, s_2, s_3, s_4\}$. Finally, compose the complete system instantiating the controller and plant models. Simulate the model for different values of the **A** matrix, and the **B** and **C** vectors. The reference variable is calculated as shown below.

$$r = 15 \cdot \sin\left(\frac{t}{10}\right) + \sin(t) \tag{6.3}$$

6.3 Solution

The finite-state machine (FSM) is plotted in Figure 6.3.

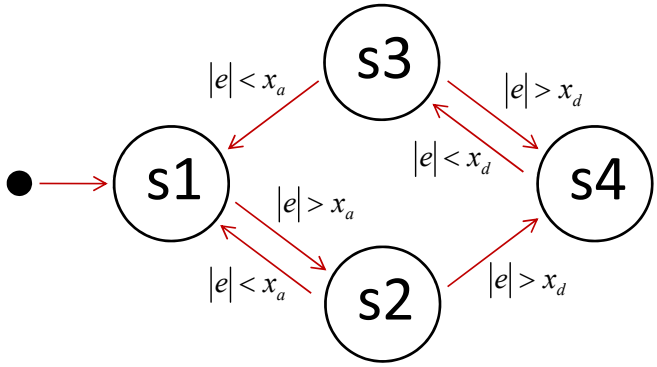


Figure 6.3: Behavior of the hysteresis controller described as a FSM.

The models of the controller, the plant and the control loop are shown in Modelica Codes 6.1 – 6.3. The package structure is shown in Figure 6.4, and the simulation results in Figure 6.5.

```
package HysteresisControlLoop

package Components

block Controller
   input      Real  e;
   output     Real  u;
   // Coordinates of the {a,b,c,d,e,f} points
   parameter Real     xA,
                      xB,
                      xC, yC,
                      xD, yD,
                      xE, yE,
                      xF, yF;
   constant Real      eps = 1e-10;
protected
   Real       traject1  "0 - a",
              traject2  "a - b - d",
              traject3  "a - c - d",
              traject4  "d - e - f";
   Boolean    s1  (start=true)  "traject1",
              s2  (start=false)  "traject2",
              s3  (start=false)  "traject3",
              s4  (start=false)  "traject4";
   Real       absE,
              absU;
```

Modelica Code 6.1: SISO plant and hysteresis controller (1/3).

```
equation

  // ---------------------------------------
  // Parts of the characteristic curve
  // ---------------------------------------
  absE = abs(e);
  traject1 = 0;
  traject2 = if absE < xB
          then 0
          else ( yD * absE - xB * yD ) / ( abs( xD - xB ) + eps );
  traject3 = if absE < xC
          then ( yC * absE - xA * yC ) / ( abs( xC - xA ) + eps )
          else ( ( yD - yC ) * absE + yC * xD - xC * yD ) / ( abs( xD - xC ) + eps );
  traject4 = if absE < xE
          then ( ( yE - yD ) * absE + yD * xE - xD * yE ) / ( abs( xE - xD ) + eps )
          else ( ( yF - yE ) * absE + yE * xF - xE * yF ) / ( abs( xF - xE ) + eps );

  // -------------------------
  // Characteristic curve
  // -------------------------
  absU = if s1
         then traject1
         else if s2
              then traject2
              else if s3
                   then traject3
                   else traject4;
  u = if e > 0 then absU else -absU;

  // -------------------
  // Mode transition
  // -------------------
  s1 = pre(s3) and absE < xA or
       pre(s2) and absE < xA or
       pre(s1) and not absE > xA;

  s2 = pre(s1) and absE > xA or
       pre(s2) and not ( absE < xA or absE > xD );

  s3 = pre(s4) and absE < xD or
       pre(s3) and not ( absE < xA or absE > xD );

  s4 = pre(s2) and absE > xD or
       pre(s3) and absE > xD or
       pre(s4) and not absE < xD;

end Controller;
```

Modelica Code 6.2: SISO plant and hysteresis controller (2/3).

```
block SISOplant
   input  Real    u;
   output Real    y;

   parameter Real A[:,:],
                 B[size(A,1)],
                 C[size(A,2)];
protected
   Real   x[size(A,2)];
equation
   assert( size(A,1) == size(A,2), "A should be a square matrix");
   der(x) = A * x + B * u;
   y      = C * x;
end SISOplant;

end Components;



model ControlLoop
   Components.Controller NL(xA = 1,
                            xB = 2,
                            xC = 1.5, yC = 1,
                            xD = 2.5, yD = 2.1,
                            xE = 3,   yE = 2.5,
                            xF = 4,   yF = 5);

   Components.SISOplant Plant ( A = [ -2,  15; 3, 1 ],
                                B = { 4, 2 },
                                C = { 2, 5 } );
   Real  reference;
equation
   // -----------------
   // Reference variable
   // -----------------
   reference = 15*sin(time/10) + sin(time);
   // --------------
   // Control loop
   // --------------
   NL.u = Plant.u;
   NL.e = reference - Plant.y;
end ControlLoop;

end HysteresisControlLoop;
```

Modelica Code 6.3: SISO plant and hysteresis controller (3/3).

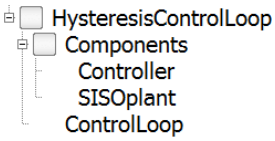


Figure 6.4: Architecture of the *HysteresisControlLoop* package.

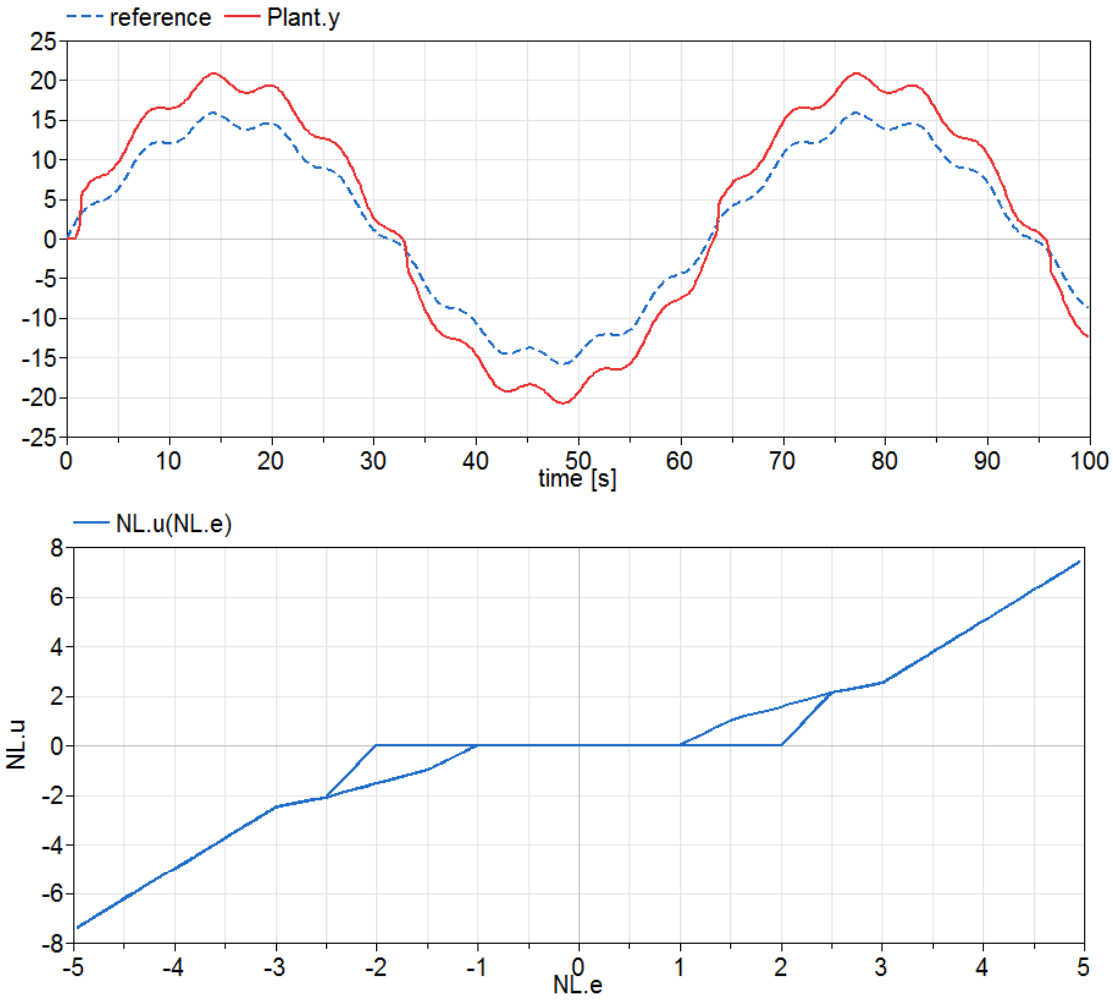


Figure 6.5: Simulation of the *ControlLoop* model.

Draining of a benzene storage tank

Purpose of this assignment

– Practice developing a Modelica library, and composing a model by instantiating and connecting library components.

7.1 System description

A tank used to store benzene is drained through a horizontal circular pipe connected to the tank bottom. The other end of the pipe is connected to a liquid sink that is at atmospheric pressure. The inner diameter (D), and length (L) of the pipe are $D = 5.08$ cm and $L = 2$ m. The cross-sectional inner area of the pipe is $S = \pi \cdot D^2/4$.

The volume of the storage tank is 1 m^3, and its bottom area 1 m^2. The benzene is stored at 27 °C. The density and dynamic viscosity of benzene at this temperature are approximately $\rho = 875$ kg/m^3 and $\mu = 56.5 \cdot 10^{-5}$ N·s/m^2.

The balance of linear momentum (P) is applied to the benzene that flows through the pipe, as shown in Eq. (7.1). Two forces are exerted on the benzene that flows through the pipe: (1) the force f_p, produced by the pressure difference between the pipe ends; and (2) the friction f_F between the pipe wall and the benzene.

$$\frac{dP}{dt} = f_F + f_p \tag{7.1}$$

Assuming that the pipe is completely filled with benzene, these forces are calculated as shown below. The area of the wetted surface is S_w.

$$f_p = S \cdot \Delta p \tag{7.2}$$

$$f_F = \begin{cases} -S_w \cdot \frac{1}{2} \cdot \rho \cdot v^2 \cdot \kappa_{Fanning} & \text{if} \quad P \geq 0 \\ S_w \cdot \frac{1}{2} \cdot \rho \cdot v^2 \cdot \kappa_{Fanning} & \text{if} \quad P < 0 \end{cases} \tag{7.3}$$

The linear momentum (P) of the benzene that flows through the pipe is related to the mass flow rate (F) and the pipe length (L) as follows:

$$P = F \cdot L \tag{7.4}$$

The relationship among the liquid velocity (v), density (ρ) and mass flow rate (F), and the cross-sectional inner area (S) of the pipe is:

$$F = S \cdot \rho \cdot v \tag{7.5}$$

The Fanning friction factor, $\kappa_{Fanning}$, is estimated in this model applying the Blasius correlation:

$$\kappa_{Fanning} = \begin{cases} \frac{16}{Re} & \text{if} \quad Re < 2100 \quad \text{(laminar flow)} \\ \frac{0.0791}{Re^{1/4}} & \text{if} \quad 2100 < Re < 10^5 \quad \text{(turbulent flow)} \end{cases} \tag{7.6}$$

where the Reynolds number is calculated as follows:

$$Re = \frac{D \cdot v \cdot \rho}{\mu} \tag{7.7}$$

7.2 Task

A source of benzene is connected to the upper part of the storage tank. The storage tank is initially empty. Benzene is introduced in the tank, at a constant volumetric flow rate of 0.01 m^3/s, during ten minutes. After this time, the input flow is zero. The benzene is drained through the pipe. The objective is to simulate the mass of the benzene stored in the tank, and the linear momentum, Reynolds number and regime (laminar or turbulent flow) of the benzene that flows through the pipe.

7.3 Solution

The architecture of the developed library is shown in Figure 7.1. The Modelica description is shown in Modelica Code 7.1 – 7.4. The evolution of the stored mass of liquid, the linear momentum of the liquid inside the pipe, the Reynolds number and the flow regime is plotted in Figure 7.2.

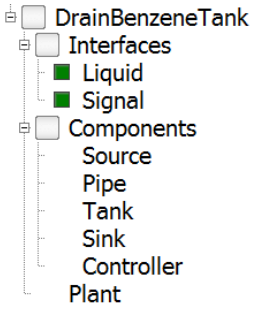


Figure 7.1: Architecture of the library.

```
package DrainBenzeneTank

import SI = Modelica.SIunits;


package Interfaces

connector Liquid
   SI.Pressure p "Pressure";
   flow SI.MassFlowRate Fm "Mass flow rate";
end Liquid;

connector Signal
   Real s "Setpoint signal";
end Signal;

end Interfaces;



package Components

model Source
   Interfaces.Liquid liq;
   Interfaces.Signal  sig;
equation
   liq.Fm = -sig.s;
end Source;
```

Modelica Code 7.1: Draining of a benzene storage tank through a pipe (1/4).

```
model Pipe
   constant Real PI = 2*Modelica.Math.asin(1.0);

   Interfaces.Liquid liqI, liq0;

   parameter SI.Length L "Length";
   parameter SI.Diameter D "Inner diameter";
   parameter SI.Area S = PI*D^2/4 "Cross-sectional inner area";
   parameter SI.Area Sw = PI*D*L "Wetted surface area";

   parameter SI.Density rho "Liquid density";
   parameter SI.DynamicViscosity mu "Dynamic viscosity of liquid";

   parameter Real ReC(unit="") = 2100 "Critical Reynolds number";
   SI.Force fP "Force produced by pressure difference";
   SI.Force fF "Friction force";
   SI.Momentum P(start=0, fixed=true) "Linear momentum of liquid";
   SI.Velocity v "Velocity of liquid";

   Real Re(unit="") "Reynolds number";
   Real Kfanning "Fanning friction factor";
   Boolean laminar "Flow regime: true - laminar, false - turbulent";

equation
   der(P) = fF + fP;
   fP = S * ( liqI.p - liq0.p );
   fF = if noEvent(P<0) then  0.5*Sw*rho*v^2*Kfanning
                        else -0.5*Sw*rho*v^2*Kfanning;
   P = liqI.Fm * L;
   liqI.Fm =-liq0.Fm;
   liqI.Fm = S*rho*v;
   laminar = Re < ReC;
   Kfanning = if noEvent(Re<ReC) then  16/Re else
                                       0.0791/Re^0.25;
   Re = if noEvent( v>0 )
            then D*v*rho/mu
            else if noEvent( v<0 )
                     then -D*v*rho/mu
                     else 1;
   when fP <= 0 then
      reinit(P,0);
   end when;
end Pipe;
```

Modelica Code 7.2: Draining of a benzene storage tank through a pipe (2/4).

```
model Tank
   Interfaces.Liquid liqBase, liqSuperior;

   constant SI.Acceleration g= 9.81 "Gravitational acceleration";

   SI.Mass m(start=0, fixed=true) "Mass of liquid";
   SI.Volume V "Volume of liquid";

   parameter SI.Density rho "Liquid density";
   parameter SI.Area S "Bottom area of tank";
   parameter SI.Volume Vmax "Maximum volume of liquid";

equation
   der(m) = liqBase.Fm + liqSuperior.Fm;
   V      = m/rho;
   liqSuperior.p   = 0;
   liqBase.p = if m>0 then m*g/S else 0;
   when m<0 then
      reinit(m,0);
   end when;
   assert(V<=Vmax, "Maximum volume exceeded");
end Tank;

model Sink
   Interfaces.Liquid liq;
equation
   liq.p = 0;
end Sink;

model Controller
   Interfaces.Signal signal;
   parameter SI.Time t1;
   parameter SI.MassFlowRate FmSP1, FmSP2;
equation
   signal.s = if time<t1 then FmSP1 else FmSP2;
end Controller;

end Components;
```

Modelica Code 7.3: Draining of a benzene storage tank through a pipe (3/4).

```
model Plant
  // Benzene
  parameter SI.Density rho = 875 "Benzene density";
  parameter SI.DynamicViscosity mu = 56.5e-5 "Dynamic viscosity of benzene";
  // Pipe
  parameter SI.Length L = 2 "Pipe length";
  parameter SI.Diameter D = 5.08E-2 "Pipe diameter";
  // Tank
  parameter SI.Area S = 1 "Bottom area";
  parameter SI.Volume Vmax = 1 "Maximum volume";

  Components.Tank tank(S=S, Vmax=Vmax, rho=rho);
  Components.Source source;
  Components.Pipe pipe(L=L, D=D, rho=rho, mu=mu);
  Components.Sink sink;
  Components.Controller control(t1=600, FmSP1=0.01*rho, FmSP2=0);
equation
  connect(control.signal,source.sig);
  connect(tank.liqSuperior,source.liq);
  connect(tank.liqBase,pipe.liqI);
  connect(sink.liq,pipe.liqO);
end Plant;

end DrainBenzeneTank;
```

Modelica Code 7.4: Draining of a benzene storage tank through a pipe (4/4).

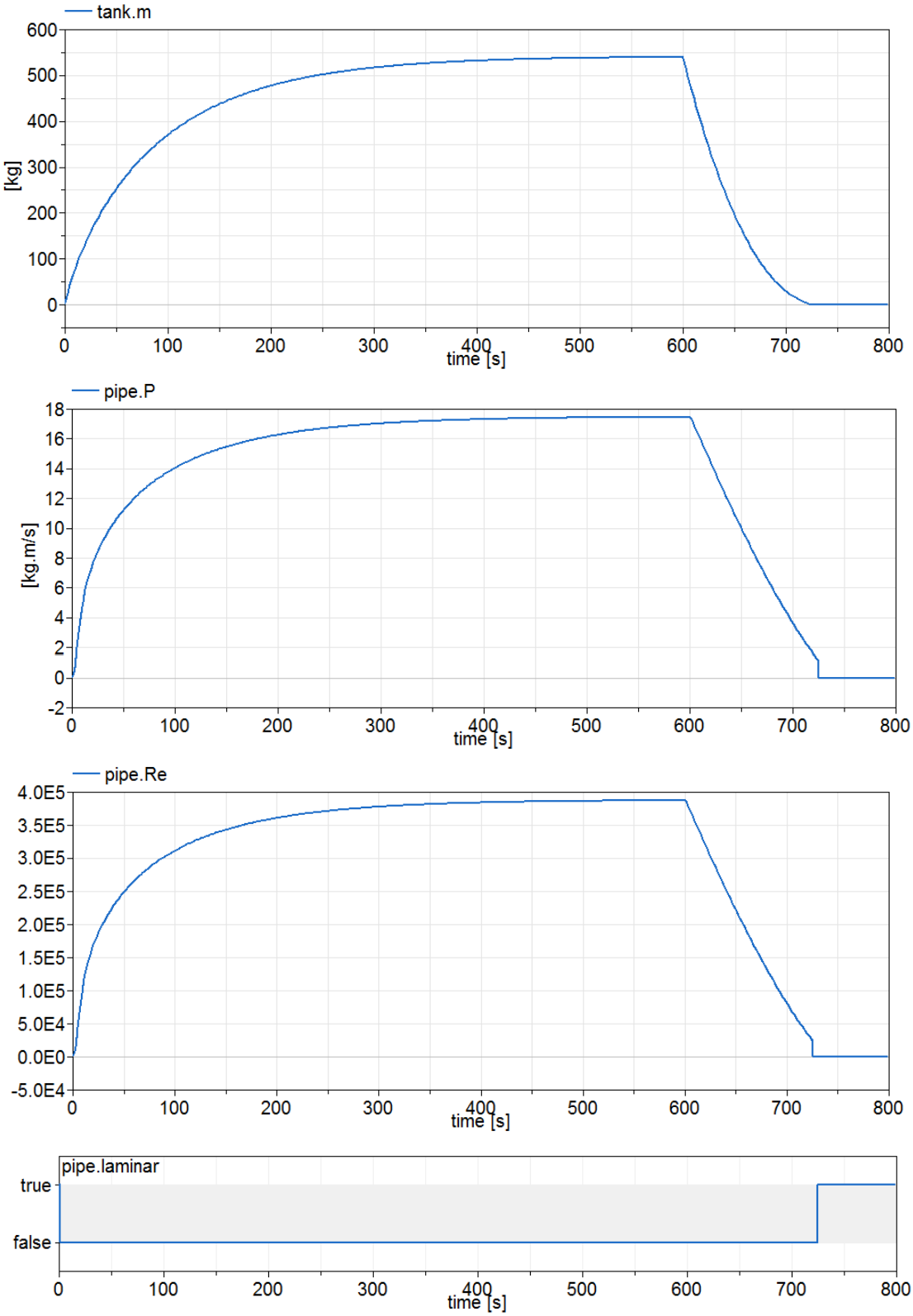


Figure 7.2: Result obtained simulating the *Plant* model.

Heating a liquid mixture

Purpose of this assignment

- Practice modeling of simple thermo-hydraulic systems, posing mass and energy balances, and describing changes in the liquid flow direction.
- Describe multi-mode models in Modelica.

8.1 System description

The model diagram of a system composed of two liquid storage tanks, a liquid source and a pump is shown in Figure 8.1. The tanks, named Tank 1 and Tank 2, have a capacity of 10 m^3 and 1 m^3, respectively. Tank 1 is connected to the liquid source. The pump transfers liquid between the tanks. Tank 2 is heated. The setpoint signals of the source, the pump and the heater are provided by three controllers.

As the tanks are closed vessels, the volume of the liquid stored within a tank cannot be greater than the tank volume. The flows of the source and pump are determined by their setpoint signals, with the following two exceptions: (1) it is not possible to extract liquid from an empty tank; and (2) it is not possible to introduce liquid into a filled tank. It is assumed that a tank is empty while the mass of stored liquid is not greater than 1 kg.

Both tanks are initially empty. The system is operated following the sequence of steps described below.

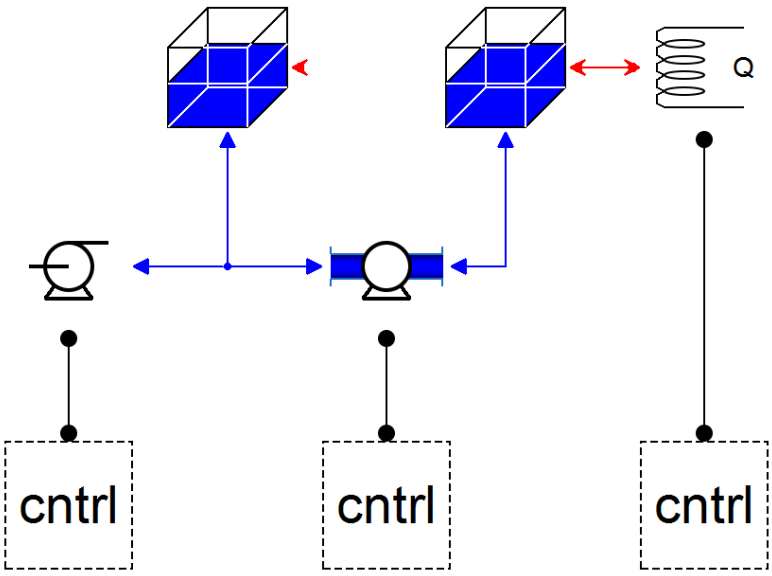


Figure 8.1: Diagram of the two-tank system.

1. During 3 minutes (i.e., from $t = 0$ to $t = 180$ s), the liquid source introduces into Tank 1 a liquid mixture composed of a 25 % concentration by mass of kerosene, and a 75 % concentration by mass of benzene. The liquid temperature is 300 K. The mass flow rate is 40 kg/s.
2. At $t = 240$ s, the pump starts transferring liquid from Tank 1 to Tank 2. The setpoint value of the mass flow rate is 10 kg/s during 2 minutes. Observe that when the maximum volume of liquid is reached, no more liquid can be introduced in Tank 2.
3. At $t = 360$ s, the heater is switched on. The liquid stored in Tank 2 is heated during 6 minutes. The heat flow rate is $2.5 \cdot 10^5$ J/s. The heater is switched off at $t = 720$ s.
4. At $t = 720$ s, the pump starts transferring liquid from Tank 2 to Tank 1 at a mass flow rate of 10 kg/s. The controller sets the setpoint signal at this value during 3 minutes (from $t = 720$ s to $t = 900$ s). Nevertheless, once Tank 2 becomes empty, no more liquid can be extracted from it.
5. At $t = 840$ s, the liquid source starts to drain Tank 1 at a mass flow rate of 40 kg/s. The simulation finishes at $t = 1200$ s.

The density of kerosene and benzene is 760 kg/m^3 and 849 kg/m^3, respectively. The heat capacities of kerosene and benzene, within the temperature range of interest, can be approximated to the following linear functions of temperature: $C_p = 446 + 5.36 \cdot T$ and $C_p = 325 + 4.60 \cdot T$, expressed in J/(kg·K).

8.2 Task

Program a Modelica library named *TwoTanks*, composed of the following classes.

- A package named *Interfaces* that contains the declaration of the connectors describing the flow of liquid, heat and information (setpoint signals). These connectors are represented in Figure 8.1 as a blue triangle, a red triangle and a black dot, respectively.
- A package named *Components* that contains the declaration of the plant unit classes: the liquid source, the tank, the pump and the heater.
- A package named *Controllers* that contains the declaration of the three controllers.
- A model named *Plant*, describing the complete system. The diagram of this model is shown in Figure 8.1.

8.3 Solution

The library is shown in Modelica Codes 8.1 – 8.8. The annotations comply with the syntax supported by Dymola 6.1. The structure of the library is depicted in Figure 8.2. The simulation results are represented in Figures 8.3 – 8.5.

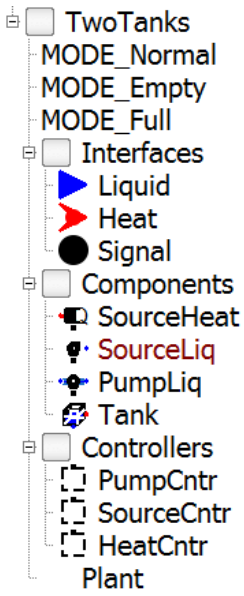


Figure 8.2: Structure of the TwoTanks library.

```
package TwoTanks

import SI = Modelica.SIunits;

constant Integer MODE_Normal = 0;
constant Integer MODE_Empty  = 1;
constant Integer MODE_Full   = 2;

package Interfaces

connector Liquid
   parameter Integer nComp=1;

   Integer mode;

   SI.Mass mass[nComp] "Mass of liquid components";
   SI.Temperature temp "Temperature";

   flow SI.MassFlowRate   Fm[nComp]  "Mass flow rate";
   flow SI.EnergyFlowRate Fh         "Enthapy flow";

  annotation (
    Coordsys(extent=[-1, -1; 1, 1]),
    Documentation(info="Liquid flow"),
    Icon     (Polygon(points=[-1, -1; 1, 0; -1, 1],
                      style(color=3, fillColor=3))),
    Diagram  (Polygon(points=[-1, -1; 1, 0; -1, 1],
                      style(color=3, fillColor=3))),
    Terminal(Polygon(points=[-1, -1; 1, 0; -1, 1],
                      style(color=3, fillColor=3))));
end Liquid;

connector Heat

   Integer mode;

   flow SI.EnergyFlowRate Q;

  annotation (
    Coordsys(extent=[-1, -1; 1, 1]),
    Documentation(info="Heat flow"),
    Icon     (Polygon(points=[-1, -1; 1, 0; -1, 1; 0,0],
                      style(color=1, fillColor=1)),
              Ellipse(extent=[-0.5,-0.5; 0.5,0.5],
                      style(color=1, fillColor=1))),
    Diagram  (Polygon(points=[-1, -1; 1, 0; -1, 1; 0,0],
                      style(color=1, fillColor=1)),
              Ellipse(extent=[-0.5,-0.5; 0.5,0.5],
                      style(color=1, fillColor=1))),
    Terminal(Polygon(points=[-1, -1; 1, 0; -1, 1; 0,0],
                      style(color=1, fillColor=1)),
              Ellipse(extent=[-0.5,-0.5; 0.5,0.5],
                      style(color=1, fillColor=1))));
end Heat;
```

Modelica Code 8.1: Two-tank system (1/8).

```
connector Signal
   parameter Integer nSignal = 1;
   Real s[nSignal];
  annotation (
    Coordsys(extent=[-1, -1; 1, 1]),
    Documentation(info="Information"),
    Icon    (Ellipse(extent=[-1, -1; 1,1],
                     style(color=0, fillColor=0))),
    Diagram (Ellipse(extent=[-1, -1; 1,1],
                     style(color=0, fillColor=0))),
    Terminal(Ellipse(extent=[-1, -1; 1,1],
             style(color=0, fillColor=0))));
end Signal;

end Interfaces;



package Components

model SourceHeat
   Interfaces.Heat heat
         annotation (extent=[-100,-10; -80,10]);
   Interfaces.Signal signal(nSignal=1)
         annotation (extent=[-10,-100; 10,-80]);

equation
   {heat.Q} = if noEvent(heat.mode == MODE_Empty)
              then {0} else signal.s;

  annotation (
    Coordsys(extent=[-100, -100; 100, 100]),
    Documentation(info="Heat flow source"),
    Icon    (
       Ellipse(extent=[-50,25; 0,35],   style(color=0)),
       Ellipse(extent=[-50,5; 0,15],    style(color=0)),
       Ellipse(extent=[-50,-15; 0,-5],  style(color=0)),
       Ellipse(extent=[-50,-35; 0,-25], style(color=0)),
       Line(points=[50,50; -50,50; -60,45; -60,35; -50,30;
                    -60,25; -60,15; -50,10; -60,5;
                    -60,-5; -50,-10; -60,-15; -60,-25;
                    -50,-30; -60,-35; -60,-45;
                    -50,-50; 50,-50], style(color=0)),
       Text(string="Q", extent=[30,-20; 70,20], style(color=0))));
end SourceHeat;
```

Modelica Code 8.2: Two-tank system (2/8).

```
model SourceLiq
   parameter Integer nComp = 1;

   Interfaces.Liquid liq(nComp=nComp)
         annotation (extent=[80,-10; 100,10], rotation=180);
   Interfaces.Signal signal(nSignal=nComp+2)
         annotation (extent=[-10,-100; 10,-80]);
   // Control signals
   Real sFracMass[nComp];
   SI.MassFlowRate sTotalMassF;
   SI.Temperature sTempF;
   // Specific heat capacity
   parameter Real CpCoefs[nComp,2];

protected
   SI.SpecificHeatCapacity Cp[nComp];
   SI.Temperature TempF;

equation
   signal.s[1:nComp] = sFracMass;
   signal.s[nComp+1] = sTotalMassF;
   signal.s[nComp+2] = sTempF;

   if sTotalMassF > 0 then
      // Liquid enters into the source
      liq.Fm = if noEvent(liq.mode == MODE_Empty)
               then zeros(nComp)
               else sTotalMassF*liq.mass/sum(liq.mass);
      TempF = liq.temp;
   else
      // Liquid exits from the source
      liq.Fm = if noEvent(liq.mode == MODE_Full)
               then zeros(nComp)
               else sTotalMassF*sFracMass;
      TempF  = sTempF;
   end if;
   for i in 1:nComp loop
      Cp[i] = CpCoefs[i,1] + CpCoefs[i,2]*TempF;
   end for;
   liq.Fh = Cp*TempF*liq.Fm;

  annotation (
    Coordsys(extent=[-100, -100; 100, 100]),
    Documentation(info="Liquid source"),
    Icon     (
      Line(points=[0,0; -30,-40; 30,-40; 0,0],
           style(fillColor=7, color=0, thickness=2)),
      Ellipse(extent=[-30,-30; 30,30],
              style(fillColor=7, color=0, thickness=2)),
      Line(points=[-50,0; 0,0], style(thickness=2, color=0)),
      Line(points=[0,30; 50,30], style(thickness=2, color=0))));

end SourceLiq;
```

Modelica Code 8.3: Two-tank system (3/8).

```
model PumpLiq
   parameter Integer nComp = 1;
   parameter SI.Mass epsMass = 1;
   Interfaces.Signal signal(nSignal=1)
         annotation (extent=[-10,-100; 10,-80]);
   Interfaces.Liquid liqIn(nComp=nComp)
         annotation (extent=[-100,-10; -80,10]);
   Interfaces.Liquid liqOut(nComp=nComp)
         annotation (extent=[80,-10; 100,10], rotation=180);
   // Specific heat capacity
   parameter Real CpCoefs[nComp,2];
   // Control signal
   SI.MassFlowRate sTotalMassF;
protected
   SI.SpecificHeatCapacity Cp[nComp];
   SI.Temperature TempF;
equation
   signal.s = {sTotalMassF};
   if sTotalMassF > 0 then
      // Liquid flows from liqIn to liqOut
      liqIn.Fm = if noEvent( liqIn.mode == MODE_Empty or
                             liqOut.mode == MODE_Full )
                 then zeros(nComp)
                 else sTotalMassF*liqIn.mass/sum(liqIn.mass);
      TempF = liqIn.temp;
   else
      // Liquid flows from liqOut to liqIn
      liqIn.Fm = if noEvent( liqOut.mode == MODE_Empty or
                              liqIn.mode == MODE_Full )
                 then zeros(nComp)
                 else sTotalMassF*liqOut.mass/sum(liqOut.mass);
      TempF = liqOut.temp;
   end if;
   liqOut.Fm = -liqIn.Fm;
   for i in 1:nComp loop
      Cp[i] = CpCoefs[i,1] + CpCoefs[i,2]*TempF;
   end for;
   liqIn.Fh  = Cp*TempF*liqIn.Fm;
   liqOut.Fh = -liqIn.Fh;

  annotation (
    Coordsys(extent=[-100, -100; 100, 100]),
    Documentation(info="Pump"),
    Icon    (
       Rectangle(extent=[-70,-15; 70,15],
                 style(color=7, fillColor=3, gradient=2)),
       Line(points=[-70,-25; -70,-15; 70,-15; 70,-25]),
       Line(points=[-70,25; -70,15; 70,15; 70,25]),
       Line(points=[0,0; -30,-40; 30,-40; 0,0],
            style(fillColor=7, color=0, thickness=2)),
       Ellipse(extent=[-30,-30; 30,30],
               style(fillColor=7, color=0, thickness=2)) ));
end PumpLiq;
```

Modelica Code 8.4: Two-tank system (4/8).

```
model Tank
   parameter Integer nComp = 1;
   Interfaces.Liquid liq(nComp=nComp)
         annotation (extent=[-10,-100; 10,-80], rotation=90);
   Interfaces.Heat   heat
         annotation (extent=[80,-10; 100,10], rotation=180);
   // Specific heat capacity
   parameter Real CpCoefs[nComp,2];
   // Density
   parameter SI.Density density[nComp];

   SI.Mass mass[nComp] (start=epsMass*ones(nComp)/nComp, fixed=true);
   SI.Temperature temp(start=300, fixed=true);
   SI.Enthalpy enthalpy;
   SI.SpecificHeatCapacity Cp[nComp];
   parameter SI.Volume volumeTank = 1 "Tank volume";
   SI.Volume volL "Volume of liquid";
   parameter SI.Mass epsMass = 1;
equation
   // Modes
   liq.mode = heat.mode;
   liq.mode = if not sum(mass) > epsMass then MODE_Empty
              else if not volL < volumeTank then MODE_Full
              else MODE_Normal;
   // Mass balance
   der(mass) = liq.Fm;
   liq.mass = mass;
   volL = sum( mass[i]/density[i] for i in 1:nComp);
   // Energy balance
   for i in 1:nComp loop
      Cp[i] = CpCoefs[i,1] + CpCoefs[i,2]*temp;
   end for;
   der(enthalpy) = liq.Fh + heat.Q;
   enthalpy = liq.mass*Cp*temp;
   liq.temp = temp;

  annotation (
    Coordsys(extent=[-100, -100; 100, 100]),
    Documentation(info="Control volume containing a liquid mixture"),
    Icon    (
       Rectangle(extent=[-75,-75; 25,25], style(color=0, fillColor=7)),
       Rectangle(extent=[-25,-25; 75,75], style(color=0, fillColor=7)),
       Polygon(points=[-75,25; -25,75; 75,75; 25,25],
               style(color=0, fillColor=7)),
       Polygon(points=[-75,-75; -25,-25; 75,-25; 25,-75],
               style(color=0, fillColor=7)),
       Polygon(points=[-75,-75; -25,-25; -25,75; -75,25],
               style(color=0, fillColor=7)),
       Polygon(points=[25,-75; 75,-25; 75,75; 25,25],
               style(color=0, fillColor=7)),
       Polygon(points=[25,-75; 75,-25; 75,25; 25,25],
               style(color=0, fillColor=7)),
```

Modelica Code 8.5: Two-tank system (5/8).

```
        Rectangle(extent=[-75,-75; 25,-20],
                style(color=0, fillColor=3)),
        Rectangle(extent=[-25,-25; 75,30],
                  style(color=0, fillColor=3)),
        Polygon(points=[-75,-20; -25,30; 75,30; 25,-20],
                style(color=0, fillColor=3)),
        Polygon(points=[25,-75; 75,-25; 75,30; 25,-20],
                style(color=0, fillColor=3)),
        Line(points=[-25,-25; 75,-25], style(color=7)),
        Line(points=[-75,-75; -25,-25], style(color=7)),
        Line(points=[-75,25; 25,25], style(color=0)),
        Line(points=[-25,-25; -25,30], style(color=7)),
        Line(points=[25,-20; 25,25; 75,75], style(color=0)),
        Line(points=[-75,-20; 25,-20; 75,30], style(color=7)),
        Line(points=[25,-75; 25,25; -31,25], style(color=7)),
        Line(points=[25,25; 30,30], style(color=7))
        ));
end Tank;

end Components;



package Controllers

model PumpCntr
   Interfaces.Signal signal(nSignal=1)
        annotation (extent=[-10,80; 10,100]);
   parameter SI.Time event1 = 240,
                     event2 = 360,
                     event3 = 720,
                     event4 = 900;
   parameter SI.MassFlowRate flow1 = 10,
                             flow2 = 10;
   SI.MassFlowRate totalMassFSP;
equation
   totalMassFSP = if      time > event4 then 0
                  else if time > event3 then -flow1
                  else if time > event2 then 0
                  else if time > event1 then flow2
                  else                       0;
   signal.s = {totalMassFSP};
  annotation (
    Coordsys(extent=[-100, -100; 100, 100]),
    Documentation(info="Controller of the pump"),
    Icon    (
       Rectangle(extent=[-80,-80; 80,80], style(color=0,pattern=2)),
       Text(string="cntrl", extent=[-60,-60; 60,60],
            style(color=0))
       ));
end PumpCntr;
```

Modelica Code 8.6: Two-tank system (6/8).

```
model SourceCntr
   parameter Integer nComp = 2;
   Interfaces.Signal signal(nSignal=nComp+2)
        annotation (extent=[-10,80; 10,100]);
   parameter SI.Time event1 = 180,
                     event2 = 840;
   parameter SI.MassFlowRate flow1 = 40,
                             flow2 = 40;
   SI.MassFlowRate totalMassFSP;
   Real massFractSP[nComp];
   SI.Temperature tempFSP;
equation
   totalMassFSP = if      time > event2 then flow2
                  else if time > event1 then 0
                  else                       -flow1;
   massFractSP = {0.25, 0.75};
   tempFSP = 300;
   signal.s[1:nComp] = massFractSP;
   signal.s[nComp+1] = totalMassFSP;
   signal.s[nComp+2] = tempFSP;
  annotation (
    Coordsys(extent=[-100, -100; 100, 100]),
    Documentation(info="Controller of the liquid source"),
    Icon    (
       Rectangle(extent=[-80,-80; 80,80], style(color=0,pattern=2)),
       Text(string="cntrl", extent=[-60,-60; 60,60],
            style(color=0))
       ));
end SourceCntr;

model HeatCntr
   Interfaces.Signal signal(nSignal=1)
        annotation (extent=[-10,80; 10,100]);
   parameter SI.EnergyFlowRate heatFlowSP = 2.5E5;
   parameter SI.Time event1 = 360,
                     event2 = 720;
   SI.EnergyFlowRate heatFSP;
equation
   heatFSP = if time > event1 and time < event2
             then -heatFlowSP
             else 0;
   signal.s = {heatFSP};
  annotation (
    Coordsys(extent=[-100, -100; 100, 100]),
    Documentation(info="Controller of the heat source"),
    Icon    (
       Rectangle(extent=[-80,-80; 80,80], style(color=0,pattern=2)),
       Text(string="cntrl", extent=[-60,-60; 60,60],
            style(color=0))
       ));
end HeatCntr;

end Controllers;
```

Modelica Code 8.7: Two-tank system (7/8).

```
model Plant
    parameter Integer nComp = 2;
    // Specific heat capacity.  [1] Kerosene, [2] Benzene;
    parameter Real CpCoefs[nComp,2] = [446, 5.36; 325, 4.60];
    // Density
    parameter SI.Density density[nComp] = {760,849};
    // Tank volumes
    parameter SI.Volume Tank1Vol = 10, Tank2Vol = 1;
    Components.SourceLiq  sourceL(nComp=nComp, CpCoefs=CpCoefs)
          annotation(extent=[-100,-10; -60,30]);
    Components.Tank tank1(nComp=nComp, CpCoefs=CpCoefs,
          density=density, volumeTank=Tank1Vol)
          annotation(extent=[-60,40; -20,80]);
    Components.PumpLiq pumpLiq(nComp=nComp, CpCoefs=CpCoefs)
          annotation(extent=[-20,-10; 20,30]);
    Components.Tank tank2(nComp=nComp, CpCoefs=CpCoefs,
          density=density, volumeTank=Tank2Vol)
          annotation(extent=[10,40; 50,80]);
    Components.SourceHeat sourceQ
          annotation(extent=[60,40; 100,80]);
    Controllers.SourceCntr sourceCntr
          annotation(extent=[-100,-70; -60,-30]);
    Controllers.PumpCntr   pumpCntr
          annotation(extent=[-20,-70; 20,-30]);
    Controllers.HeatCntr   heatCntr
          annotation(extent=[60,-70; 100,-30]);
equation
  connect(sourceL.liq, tank1.liq)
    annotation (  points=[-62,10; -52,10; -40,10; -40,42],
          style(color=3, rgbcolor={0,0,255}));
  connect(pumpLiq.liqIn, tank1.liq)
    annotation (  points=[-18,10; -40,10; -40,42],
          style(color=3, rgbcolor={0,0,255}));
  connect(sourceCntr.signal, sourceL.signal)
    annotation (  points=[-80,-32; -80,-32; -80,-8; -80,-8],
          style(color=0, rgbcolor={0,0,0}));
  connect(pumpCntr.signal, pumpLiq.signal)
    annotation (  points=[0,-32; 0,-32; 0,-8],
          style(color=0, rgbcolor={0,0,0}));
  connect(heatCntr.signal, sourceQ.signal)
    annotation (  points=[80,-32; 80,-32; 80,42],
          style(color=0, rgbcolor={0,0,0}));
  connect(pumpLiq.liqOut, tank2.liq)
    annotation (  points=[18,10; 24,10; 30,10; 30,42],
          style(color=3, rgbcolor={0,0,255}));
  connect(tank2.heat, sourceQ.heat)
    annotation (  points=[48,60; 55,60; 62,60],
          style(color=1, rgbcolor={255,0,0}));

  annotation (experiment(StopTime=1200), experimentSetupOutput);
end Plant;

end TwoTanks;
```

Modelica Code 8.8: Two-tank system (8/8).

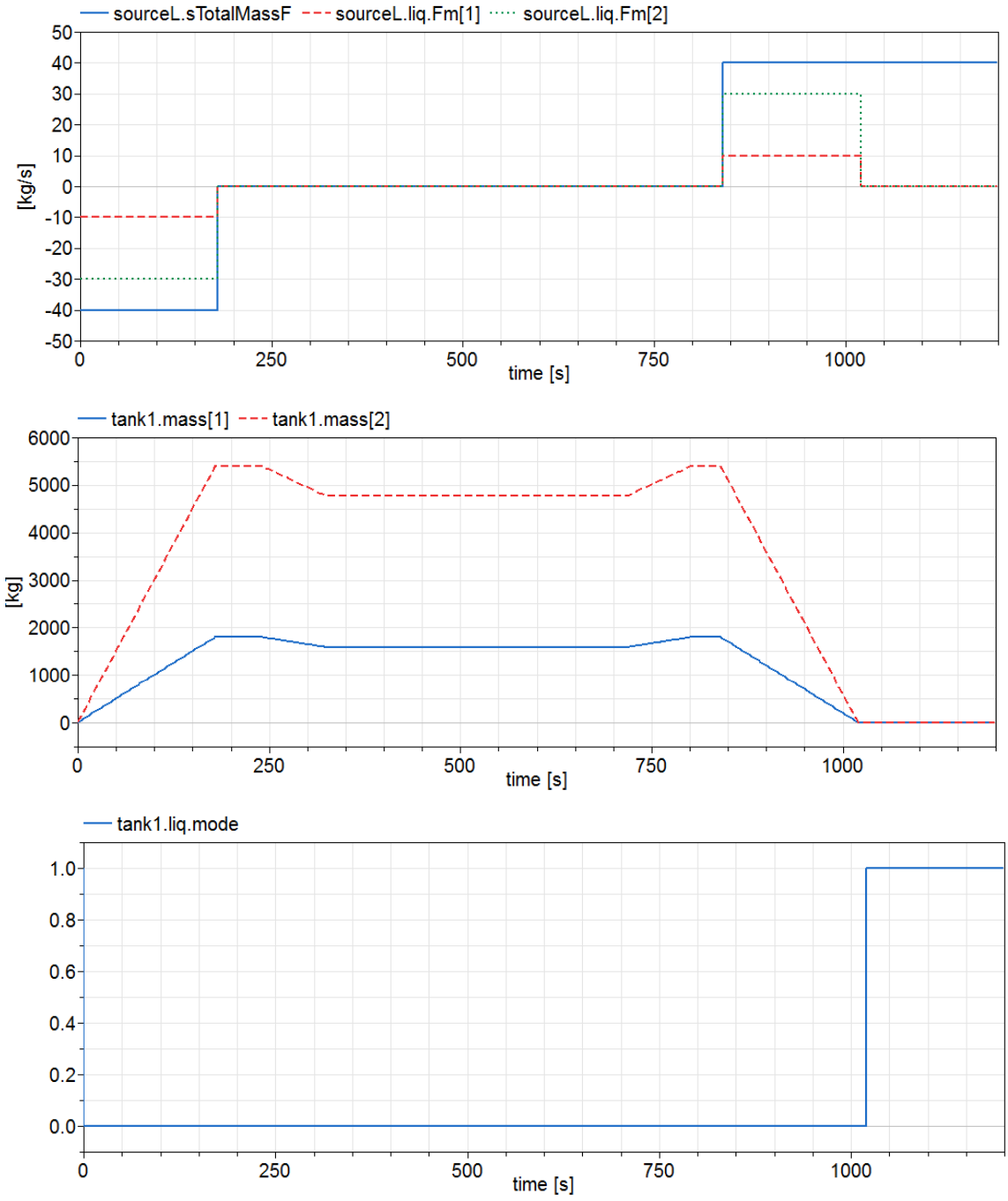


Figure 8.3: Setpoint and actual mass flow rate provided by the liquid source (top); mass of liquid components stored in Tank 1 (middle); and Tank 1 mode (bottom).

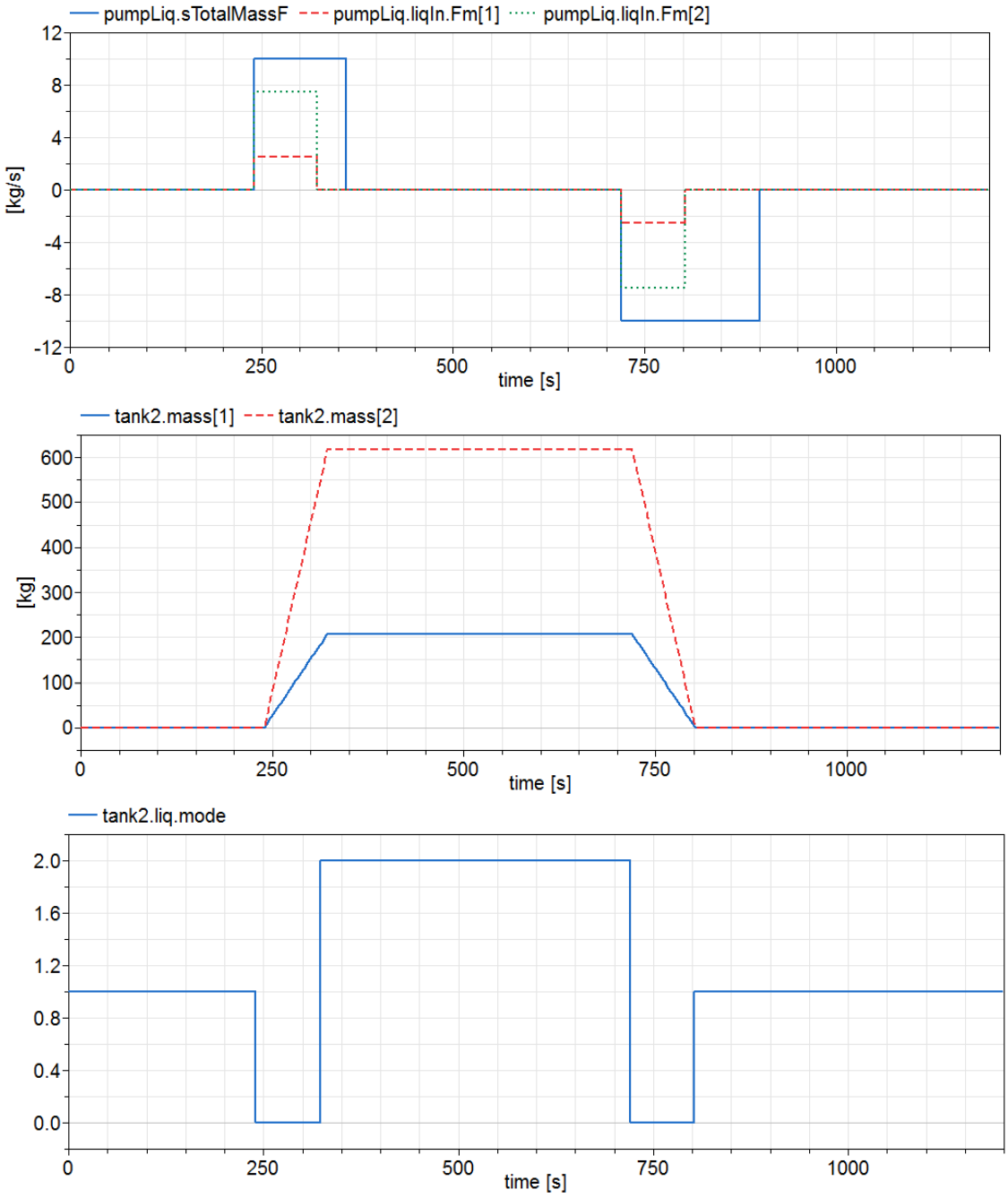


Figure 8.4: Setpoint and actual mass flow rate through the pump (top); mass of liquid components stored in Tank 2 (middle); and Tank 2 mode (bottom).

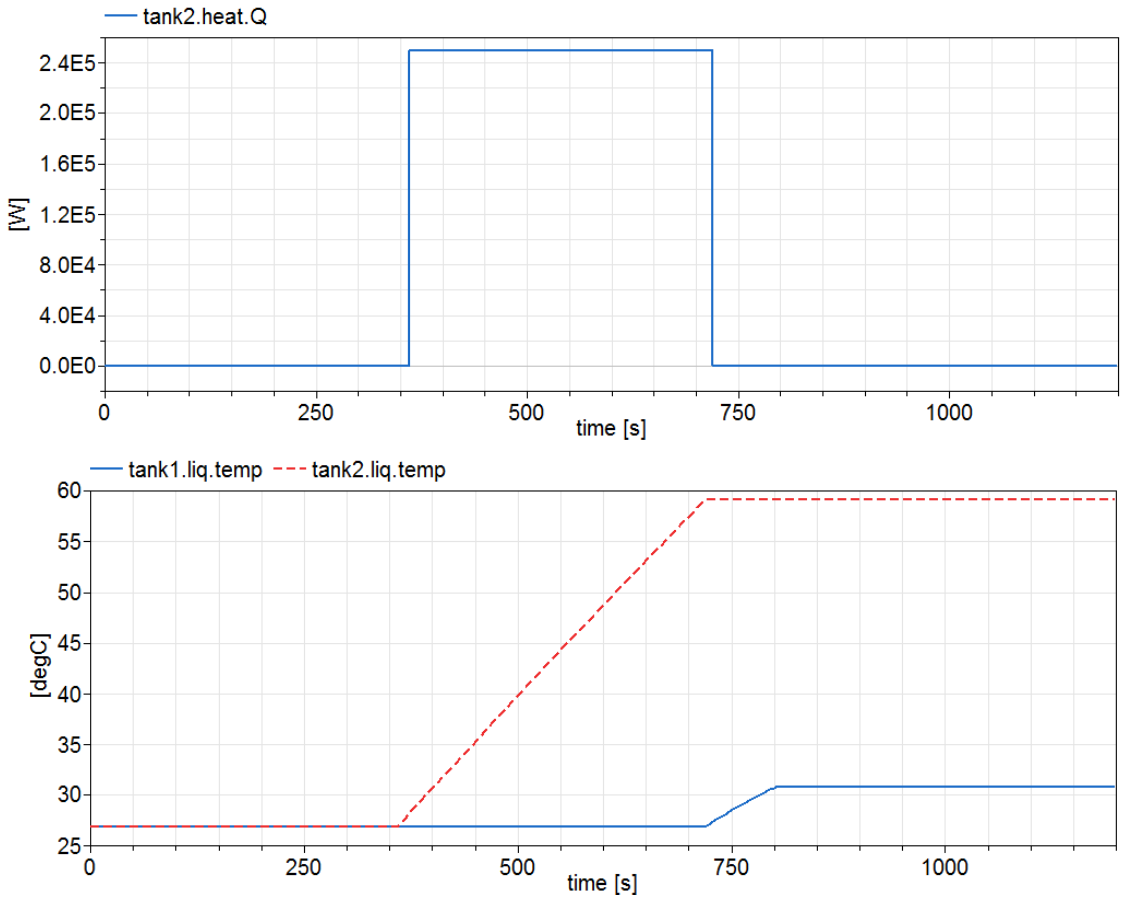


Figure 8.5: Heat flow rate provided by the heater (top); and temperatures of the liquid (bottom).

Double-pipe heat exchanger

Purpose of this assignment

- Practice modeling of cocurrent and countercurrent heat exchangers, with local variations in physical properties and heat transfer coefficients.
- Use array variables in Modelica.
- Facilitate the numerical solution of models with systems of simultaneous nonlinear equations.

9.1 System description

A heat exchanger is composed of two straight, concentric pipes. The inner pipe is made of copper, with thermal conductivity $k_w = 381$ W/(m·K), density $\rho_w = 8950$ kg/m^3, and specific heat capacity $C_{p,w} = 383$ J/(kg·K). Its wall is 1.65 mm thick: the inner and outer diameters are $D_1 = 18.92$ mm and $D_2 = 22.22$ mm respectively. The outer pipe is made of steel, with inner diameter $D_3 = 38.1$ mm. It is assumed that it is perfectly insulated from the environment. The heat exchanger is L meters long.

The heat exchanger is employed to cool down a gas mixture of carbon dioxide and sulfur dioxide, using water as coolant. The gas circulates through the inner pipe and the cooling water through the outer pipe. The mass flow rate of water is controlled to be approximately thirty times the mass flow rate of gas. Typical values of the gas and water input temperatures are 130 °C and 18 °C respectively.

The water properties are the following: density $\rho_l = 996$ kg/m^3, specific heat capacity $C_{p,l} = 4185$ J/(kg·K), thermal conductivity $k_l = 0.61$ W/(m·K), and dynamic viscosity $\mu_l = \exp(-10.547 + \frac{541.69}{T-144.53})$ kg/(m·s), where T is the temperature in K.

The gas is a mixture of carbon dioxide and sulfur dioxide. The specific heat capacities are $C_{p,CO_2} = 837$ J/(kg·K) and $C_{p,SO_2} = 657$ J/(kg·K). Assuming that the molar fractions are approximately 0.5, the average dynamic viscosity of the gas mixture in the temperature interval of interest is $\mu_g = 1.55 \cdot 10^{-5}$ kg/(m·s), and the thermal conductivity is $k_g = 0.014$ W/(m·K).

The heat transfer coefficients by convection between the inner pipe wall and the gas (h_g), and between the inner pipe wall and the water (h_l), can be calculated from the Dittus-Boelter correlations shown below. The convective heat transfer coefficients h_g and h_l are expressed in W/(m^2·K). The v_g and v_l variables represent the velocity of the gas and water respectively, and $|v_g|$ and $|v_l|$ their absolute values.

$$\frac{h_g \cdot D_1}{k_g} = 0.023 \cdot \left(\frac{D_1 \cdot |v_g| \cdot \rho_g}{\mu_g}\right)^{0.8} \cdot \left(\frac{C_{p,g} \cdot \mu_g}{k_g}\right)^{0.3} \tag{9.1}$$

$$\frac{h_l \cdot D_2}{k_l} = 0.023 \cdot \left(\frac{D_2 \cdot |v_l| \cdot \rho_l}{\mu_l}\right)^{0.8} \cdot \left(\frac{C_{p,l} \cdot \mu_l}{k_l}\right)^{0.4} \tag{9.2}$$

The mass flow rates of gas (F_g^{mass}) and water (F_l^{mass}) are time-dependent quantities, but independent of the spatial coordinates.

The product of the gas density and velocity ($\rho_g \cdot v_g$), and the product of the water density and velocity ($\rho_l \cdot v_l$), intervene in Eqs. (9.1) and (9.2) respectively. The mass flow rates, densities and velocities are related by the following equations.

$$F_g^{mass} = \rho_g \cdot v_g \cdot \pi \cdot \left(\frac{D_1}{2}\right)^2 \tag{9.3}$$

$$F_l^{mass} = \rho_l \cdot v_l \cdot \pi \cdot \left(\left(\frac{D_3}{2}\right)^2 - \left(\frac{D_2}{2}\right)^2\right) \tag{9.4}$$

Replacing Eqs. (9.3) and (9.4) in Eqs. (9.1) and (9.2), it is obtained:

$$\frac{h_g \cdot D_1}{k_g} = 0.023 \cdot \left(\frac{\left|F_g^{mass}\right|}{\mu_g \cdot \frac{\pi}{4} \cdot D_1}\right)^{0.8} \cdot \left(\frac{C_{p,g} \cdot \mu_g}{k_g}\right)^{0.3} \tag{9.5}$$

$$\frac{h_l \cdot D_2}{k_l} = 0.023 \cdot \left(\frac{D_2 \cdot |F_l^{mass}|}{\mu_l \cdot \frac{\pi}{4} \cdot (D_3^2 - D_2^2)}\right)^{0.8} \cdot \left(\frac{C_{p,l} \cdot \mu_l}{k_l}\right)^{0.4} \tag{9.6}$$

9.2 Tasks

1. Write the steady-state energy balances in the control volumes obtained by performing a spatial discretization in the flow direction of the gas and water streams, and the inner pipe wall. The temperature of the medium (gas, liquid or metal) contained inside each control volume is considered as a time-dependent, homogeneous quantity. Name the quantities as in Figure 9.1.

2. Describe the model in Modelica as an atomic model. Ramp up linearly the mass flow rate of gas from 0.002 to 0.006 kg/s, operating the exchanger both in cocurrent and in countercurrent, for three input gas temperatures: 110, 130 and 150 °C (see Figure 9.2). The mass flow rate of water is 30 times the mass flow rate of gas. The input temperature of the liquid is 18 °C.

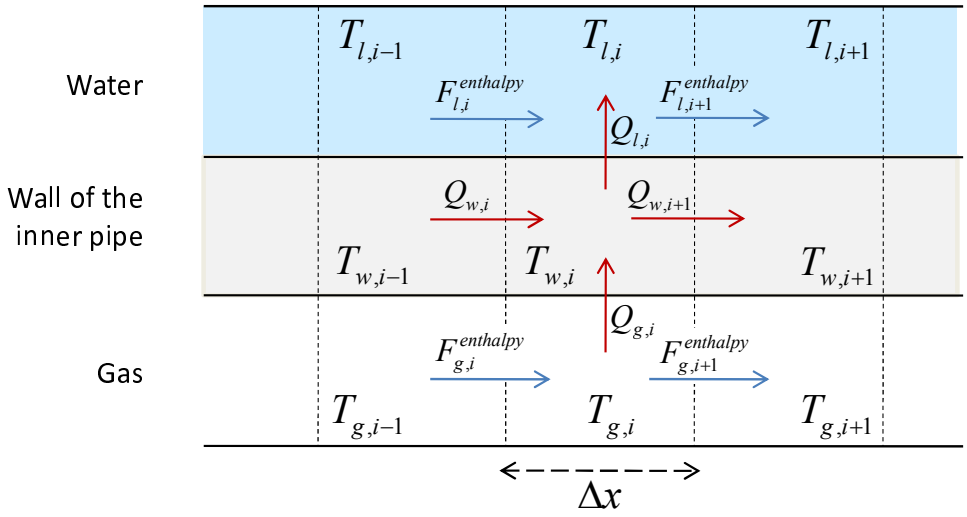


Figure 9.1: Detail of the spatial discretization into control volumes.

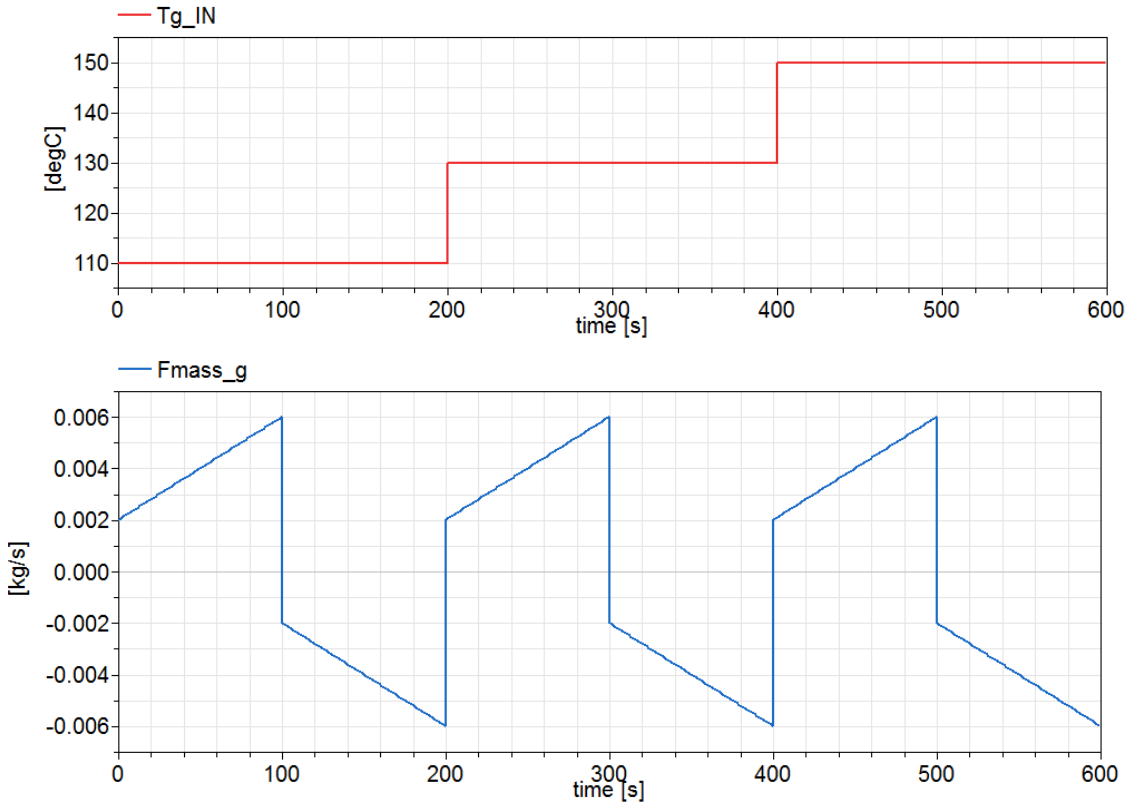


Figure 9.2: Input temperature and mass flow rate of the gas. It is a full-factorial experiment with 2 factors. The first factor has three levels (110, 130, 150 °C) and the second factor has two levels (cocurrent, countercurrent).

9.3 Solution to Task 1

The steady-state energy balances on the control volumes are posed as follows.

- The water stream is divided into N equal control volumes in the flow direction. The water temperature in the i-th control volume ($T_{l,i}$) is calculated imposing that the input flow of enthalpy ($F_{l,i}^{enthalpy}$), minus the output flow of enthalpy ($F_{l,i+1}^{enthalpy}$), plus the heat transferred from the wall of the inner pipe ($Q_{l,i}$), equals zero.

$$F_{l,i}^{enthalpy} - F_{l,i+1}^{enthalpy} + Q_{l,i} = 0 \qquad \text{for} \quad i = 1, \ldots, N \tag{9.7}$$

- The wall of the inner pipe is divided into N equal control volumes along the pipe length. The wall temperature in the i-th control volume ($T_{w,i}$) is calculated imposing that the heat transfer by conduction from the wall's adjacent control volumes ($Q_{w,i}$, $Q_{w,i+1}$), plus the convective heat transfer from the gas ($Q_{g,i}$), minus the convective heat transfer to the water ($Q_{l,i}$), equals zero.

$$Q_{w,i} - Q_{w,i+1} + Q_{g,i} - Q_{l,i} = 0 \qquad \text{for} \quad i = 1, \ldots, N \tag{9.8}$$

- The gas stream is divided into N equal control volumes in the flow direction. The temperature of the gas in the i-th control volume ($T_{g,i}$) is calculated imposing that the input flow of enthalpy ($F_{g,i}^{enthalpy}$), minus the output flow of enthalpy ($F_{g,i+1}^{enthalpy}$), minus the heat transferred to the wall of the inner pipe ($Q_{g,i}$), equals zero.

$$F_{g,i}^{enthalpy} - F_{g,i+1}^{enthalpy} - Q_{g,i} = 0 \qquad \text{for} \quad i = 1, \ldots, N \tag{9.9}$$

The length of the control volumes (Δx) can be calculated dividing the pipe length by the number of control volumes.

$$\Delta x = \frac{L}{N} \tag{9.10}$$

The convective heat transfer between the gas and the pipe wall, and between the pipe wall and the water are related with the corresponding temperature differences as shown in Eqs. (9.11) and (9.12).

$$T_{g,i} - T_{w,i} = \frac{1}{h_g \cdot \pi \cdot D_1 \cdot \Delta x} \cdot Q_{g,i} \qquad \text{for} \quad i = 1, \ldots, N \tag{9.11}$$

$$T_{w,i} - T_{l,i} = \frac{1}{h_l \cdot \pi \cdot D_2 \cdot \Delta x} \cdot Q_{l,i} \qquad \text{for} \quad i = 1, \ldots, N \tag{9.12}$$

The heat conduction between adjacent control volumes of the inner pipe wall is described by the following equations

$$T_{w,i} - T_{w,i+1} = \frac{\Delta x}{k_w \cdot \pi \cdot (D_2^2 - D_1^2)} \cdot Q_{w,i+1} \qquad \text{for} \quad i = 1, \ldots, N-1 \tag{9.13}$$

The boundary conditions are:

$$Q_{w,1} = Q_{w,N+1} = 0 \tag{9.14}$$

The flows of enthalpy can be calculated from Eqs. (9.15) and (9.16). Observe that the temperature of the liquid or gas that flows between two control volumes is assumed to be equal to the temperature in the control volume located upstream.

$$F_{l,i}^{enthalpy} = \begin{cases} F_l^{mass} \cdot C_{p,l} \cdot T_{l,i-1} & \text{if } F_l^{mass} \geq 0 \\ F_l^{mass} \cdot C_{p,l} \cdot T_{l,i} & \text{if } F_l^{mass} < 0 \end{cases} \qquad \text{for } i = 1, \ldots, N+1 \tag{9.15}$$

$$F_{g,i}^{enthalpy} = \begin{cases} F_g^{mass} \cdot C_{p,g} \cdot T_{g,i-1} & \text{if } F_g^{mass} \geq 0 \\ F_g^{mass} \cdot C_{p,g} \cdot T_{g,i} & \text{if } F_g^{mass} < 0 \end{cases} \qquad \text{for } i = 1, \ldots, N+1 \tag{9.16}$$

The $T_{g,0}$ and $T_{g,N+1}$ variables represent the input temperature of the gas while $F_g^{mass} \geq 0$ and $F_g^{mass} < 0$ respectively. Analogously, $T_{l,0}$ and $T_{l,N+1}$ represent the input temperature of the water while $F_l^{mass} \geq 0$ and $F_l^{mass} < 0$ respectively.

9.4 Solution to Task 2

The Modelica description of the model and the experiment is Modelica Codes 9.1 – 9.4. Observe that the six experimental runs are executed in sequence (see again Figure 9.2). The value of the `expRun` variable indicates the experimental run, as shown in the following table.

Run	Factor 1	Factor 2
1	110 °C	cocurrent
2	110 °C	countercurrent
3	130 °C	cocurrent
4	130 °C	countercurrent
5	150 °C	cocurrent
6	150 °C	countercurrent

The Boolean array variables `levels_Tg_IN` and `levels_Fmass_g` indicate the actual level of the corresponding factor. For example, `levels_Tg_IN` equals {*true*, *false*, *false*} while the gas input temperature is 110°C; and `levels_Fmass_g` equals {*true*, *false*} while the heat exchanger is operated in the cocurrent configuration.

```
model DoublePipeHeatExchanger

  import SI = Modelica.SIunits;
  import Modelica.Constants.pi;

  parameter Integer N = 20 "Number of control volumes";

  parameter SI.Length D_1 = 18.92e-3 "Inner diameter of inner pipe";
  parameter SI.Length D_2 = 22.22e-3 "Outer diameter of inner pipe";
  parameter SI.Length D_3 = 38.10e-3 "Inner diameter of outer pipe";
  parameter SI.Length L = 1 "Pipe length";

  // Control volume length
  parameter SI.Length DeltaX = L/N "Control volume length";

  // Thermal conductivities
  parameter SI.ThermalConductivity k_l = 0.61  "Thermal conductivity of water";
  parameter SI.ThermalConductivity k_g = 0.014 "Thermal conductivity of gas";
  parameter SI.ThermalConductivity k_w = 381 "Thermal conductivity of wall pipe";

  // Convective heat transfer coefficients
  SI.CoefficientOfHeatTransfer h_g "Heat transfer coeff. of gas";
  SI.CoefficientOfHeatTransfer h_l[N](start=50*ones(N), fixed=false)
      "Heat transfer coeff. of water";

  // Dynamic viscosities
  SI.DynamicViscosity dVisco_l[N](start=0.001*ones(N), fixed=false)
      "Dynamic viscosity of water";
  parameter SI.DynamicViscosity dVisco_g = 1.55e-5
      "Dynamic viscosity of gas";

  // Input temperatures
  SI.Temperature Tg_IN "Input temperature of gas";
  SI.Temperature Tl_IN "Input temperature of water";

  // Mass flow rates
  SI.MassFlowRate Fmass_l "Mass flow rate of water";
  SI.MassFlowRate Fmass_g "Mass flow rate of gas";

  // Specific heat capacities
  parameter SI.SpecificHeatCapacity Cp_l = 4185;
  parameter SI.SpecificHeatCapacity Cp_w = 383;
  parameter SI.SpecificHeatCapacity Cp_g = 0.5*(837+657);

  // Heat flow rate
  SI.HeatFlowRate Q_l[N] "Convective heat from pipe wall to water";
  SI.HeatFlowRate Q_w[N+1] "Heat conduction within the inner pipe";
  SI.HeatFlowRate Q_g[N] "Convective heat from gas to inner pipe";
```

Modelica Code 9.1: Double-pipe heat exchanger (1/4).

```
  // Temperatures
  SI.Temperature T_l[N](start=(273.15+18)*ones(N), fixed=false)
      "Liquid temperature";
  SI.Temperature T_g[N](start=(273.15+130)*ones(N), fixed=false)
      "Gas temperature";
  SI.Temperature T_w[N](start=(273.15+70)*ones(N), fixed=false)
      "Inner pipe wall temperature";

  // EnthalpyFlowRates
  SI.EnthalpyFlowRate Fenthalpy_l[N+1] "Enthalpy flow rate of water";
  SI.EnthalpyFlowRate Fenthalpy_g[N+1] "Enthalpy flow rate of gas";

  // Experiment
  parameter SI.Time Trun = 100 "How long an experimental run takes";
  SI.Time TstartedRun(start=0, fixed=true)
      "Stating time of the actual experimental run";
  Real expRun(start=1, fixed=true)
      "Number of the actual experimental run";
  Boolean levels_Tg_IN[3];
  Boolean levels_Fmass_g[2];

equation

  // Dynamic viscosity of water
  for i in 1:N loop
    dVisco_l[i] = exp(-10.547+541.69/(T_l[i]-144.53));
  end for;

  // Dittus-Boelter correlations
  h_g*D_1/k_g = 0.023*( abs(Fmass_g)/(dVisco_g*pi/4*D_1)) ^0.8 *
                ( Cp_g*dVisco_g/k_g) ^0.3;
  for i in 1:N loop
     h_l[i]*D_2/k_l =
        0.023*( (D_2*abs(Fmass_l))/(dVisco_l[i]*pi/4*(D_3^2-D_2^2))) ^0.8 *
                ( Cp_l*dVisco_l[i]/k_l) ^0.4;
  end for;
```

Modelica Code 9.2: Double-pipe heat exchanger (2/4).

```
// Steady-state energy balances
for i in 1:N loop
   Fenthalpy_l[i] - Fenthalpy_l[i+1] + Q_l[i] = 0;
   Q_w[i] - Q_w[i+1] + Q_g[i] - Q_l[i] = 0;
   Fenthalpy_g[i] - Fenthalpy_g[i+1] - Q_g[i] = 0;
end for;

// Convective heat transfer
for i in 1:N loop
   T_g[i] - T_w[i] = Q_g[i]/(h_g*pi*D_1*DeltaX);
   T_w[i] - T_l[i] = Q_l[i]/(h_l[i]*pi*D_2*DeltaX);
end for;

// Heat conduction in the wall of the inner pipe
for i in 1:(N-1) loop
   T_w[i] - T_w[i+1] = Q_w[i+1]*DeltaX/(k_w*pi*(D_2^2 - D_1^2));
end for;
Q_w[1] = 0;
Q_w[N+1] = 0;

// Enthalpy flow rates of water
if Fmass_l >= 0 then
  Fenthalpy_l[1] = Fmass_l*Cp_l*Tl_IN;
  for i in 2:(N+1) loop
     Fenthalpy_l[i] = Fmass_l*Cp_l*T_l[i-1];
  end for;
else
  for i in 1:N loop
     Fenthalpy_l[i] = Fmass_l*Cp_l*T_l[i];
  end for;
  Fenthalpy_l[N+1] = Fmass_l*Cp_l*Tl_IN;
end if;

// Enthalpy flow rates of gas
if Fmass_g >= 0 then
  Fenthalpy_g[1] = Fmass_g*Cp_g*Tg_IN;
  for i in 2:(N+1) loop
     Fenthalpy_g[i] = Fmass_g*Cp_g*T_g[i-1];
  end for;
else
  for i in 1:N loop
     Fenthalpy_g[i] = Fmass_g*Cp_g*T_g[i];
  end for;
  Fenthalpy_g[N+1] = Fmass_g*Cp_g*Tg_IN;
end if;
```

Modelica Code 9.3: Double-pipe heat exchanger (3/4).

```
  // ------------------
  // Boundary conditions
  // ------------------

  // Input liquid temperature and mass flow rate
  Tl_IN = 273.15 + 18;
  Fmass_l = 30*abs(Fmass_g);

  // Experimental levels - Full factorial experiment
  levels_Tg_IN = {
      expRun == 1 or expRun == 2,
      expRun == 3 or expRun == 4,
      expRun == 5 or expRun == 6};
  levels_Fmass_g = {
      expRun == 1 or expRun == 3 or expRun == 5,
      expRun == 2 or expRun == 4 or expRun == 6};

  // Factor: Tg_IN
  // Levels: 110, 130, 150 degC
  if levels_Tg_IN[1] then
    Tg_IN = 273.15 + 110;
  elseif levels_Tg_IN[2] then
    Tg_IN = 273.15 + 130;
  else
    Tg_IN = 273.15 + 150;
  end if;

  // Factor: Fmass_g
  // Levels: cocurrent, countercurrent
  if levels_Fmass_g[1] then
    Fmass_g = 0.004+0.002*(-1+2*(time-TstartedRun)/Trun);
  else
    Fmass_g = -0.004-0.002*(-1+2*(time-TstartedRun)/Trun);
  end if;

  // Experiment running
  when sample(Trun,Trun) then
    expRun = pre(expRun) + 1;
    TstartedRun = time;
  end when;

  annotation (experiment(StopTime=600));
end DoublePipeHeatExchanger;
```

Modelica Code 9.4: Double-pipe heat exchanger (4/4).

Assignment 10

Cellular Automata – The Game of Life

Purpose of this assignment

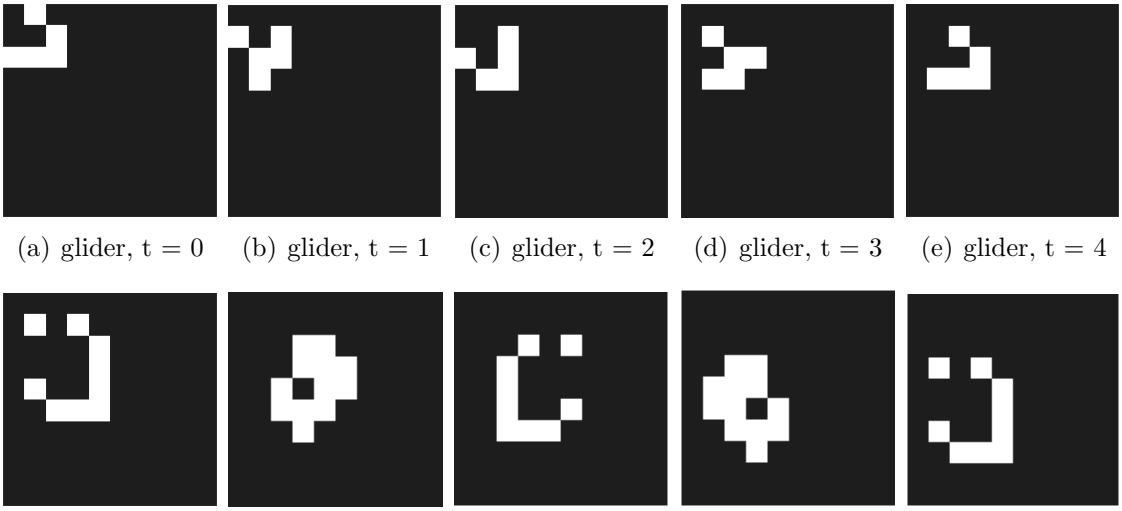
- Modeling and simulation of discrete-event models using Modelica.
- Experience and practice with the external function interface of Modelica.
- Implement graphical animations for the simulations using visualizers of the *Modelica.Mechanics.Multibody* library, and gnuplot.
- Analyze the simulation performance.

10.1 System description

Cellular Automata (CA) are discrete-time, dynamic models initially proposed by John von Neumann for the study of self-reproducing automata (von Neumann 1966). These models are represented as a grid of identical volumes, named cells, that can be in any finite number of dimensions (Ilachinski 2001).

The state of each cell in the automata is discrete, and it is updated at discrete time steps during the simulation following a transition function or rule. This rule constitutes a function of the current state of the cell and the state of its neighbors, and defines the state of the cell for the next time step (Schiff 2008).

The neighborhood of a cell is usually composed of a selection of its surrounding cells, but not necessarily. It can be defined in different ways, such as the Moore's neighborhood that includes all the surrounding cells; the von Neumann's neighbor-



(a) glider, t = 0 (b) glider, t = 1 (c) glider, t = 2 (d) glider, t = 3 (e) glider, t = 4

(f) torus, t = 0 (g) torus, t = 1 (h) torus, t = 2 (i) torus, t = 3 (j) torus, t = 4

Figure 10.1: Game of Life simulations.

hood that includes the cells adjoining the four faces of one cell; or the extended von Neumann's that also includes each cell just beyond one of the four adjoining cells (Wolfram 2002).

The Game of Life is a two-dimensional cellular automata model proposed by John H. Conway in 1970, with the purpose of studying the evolution of the different patterns that arise from different initial configurations of the cellular space. The behavior of the model is as follows. Each cell can be in one of two states, alive or dead, calculated using the following rules (considering the Moore's neighborhood, as described above):

1. A living cell remains alive if it has two or three living neighbors.
2. A living cell dies if it has less than two living neighbors (isolation).
3. A living cell dies if it has more that three living neighbors (overpopulation).
4. A dead cell becomes alive if it has exactly three living neighbors (reproduction).

As an example of behavior, the first five steps of the simulation of two initial states, named *glider* and *torus*, are shown in Figure 10.1.

– The *glider* model corresponds to the initial cells: [1,2; 2,3; 3,1; 3,2; 3,3]. This model evolves in a periodical diagonal movement from the top-left area of the cellular space to the bottom-right.

– The *torus* model corresponds to the initial cells: [2,2; 2,4; 3,5; 4,5; 5,5; 6,3; 6,4; 6,5; 5,2]. This model evolves in a periodical vertical movement from the top area of the cellular space to the bottom area.

10.2 Tasks

1. Write a Modelica model that simulates the Game of Life.
2. Evaluate the maximum size of cellular space that can be simulated using the model written in the previous task.
3. Include a graphical animation in the model.
4. Substitute the code that represents the behavior of the model by external code written in C. The graphical animation can also be substituted by using external libraries of programs (e.g., gnuplot).
5. Compare the performance of the models developed in Tasks 1 and 4, in terms of size of the cellular space and execution time.

10.3 Solution to Task 1

The model, shown in Modelica Code 10.1, has been developed using Dymola. It includes two matrices of size $n \times n$, named `state` and `newstate`, that are used to store the current and new state values during each iteration.

The execution is controlled using the `sample()` operator, performing one simulation step per simulated second. The neighbors variable is used to compute the number of living neighbors for each cell, and it is used to compute the new state following the behavior of the model.

In order to validate the results, the evolution of the state of cells [1,1] and [2,1] is shown in Figure 10.2.

```
model GameofLife
  parameter Integer n=10; //dimension of the cellular space
  parameter Integer initState[:,2]= [1, 2; 2, 3; 3, 1; 3, 2; 3, 3]; //glider
  Integer state[n,n]; // current state of cells
  Integer newstate[n,n]; // new state of cells
  Integer neighbors; // number of neighbors alive
initial algorithm
    // set initial configuration for the cells
    for i in 1:size(initState,1) loop
      state[initState[i,1],initState[i,2]] := 1;
    end for;
algorithm
  when sample(1,1) then // periodic iterations or steps
    for i in 0:n-1 loop // two for loops to evaluate all cells in the space
      for j in 0:n-1 loop
        neighbors := state[mod(i-1,n)+1,mod(j-1,n)+1]+state[i+1,mod(j-1,n)+1]+
                     state[mod(i+1,n)+1,mod(j-1,n)+1]+state[mod(i-1,n)+1,j+1]+
                     state[mod(i+1,n)+1,j+1]+state[mod(i-1,n)+1,mod(j+1,n)+1]+
                     state[i+1,mod(j+1,n)+1]+state[mod(i+1,n)+1,mod(j+1,n)+1];
        if state[i+1,j+1] == 1 then // if alive
          if neighbors < 2  or neighbors > 3 then
            newstate[i+1,j+1] := 0; // dies because of isolation or overpopulation
                                    // (less than 2 or more than 3 neighbors alive)
          else
            newstate[i+1,j+1] := 1; // otherwise still alive
          end if;
        elseif neighbors == 3 then // if not alive
          newstate[i+1,j+1] := 1; // becomes alive because of reproduction
                                  // (3 neighbors alive)
        end if;
      end for;
    end for;
    state := newstate; // update state
  end when;
end GameofLife;
```

Modelica Code 10.1: The Game of Life model.

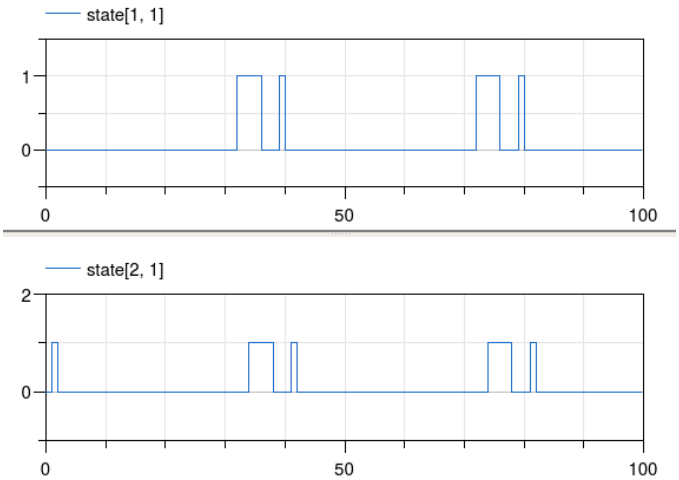


Figure 10.2: Game of Life simulation results (cells [1,1] and [2,1]).

10.4 Solution to Task 2

The goal is to evaluate the maximum size of cellular space that can be simulated using the model written in the previous task. To this end, several runs can be performed with an increasing cellular space size. The results obtained using Dymola 2018 on an Intel Core i7-4720HQ (2.60 GHz with 8 cores) machine, with 16 Gb of memory, are shown in Table 10.1 and Figure 10.3.

Table 10.1: Simulation performance of Modelica Code 10.1.

Size of Cellular Space	10^2	50^2	100^2	200^2	300^2
Size of Compiled Model	56 kb	1.3 Mb	5.2 Mb	22 Mb	49 Mb
Simulation Time	0.0313 s	12.8 s	203 s	3450 s	16600 s

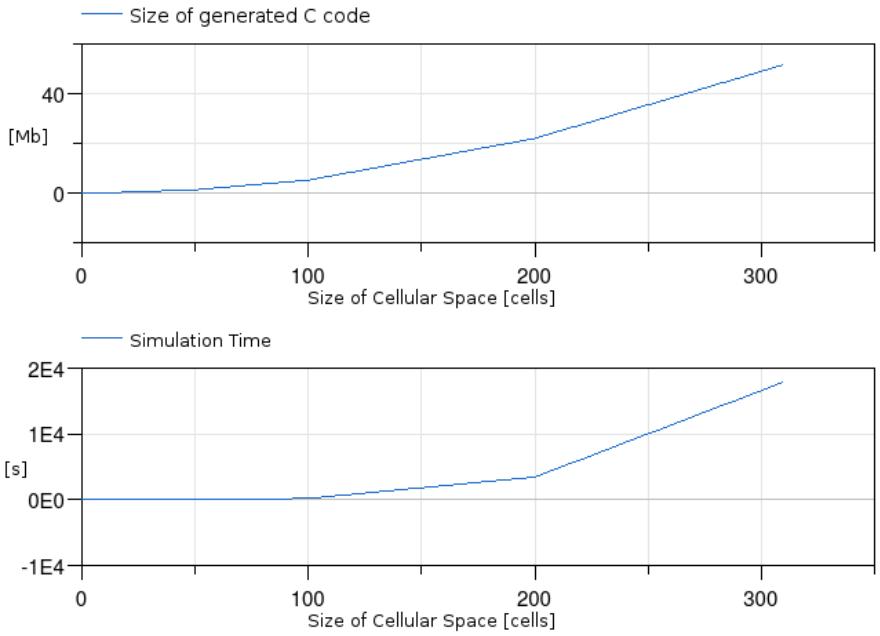


Figure 10.3: Simulation performance.

10.5 Solution to Task 3

The graphical animation of the model simulation is implemented by employing the `FixedShape`, `Fixed` and `World` models of the *MultiBody* library of the Modelica Standard Library (MSL). The model shown in Modelica Codes 10.2 and 10.3 has been developed using Dymola 2018 and requires MSL version 2.2.1 or later. Screenshots of the automaton animation obtained simulating Modelica Codes 10.2 and 10.3 during 16 s are shown in Figure 10.4.

```
model GameofLifeAnim
  parameter Integer n=10;    // Dimension of the cellular space
  parameter Integer initState[:,2] =
          [1, 2; 2, 3; 3, 1; 3, 2; 3, 3]; // Glider
  Integer state[n,n];     // Current state of cells
  Integer newstate[n,n]; // New state of cells
  Integer neighbors;      // Number of neighbors alive

  Modelica.Mechanics.MultiBody.Visualizers.FixedShape fixedShape[n,n](
    each shapeType = "box",
    each length =  0.1,
    each height = 0.1,
    each width = 0.1,
    r_shape = {{j/10,-i/10,0} for i in 1:n, j in 1:n},
    color = {{integer(state[i,j])*255,
              integer(state[i,j])*255,
              integer(state[i,j])*255} for i in 1:n, j in 1:n},
    each specularCoefficient = 0)
    annotation (Placement(transformation(
        extent={{0,40},{20,60}}, rotation=0)));

  Modelica.Mechanics.MultiBody.Parts.Fixed fixed
    annotation (Placement(transformation(
        extent={{-40,42},{-20,62}}, rotation=0)));

  inner Modelica.Mechanics.MultiBody.World world
    (animateWorld=false, animateGravity=false)
    annotation (Placement(transformation(
        extent={{-80,40},{-60,60}}, rotation=0)));

initial algorithm

    // Set initial configuration for the cells
    for i in 1:size(initState,1) loop
      state[initState[i,1],initState[i,2]] := 1;
    end for;
```

Modelica Code 10.2: The Game of Life model with animation (1/2).

```
algorithm
  when sample(1,1) then // Periodic iterations or steps
    for i in 0:n-1 loop // Two for loops to evaluate all cells in the space
      for j in 0:n-1 loop
        neighbors := state[mod(i-1,n)+1,mod(j-1,n)+1]+state[i+1,mod(j-1,n)+1]+
                     state[mod(i+1,n)+1,mod(j-1,n)+1]+state[mod(i-1,n)+1,j+1]+
                     state[mod(i+1,n)+1,j+1]+state[mod(i-1,n)+1,mod(j+1,n)+1]+
                     state[i+1,mod(j+1,n)+1]+state[mod(i+1,n)+1,mod(j+1,n)+1];
        if state[i+1,j+1] == 1 then // if alive
          if neighbors < 2  or neighbors > 3 then
            newstate[i+1,j+1] := 0; // Dies because of isolation or overpopulation
                                    // (less than 2 or more than 3 neighbors alive)
          else
            newstate[i+1,j+1] := 1; // Otherwise, still alive
          end if;
        elseif neighbors == 3 then // If not alive
          newstate[i+1,j+1] := 1;  // Becomes alive because of reproduction
                                   // (3 neighbors alive)
        end if;
      end for;
    end for;
    state := newstate;  // Update state
  end when;
equation
  for i in 1:n loop
    for j in 1:n loop
       connect(fixed.frame_b, fixedShape[i,j].frame_a);
    end for;
  end for;
end GameofLifeAnim;
```

Modelica Code 10.3: The Game of Life model with animation (2/2).

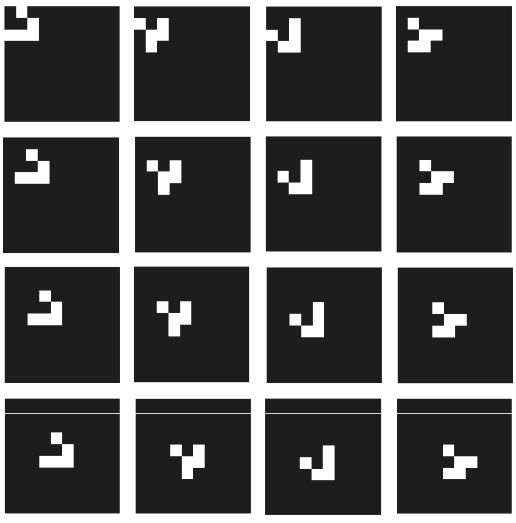


Figure 10.4: Evolution of the automaton state obtained simulating Modelica Codes 10.2 and 10.3.

10.6 Solution to Task 4

The code that represents the automaton behavior is replaced in this task by external code written in the C programming language. The graphical animation can also be substituted by using external libraries (e.g., gnuplot).

The Modelica model is Modelica Codes 10.4 and 10.5. The architecture of the package is shown in Figure 10.5.

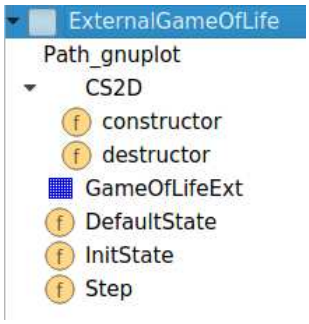


Figure 10.5: Architecture of the *ExternalGameOfLife* Modelica Package.

In Modelica, the cellular space has been described as an external object (cf. `CS2D` model shown in Figure 10.5). It includes two functions, `constructor` and `destructor`, that are used to initialize and remove the object from the simulation. These functions correspond to functions of the C code (cf. Modelica code and annotations of the model).

The `GameOfLifeExt` model instantiates the mentioned external object and uses three functions to initialize the space and perform simulation steps (cf. `DefaultState`, `InitState` and `Step` functions). Periodic simulation steps are performed using the `sample` operator, as in the previous tasks. The `Path_gnuplot` variable is used to setup the path to execute gnuplot in order to generate the graphical animation.

The C code is written in a file named `gameoflifeext.c`, listed after the Modelica code. The developed C code is used to represent the cellular space as a two-dimensional array of cells. In this implementation only the active cells in each step are evaluated, which improves the performance of the simulation.

The graphical animation of the simulation is generated using gnuplot. An output buffer is used to write the new state of the cells after every step, and then display it in the animation.

```
package ExternalGameOfLife

  //constant String Path_gnuplot="C:\\gnuplot\\bin\\gnuplot"; //WINDOWS
  constant String Path_gnuplot="/usr/bin/gnuplot -persist"; // LINUX

  class CS2D
        extends ExternalObject;

  function constructor
      input Integer nrows "Number of rows";
      input Integer ncols "Number of columns";
      input Integer plot_animation "1 to generate animation, 0 otherwise";
      input Integer plot_range "Maximum range for values in the animation";
      input Integer display_delay "Time delay for the animation";
      input String path "Path to GNUPlot";
      input String name "Name of the space";
      output CS2D s "Reference to the created space";
      external "C" s = CS2D_Create(nrows,ncols,plot_animation,
                       plot_range,display_delay,path,name);
         annotation (Include="#include \"gameoflifeext.c\"");
  end constructor;

  function destructor
        input CS2D s;
        external "C" CS2D_Delete(s);
         annotation (Include="#include \"gameoflifeext.c\"");
  end destructor;
  end CS2D;

  model GameOfLifeExt
    parameter Integer nrows=10 "Number of rows";
    parameter Integer ncols=10 "Number of columns";
    parameter Integer plot_animation=1
      "generate graphical animation (0 false / 1 true)";
    parameter Integer plot_range=1
      "Max range of output for the graphical output";
    parameter Integer display_delay=50000
      "Graphical output update delay (in microseconds)";
    parameter Integer[:, 2] init_cells=[1, 2; 2, 3; 3, 1; 3, 2; 3, 3];
    parameter String name="Game of Life (glider)";
    Integer sumactive( start = 0); // number of active cells
    Real mean; // average number of active cells over time
    CS2D s = CS2D(nrows,ncols,plot_animation,plot_range,
                  display_delay,Path_gnuplot,name);
  protected
    Boolean init(start=false, fixed=true);
```

Modelica Code 10.4: The Game of Life model using external functions (1/2).

```
  algorithm
    when initial() and not init then
      DefaultState(s);
      for i in 1:size(init_cells, 1) loop
        if init_cells[i, 1] > 0 and init_cells[i, 2] > 0 then
          InitState(s,init_cells[i, 1],init_cells[i, 2]);
        end if;
      end for;
      init := true;
    end when;
  equation
    when sample(0, 1) then
      sumactive = Step(s);
      mean = if time > 0 then sumactive/(time+1) else sumactive;
    end when;
    annotation (
      Documentation(info="<html>
<p>This model represents a two-dimensional cellular space. </p>
</html>"),
      Diagram(coordinateSystem(
          preserveAspectRatio=false,
          extent={{-100,-100},{100,100}},
          initialScale=0.1),graphics),
      experiment(StopTime=100),
        Dymola_experimentSetupOutput,
      experimentSetupOutput);
  end GameOfLifeExt;

  function DefaultState
    input CS2D space;
    external "C" CS2D_InitDefault(space);
        annotation (Include="#include \"gameoflifeext.c\"");
  end DefaultState;

  function InitState
    input CS2D space;
    input Integer row;
    input Integer col;
    external"C" CS2D_Init(space,row,col);
    annotation (Include="#include \"gameoflifeext.c\"");
  end InitState;

  function Step
    input CS2D space;
    output Integer out;
  external"C" out = CS2D_Step(space);
    annotation (Include="#include \"gameoflifeext.c\"");
  end Step;
end ExternalGameOfLife;
```

Modelica Code 10.5: The Game of Life model using external functions (2/2).

C Code

```
/*****************************************************
* Author:
*          Victorino Sanz
*          Dpto. Informatica y Automatica, UNED
*          Juan del Rosal, 16
*          28040, Madrid
*          Spain
*          Email: vsanz@dia.uned.es
*
* Licensed by Victorino Sanz under the Modelica License 2
* Copyright 2013, Victorino Sanz.
*
*****************************************************/

#ifndef CELLULARAUTOMATA
#define CELLULARAUTOMATA

#ifdef LINUX
#include <unistd.h>
#endif

#include <stdio.h>
#include <stdlib.h>


/*****************************************************
  2D CELLULAR AUTOMATA
*****************************************************/


/*****************************************************
 Cell data type
 This data structure is used to represent the cells that are involved in the simulation.
 Each cell stores its state (dead or alive), its position in the space (row, col), if it
 is active or not in the current step (only active cells are evaluted in each step), an
 array of references to its neighbors (so they are easily accesible for the calculations)
 and the references to the previous and next cells in the same row (this does not mean
 that the cells are adjacent).
*****************************************************/
typedef struct Cell2D{
    int cellstate; // state of the cell
    int row; // row in 2D space
    int col; // column in 2D space
    int active; // boolean flag to set cell as active or not
    struct Cell2D **neighbors; // references to neighbors
    struct Cell2D *prev; // link to previous cell in the same row
    struct Cell2D *next; // link to next cell in the same row
} Cell2D;
```

```
/*********************************************************
   Cellular space structure
   This data structure represents the whole cellular space. Cells are arranged in rows, but
   each row contains only the cells that have been active any time during the simulation witout
   considering the whole size of the cellular space.
   The cellular space structure stores the matrix of cells (again, only the cells that have
   been involved in the simulation), the array of currently active cells (to facilitate the
   simulation of the current step), some parameters like the size of the space, the display
   delay for the graphical animation, the topology of the neighborhood, a boolean flag to
   generate the animation or not and the file descriptor to connect with gnuplot.
*********************************************************/

typedef struct CS2D{
    Cell2D **M; // matrix of cells in the space
    int *n_M; // number of cells in the space
    Cell2D **A; // array of active cells
    int n_A; // number of active cells
    int max_A; // maximum number of possible active cells
    int ncols; // number of columns of the space
    int nrows; // number of rows of the space
    int displayDelay; // delay to display animation in gnuplot
    int neighborhood[8][2];// topology of the neighborhood
    int n_neighbors; // number of neighbors
    int plot_animation; // boolean flag to display graphical animation or not
    FILE * gp; // gnuplot terminal file descriptor
}CS2D;

// function to compute de modulus of two integers
int mod (int a, int b){
    int ret;

    if(b < 0) //you can check for b == 0 separately and do what you want
        return mod(-a, -b);
    ret = a % b;
    if(ret < 0)
        ret+=b;
    return ret;
}
```

```
/*************************************************************
   CONSTRUCTOR FOR EXTERNAL OBJECT
   This function is used to construct the external object defined in Modelica.
   It basically allocates memory for the data structures and initializes the values
   of the parameters using the values received from Modelica as function prototype parameters
*************************************************************/
void* CS2D_Create(int nrows, int ncols, int plot_animation,int plot_range,int
displayDelay, const char *path,const char *name){
    CS2D *s;
    int i,j;

    s = (CS2D *)malloc(sizeof(CS2D));
    s->M = (Cell2D **)malloc(nrows*sizeof(Cell2D*));
    s->n_M = (int*)malloc(nrows*sizeof(int));
    for(i=0;i<nrows;i++){
        s->M[i] = NULL;
        s->n_M[i] = 0;
    }
    s->max_A = ncols;
    s->A = (Cell2D**)malloc(s->max_A*sizeof(Cell2D*));
    s->n_A = 0;
    s->ncols = ncols;
    s->nrows = nrows;
    // Moore's neighborhood is predefined
    s->neighborhood[0][0] = -1;
    s->neighborhood[0][1] = -1;
    s->neighborhood[1][0] = -1;
    s->neighborhood[1][1] = 0;
    s->neighborhood[2][0] = -1;
    s->neighborhood[2][1] = 1;
    s->neighborhood[3][0] = 0;
    s->neighborhood[3][1] = -1;
    s->neighborhood[4][0] = 0;
    s->neighborhood[4][1] = 1;
    s->neighborhood[5][0] = 1;
    s->neighborhood[5][1] = -1;
    s->neighborhood[6][0] = 1;
    s->neighborhood[6][1] = 0;
    s->neighborhood[7][0] = 1;
    s->neighborhood[7][1] = 1;
    s->n_neighbors = 8;
    s->plot_animation = plot_animation;
    s->displayDelay = displayDelay;
```

```
    // if animation is required, the buffer to gnuplot is configured
    if (plot_animation){
#ifdef _WIN32
        s->gp = _popen(path, "w");
#else
        s->gp = popen(path, "w");
#endif

        if(s->gp == NULL){
            fprintf(stderr, "Oops, I can't find %s.", path);
            exit(EXIT_FAILURE);
        }
   // setting parameters for gnuplot animation
#ifdef _WIN32
        fprintf(s->gp, "set terminal wxt persist \n");
#else
        fprintf(s->gp, "set terminal x11 \n");
#endif
        fprintf(s->gp, "set title \"%s\" \n",name);
        fprintf(s->gp, "set yrange [%d:0] \n",nrows);
        fprintf(s->gp, "set autoscale xy \n");
        fprintf(s->gp, "set cbrange [0:%d] \n",plot_range);
    }

    return (void *)s; // void pointer is returned as required by Modelica external objects.
}
```

```
/*********************************************************
   DESTRUCTOR FOR EXTERNAL OBJECT
   This function is used to delete the external object defined in Modelica. All previously
   allocated memory is freed and the file descriptor used to connect with gnuplot is closed.
*********************************************************/

int CS2D_Delete(void* space){
    CS2D *s;
    Cell2D *cell;
    int i,j;

    s =(CS2D *)space;
    cell = s->M[0];
    for(i=0;i<s->nrows;i++){
        for(j=0;j<s->n_M[i];j++){
            cell = s->M[i];
            s->M[i] = cell->next;
            free(cell);
        }
    }
    free(s->M);
    free(s->n_M);
    free(s->A);
    s->ncols = 0;
    s->nrows = 0;
    if(s->plot_animation){
#ifdef _WIN32
        fprintf(s->gp, "pause mouse\n");
        _pclose(s->gp);
#else
        pclose(s->gp);
#endif
    }
    free(s);
    return 1;
}
```

```
/*******************************************************
   FIND CELL
   This function is used to find a given cell in the cellular space structure. A pointer to
   the cell is returned or NULL in case it is not found in the space. The position of the
   cell to be found is defined by the row and col parameters. The ref parameter indicates a
   reference to use as starting point for the search (i.e., when searching a cell closely
   located from another).
*******************************************************/

Cell2D* CS2D_FindCell(void* space, int row, int col, Cell2D *ref){
    CS2D *s;
    Cell2D *cell;
    int found,i;
    Cell2D *pre;


    s = (CS2D *)space;
    cell = NULL;
    found = 0;
    if (s->M[row] == NULL) // no cells in row
        return NULL;
    if (ref == NULL){ // no reference, start searching from the beginning of the row
        cell = s->M[row];
        while(cell != NULL && !found){
            if(cell->col == col){ // same columns, cell found!
                found = 1;
            }else{
                cell = cell->next; // not found, go to next
           }
        }
    }else{ // start searching from given reference cell
        // search in the neighborhood of the reference
        i = 0;
        if (ref->row == row){ // search from reference
            cell = ref;
        }else{
            while (i<s->n_neighbors){   // search from neighbor
                if(ref->neighbors[i] != NULL) // neighbor present
                    if (ref->neighbors[i]->row == row){
                        cell = ref->neighbors[i];
                        i = s->n_neighbors;
                    }
                i++;
            }
            if(cell == NULL) // search from the beginning of the row
                cell = s->M[row];
        }
```

```
        pre = cell->prev; // pre: used to search backwards in the row.
        while ((cell != NULL || pre != NULL) && !found){
            if (cell != NULL && cell->col == col){
                found = 1;
            }else if (cell != NULL)
                cell = cell->next;
            if (pre != NULL && pre->col == col){
                found = 2;
            }else if (pre != NULL)
                pre = pre->prev;
        }
    }

    if(found == 1){
        return cell;
    }else if (found == 2){
        return pre;
    }else{
        return NULL;
    }
}
```

```
/*********************************************************
    INSERT CELL
    This function is used to insert a new cell in the cellular space (i.e., a cell that has
    never been previously active).
    The cellular space, the new cell and its position in the space are passed as parameters.
*********************************************************/
void CS2D_InsertCell(void* space, Cell2D* cell, int row, int col){
    CS2D *s;
    Cell2D *n; // next cell
    Cell2D *p; // previous cell

    s = (CS2D *)space;
    n = s->M[row];
    p = s->M[row];
    s->n_M[row]++;
    if (s->M[row] == NULL){ // insert as first cell in the space
        cell->prev = NULL;
        cell->next = NULL;
        s->M[row] = cell;
    }else{
        if (s->M[row]->col > col){ // insert before first cell in space
            cell->prev = NULL;
            cell->next = s->M[row];
            s->M[row]->prev = cell;
            s->M[row] = cell;
        }else{ // find place to insert cell
            n = s->M[row]->next;
            p = s->M[row];
            while(n != NULL && n->col < col){
                p = n;
                n = n->next;
            }
            // place found, insert cell
            cell->prev = p;
            cell->next = n;
            if(n!=NULL)
                 n->prev = cell;
            p->next = cell;
        }
    }
    return;
}
```

```
/*********************************************************
    ACTIVE CELL
    This function is used to set a cell as active, in order to be evaluated during the next
    simulation step.
    Parameters are the cellular space and the cell to be active.
*********************************************************/
Cell2D* CS2D_ActiveCell(void *space, Cell2D* cell){
    int i,k;
    CS2D *s;

    s = (CS2D *)space;
    if(!cell->active){
        cell->active=1; // set active flag to 1
        s->n_A++;  // increase number of active cells
        if(s->n_A > s->max_A){ // if maximum number of active cells is reached, increase
allocated memory for s->A
            s->max_A = s->max_A*2; // double maximum number of active cells
            s->A =  (Cell2D **)realloc(s->A,s->max_A*sizeof(Cell2D*)); // reallocate
the list of currently active cells
        }s->A[s->n_A-1] = cell; // insert new active cell in the list
    }
    return cell;
}


/*********************************************************
    INIT CELL
    This function is used to set the initial state for cells before starting the simulation
    steps.
    Parameters are the cellular space and the position of the cell to be initalized (since Modelica
start arrays at index 1, modrow and modcol need to be translated to real rows and cols).
*********************************************************/
Cell2D* CS2D_Init(void* space,int modrow, int modcol){
    int i,k,row,col;
    CS2D *s;
    Cell2D *newcell;

    row = modrow-1; // C index corresponds to Modelica index -1
    col = modcol-1;
    s = (CS2D *)space;
    newcell = CS2D_FindCell(space,row,col,NULL); // find cell in the space
    newcell->cellstate = 1; // set state to 1 (alive)
    CS2D_ActiveCell(space,newcell); // set cell as initially active

    return newcell;
}
```

```
/*******************************************************
    INIT DEFAULT
    This function is used to create and set the default initial state for all the cells in the
    cellular space (i.e., default state is to be dead). Pointers to neighbors are also set to
    faclitate the computations during each simulation step.
*******************************************************/
void CS2D_InitDefault(void *space){
    int i,j,k;
    int rows,cols;
    CS2D *s;
    Cell2D *newcell;
    int nrow,ncol,en;

    s = (CS2D *)space;
    rows = s->nrows;
    cols = s->ncols;

    for(i=0;i<rows;i++){
        for(j=0;j<cols;j++){
            // create new cell to insert into cellular space
            newcell = (Cell2D *)malloc(sizeof(Cell2D));
            newcell->cellstate = 0; // default state
            newcell->row = i;
            newcell->col = j;
            newcell->prev = NULL;
            newcell->next = NULL;
            newcell->active = 0;
            //neighbor.update = 1;
            newcell->neighbors =    (Cell2D **)malloc(s->n_neighbors*sizeof(Cell2D
*)); // array of pointers to neighbors
            for(k=0;k<s->n_neighbors;k++)
                newcell->neighbors[k] = NULL;
            CS2D_InsertCell(space,newcell,i,j); // insert new cell into space
        }
    }
    // once all the cells are created, assign neighbors
    for(i=0;i<rows;i++){
        for(j=0;j<cols;j++){
            newcell = CS2D_FindCell(space,i,j,NULL);
            for(k=0;k<s->n_neighbors;k++){
                en = s->neighborhood[k][0];
                nrow = mod(i+s->neighborhood[k][0],s->nrows);
                en = s->neighborhood[k][1];
                ncol = mod(j+s->neighborhood[k][1],s->ncols);
                newcell->neighbors[k] = CS2D_FindCell(space,nrow,ncol,newcell);
            }
        }
    }
    return;
}
```

```
/*********************************************************
   GAME OF LIFE FUNCTION
   This function is used to represent the behavior of the cells as described in the Game of
   Life. This function is individually applied to every active cell in each simulation step.
   Function parameters are the change flag, to be set if the state of the cell changes from
   its previous value (and so it remains active, or otherwise is set inactive), the current
   state of the cell, the array of neighbors and the number of neighbors).
*********************************************************/
int gol(int* change, int cellstate, int* neighbors, int n_neighbors){
    int sum,i;
    int out;

    // sum number of living neighbors
    sum = 0;
    for(i=0;i<n_neighbors;i++){
        if(neighbors[i] > 0){
            sum++;;
        }
    }

    // apply rules
    out = cellstate;
    if (!cellstate && sum == 3) // dead and 3 living neighbors
        out = 1; // becomes alive
    else if (cellstate && (sum < 2 || sum > 3 ) ) // alive and less than 2 or more than 3
living neighbors
        out = 0; // dies

    *change = 1;
    if(cellstate == out)
    *change = 0;

    return out;
}
```

```
/*************************************************
    STEP
    This function is used to perform a simulation step.
*************************************************/
int CS2D_Step(void* space){
     int *new_states;
     int news;
     int i,k,nA;
     CS2D *s;
     int neighbors[8];
     Cell2D** new_A; // new active cells after current step
     int new_n_A = 0; // number of new active cells
     Cell2D* cell;
     //cell_list* lchange;
     int change;
     int active;

     //struct timeval   tv1, tv2;
     //gettimeofday(&tv1, NULL);



     s = (CS2D *)space;

     // Set neighbors of current active cell also active.
     nA = s->n_A;
     for(i=0;i<nA;i++){
          for(k=0;k<s->n_neighbors;k++){
               if(s->A[i]->neighbors[k] != NULL){
                    CS2D_ActiveCell(space,s->A[i]->neighbors[k]);
               }
          }
     }


     // calculate new cellstates
     new_states = (int*)malloc(sizeof(int)*s->n_A);
     //max_new_A = s->max_A;
     new_A = (Cell2D**)malloc(s->max_A*sizeof(Cell2D*));
     new_n_A = 0;
     active = s->n_A;  // store in active var the number active cells in this step
     for(i=0;i<s->n_A;i++){
          // join neighbors states in a single array
          for(k=0;k<s->n_neighbors;k++){
               neighbors[k] = s->A[i]->neighbors[k]->cellstate;
          }
          cell = s->A[i]; // cell is current active cell to be evaluated
          // calculate new state using user defined rule function
          news = gol(&change, cell->cellstate,neighbors,s->n_neighbors); // game of life
function
          new_states[i] = news; //store new state of current cell
```

```
        // keep cell active if state changed
        if (change){
            cell->active=1;
            new_n_A++;
            if (new_n_A > s->max_A){
                s->max_A = s->max_A*2;
                new_A = (Cell2D **)realloc(new_A,s->max_A*sizeof(Cell2D*));
            }
            new_A[new_n_A-1] = cell;
        }else // otherwise, set as inactive
            cell->active = 0;
    }

    // update cellstates after evaluating all active cells
    for(i=0;i<s->n_A;i++){
        cell = s->A[i];
        if(cell != NULL && new_states[i] != -1){
            cell->cellstate = new_states[i];
            new_states[i] = -1;
        }
    }

    // plot animation for new states
    if(s->plot_animation){
        fprintf(s->gp,"plot \'-\' matrix with image\n");
        for (i=0;i<s->nrows;i++){
            cell = s->M[i];
            for(k=0;k<s->ncols;k++){
                if (cell != NULL && k == cell->col){ // cell in space, print state
                    fprintf(s->gp,"%f ",(double)cell->cellstate);
                    cell = cell->next;
                }else{// cell not present
                    fprintf(s->gp,"-1 ");
                }
            }
            fprintf(s->gp,"\n ");
        }
    }

    if(s->plot_animation){
        // fprintf(s->gp,"\n");
        fprintf(s->gp,"\n");
        fprintf(s->gp,"e\n");
        //printf("\n");
        //printf("\n");
        //printf("e\n");
#ifdef _WIN32
        //Sleep(s->displayDelay/1000);
#else
        usleep(s->displayDelay);
#endif
        fflush(s->gp);
    }
```

```
    free(s->A);
    s->A = new_A;
    s->n_A = new_n_A;
    free(new_states);
    return active;
}

#endif
```

10.7 Solution to Task 5

The objective is to compare the performance of the models developed in Tasks 1 and 4, in terms of size of the cellular space and execution time.

Equivalent simulation runs have been performed using the model developed in Task 4. The initial configuration of the cells correspond to the glider shown in Figure 10.1.

In this case, since the number of cells in the simulation is managed using dynamic memory the size of the compiled model is 13 kb independently of the size of the cellular space. Also, since the initial configuration of the model (i.e. the glider) produces a periodic variation of the cells in the space the number of active cells is very similar in each step and the simulation time is equal for all the experiments performed. Independently on the size of the cellular space, the simulation takes around 0.0022 s to execute.

Assignment 11

Air pollution

Purpose of this assignment

- Get insight into the Gaussian plume model.
- Use arrays, for loops, and enumeration types.
- Describe a DAE system as an atomic model in Modelica.

11.1 System description

The pollutants emitted through a stack source enter the atmosphere and can be transported or dispersed by meteorological conditions. The pollutant dynamics is influenced by several phenomena, including transport, diffusion, chemical transformation, and ground deposition.

An industrial chimney (i.e. stack) emitting pollutants into the landscape is shown in Figure 11.1. The polluted air coming from the stack is called plume. The plume shape can be influenced by the meteorological conditions and emission factors such as gas temperature at the stack top, stack height, pollutant mass flow rate, or wind velocity.

The Gaussian plume model is a standard approach for studying the transport of atmospheric pollutants (Abdel-Rahman 2008). The main assumptions made in this model are the following (Brusca et al. 2016):

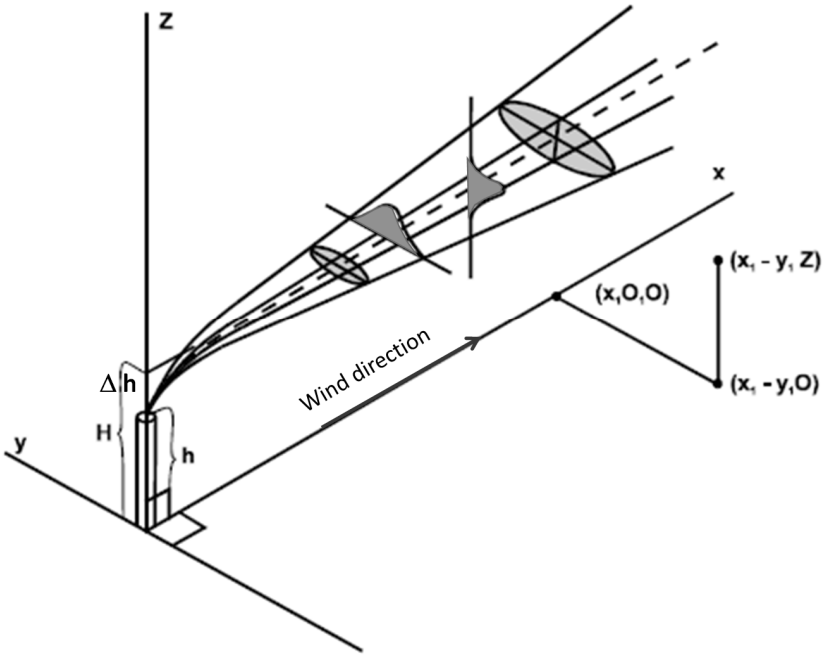


Figure 11.1: Gaussian plume model.

- Steady-state conditions, which implies a constant emission rate of pollutants, and constant wind speed, wind direction, temperature and mixing height.
- Dispersion is negligible in the downwind direction.
- No wind sheer in the horizontal or vertical plane.
- The pollutants are non-reactive gases or aerosol.
- There is a perfect reflection of the plume at the underlying surface, with no deposition or reaction with the surface.

Pollutant distribution is represented by concentrations on a (regular) three-dimensional grid of points. A Cartesian reference system is employed, with its origin at ground level, at the chimney location. The x axis is parallel to the wind direction and the z axis corresponds to the height measured from the ground level.

The Gaussian method employs the Gaussian normal distribution to determine the variation of the pollutant concentrations in the plume by relatively simple calculations. This model has a symmetry axis determined by the wind direction, as is illustrated in Figure 11.1. The pollutant concentration in the air, at the (x, y, z) position, is described by Eq. (11.1).

$$C(x,y,z) = \frac{Q}{2 \cdot \pi \cdot u \cdot \sigma_y \cdot \sigma_z} \cdot exp\left(-\frac{y^2}{2 \cdot \sigma_y^2}\right) \cdot \left\{exp\left[-\frac{1}{2}\left(\frac{z-H}{\sigma_z}\right)^2\right] + exp\left[-\frac{1}{2}\left(\frac{z+H}{\sigma_z}\right)^2\right]\right\} \tag{11.1}$$

where $C(x,y,z)$ is the concentration of pollutant in the air in g/m^3, Q is the emission rate of the pollutant from the source in g/s, σ_y is the horizontal crosswind dispersion coefficient, σ_z is the vertical dispersion coefficient, u is the mean speed of the wind in m/s, and H is the effective height of the plume in m. H is computed by adding the additional height to which the hot gases rise above the stack (Δh) to the physical height of the stack (h).

The following equation has to be applied to compute the concentration of a particular component in parts per million (ppm):

$$C_{component} = \frac{C \cdot 10^6 \cdot K}{MW} \tag{11.2}$$

where $C_{component}$ is the component concentration in ppm, C is the concentration on pollutant in the air in g/m^3, K is a constant equal to 24.5 at 1 atm and 25° C, and MW is the molecular weight of the component expressed in g/mol. The CO molecular weight is 28 g/mol, and the SO_2 molecular weight is 64 g/mol.

The values of the horizontal and vertical dispersion coefficients (σ_y and σ_z) depend on the distance to the source and the atmospheric conditions. They can be calculated by applying different theories: Martin, Briggs, Pasquill, Taylor, etc. We will calculate these coefficients as described below (Martin 1976).

Pasquill defined the following seven classes of atmospheric stability conditions: A, B, C, D, E and F. The parameters correspond to "very stable" (F class) to "very unstable" (A class). The neutral atmosphere is represented by D. The equations proposed in (Martin 1976) to compute the dispersion coefficients are the following:

$$\sigma_y = a \cdot x^b \tag{11.3}$$

$$\sigma_z = c \cdot x^d + f \tag{11.4}$$

where x is the distance to the source expressed in km; a, c, d and f are stability-dependent coefficients; and b is a constant of value 0.894. The values of the c, d and f coefficients depend on whether x is larger than one kilometer. The values of these constants are given in Table 11.1.

Table 11.1: Values of the a, c, d and f coefficients.

Stability class	a	c		d		f	
		$x \leq 1km$	$x > 1km$	$x \leq 1km$	$x > 1km$	$x \leq 1km$	$x > 1km$
A	213	440.8	459.7	1.941	2.094	9.27	-9.6
B	156	106.6	108.2	1.149	1.98	3.3	2.0
C	104	61	61	0.911	0.911	0	0
D	68	33.2	44.5	0.725	0.516	-1.7	-13.0
E	50.5	22.8	55.4	0.678	0.305	-1.3	-34.0
F	34	14.35	62.6	0.74	0.18	-0.35	-48.6

Table 11.2: Values of the p coefficient.

Stability class	p in rural area	p in urban area
A, B	0.07	0.15
C	0.10	0.20
D	0.15	0.25
E	0.35	0.40
F	0.55	0.60

We suppose that the wind speed at a height of 10 m (u_{10}) is known. The wind speed at the stack mouth is estimated from Eq. (11.5), where p is a coefficient that depends on the stability class and the surface roughness (rural or urban area) as shown in Table 11.2.

$$u = u_{10} \cdot \left(\frac{h}{10}\right)^{p} \tag{11.5}$$

11.2 Task

Suppose a stack that emits CO and SO_2. Write a Modelica model to compute the concentration of these air pollutants at the points of a rectangular mesh of size $10 \times 11 \times 11$. Define the step size in the x, y and z dimensions as 100, 100 and 2 m, respectively. Store the values of the a, c, d, f and p coefficients in constant matrices. Describe the stability class as an enumeration type. The model has the following parameter values: $u_{10} = 2$ m/s, $H = 30$ m, $Q = 80 \cdot 10^{-3}$ kg/s, urban medium, and stability class A.

11.3 Solution

The model can be described in Modelica as shown in Modelica Codes 11.1 – 11.3. Observe that the model quantities are expressed in the International System of Units. The development in Modelica of a virtual lab based on this model is described in (Martin-Villalba et al. 2018).

```
model PollutantsDispersion
  import SI = Modelica.SIunits;

  parameter Integer Nx = 10;  // deltaX, 2*deltaX, ..., Nx*deltaX
  parameter Integer Ny = 11;  // 0, deltaY, 2*deltaY, ..., (Ny-1)*deltaY
  parameter Integer Nz = 11;  // 0, deltaZ, 2*deltaZ, ..., (Nz-1)*deltaZ

  // Step size in axis X, Y and Z
  parameter SI.Distance deltaX = 100;
  parameter SI.Distance deltaY = 100;
  parameter SI.Distance deltaZ = 2;

  // Spatial coordinates
  parameter SI.Distance x[Nx]=  deltaX:  deltaX:  Nx*deltaX;
  parameter SI.Distance y[Ny]=  0:  deltaY:  (Ny-1)*deltaY;
  parameter SI.Distance z[Nz]=  0:  deltaZ:  (Nz-1)*deltaZ;

  // Pollutant concentrations
  SI.MassConcentration C[Nx,Ny,Nz];
  Real C_CO[Nx,Ny,Nz];
  Real C_SO2[Nx,Ny,Nz];

  constant Real MW_CO = 28e-3;      // CO molecular weight in kg/mol
  constant Real MW_SO2 = 64e-3;     // SO2 molecular weight in kg/mol
  constant Real K = 24.5; // Constant value at 1 atm and 25 Celsius degrees

  //Wind speed at the stack mouth
  SI.Velocity u;

  // Dispersion coefficients
  SI.Distance sigmaY[Nx];
  SI.Distance sigmaZ[Nx];

  // Wind speed at 10 m height
  parameter SI.Velocity u10 = 2;

  // Emission rate
  parameter SI.MassFlowRate Q = 80e-3;

  // Effective plume height (stack height + plume rise)
  parameter SI.Distance H = 30;
```

Modelica Code 11.1: Gaussian dispersion model (1/3).

```
// Stability (1=A, 2=B, ..., 6=F)
type stabilityClass = enumeration(
    catEstab_A,
    catEstab_B,
    catEstab_C,
    catEstab_D,
    catEstab_E,
    catEstab_F);

parameter stabilityClass stability = stabilityClass.catEstab_A;

// Medium (1=urban, 2=rural)
type mediumType = enumeration(
    urban,
    rural);

parameter mediumType medium = mediumType.urban;

// Parameters a, b, c, d, f
// First dimension: stability (1=A, 2=B, ..., 6=F)
// Second dimension: (x < 1000 m) = 1, (x > = 1000 m) = 2

constant Real b = 0.894;
constant Real a[6] = { 213, 156, 104, 68, 50.5, 34};
constant Real c[6,2] =  {  { 440.8, 459.7},
                           { 106.6, 108.2},
                           { 61.0, 61.0},
                           { 33.2, 44.5},
                           { 22.8, 55.4},
                           { 14.35, 62.6}};
constant Real d[6,2] =  {  { 1.941, 2.094},
                           { 1.149, 1.098},
                           { 0.911, 0.911},
                           { 0.725, 0.516},
                           { 0.678, 0.305},
                           { 0.740, 0.180}};
constant Real f[6,2] =  {  { 9.27, -9.6},
                           { 3.3, 2.0},
                           { 0, 0},
                           { -1.7, -13},
                           { -1.3, -34},
                           { -0.35, -48.6}};

// Exponential coefficient of the speed
// First dimension: stability (1=A, 2=B, ..., 6=F)
// Second dimension: medium (1=urban, 2=rural)

constant Real p[6,2] = {   { 0.15, 0.07},
                           { 0.15, 0.07},
                           { 0.20, 0.10},
                           { 0.25, 0.15},
                           { 0.40, 0.35},
                           { 0.60, 0.55}};
```

Modelica Code 11.2: Gaussian dispersion model (2/3).

```
equation

  // Speed wind at stack mouth
  u = (H/10)^p[stability,medium];

  // Pollutant concentration
  for ix in 1:Nx loop
    for iy in 1:Ny loop
      for iz in 1:Nz loop
        C[ix,iy,iz] = Q/(2*3.14159265358979*u*sigmaY[ix]*sigmaZ[ix])*
                      exp(-0.5*(y[iy]/sigmaY[ix])^2)*
                      ( exp(-0.5*((z[iz]-H)/sigmaZ[ix])^2) +
                        exp(-0.5*((z[iz]+H)/sigmaZ[ix])^2));
        C_CO[ix,iy,iz] = C[ix,iy,iz]*1e+6*K/MW_CO;
        C_SO2[ix,iy,iz] = C[ix,iy,iz]*1e+6*K/MW_SO2;
      end for;
    end for;
  end for;

  // Dispersion coefficients
  for ix in 1:Nx loop
     sigmaY[ix] = a[stability]*(x[ix]/1000)^b;
     sigmaZ[ix] = if x[ix] <= 1000 then
                      c[stability,1]*(x[ix]/1000)^d[stability,1]+f[stability,1] else
                      c[stability,2]*(x[ix]/1000)^d[stability,2]+f[stability,2];
  end for;

  annotation (uses(Modelica(version="3.2.2")), DymolaStoredErrors,
    version="1",
    conversion(noneFromVersion=""));
end PollutantsDispersion;
```

Modelica Code 11.3: Gaussian dispersion model (3/3).

Assignment 12

Simplified Tennessee Eastman model

Purpose of this assignment

- Practice modeling of hydraulic components and simple chemical reactors.
- Use records to describe fluid properties.
- Use arrays of variables and for loops.
- Embed a Modelica model into a Simulink block using FMI.

12.1 System description

The Tennessee Eastman Process model is based on a real chemical process that was first described in (Downs & Vogel 1993). It contains a separator/reactor/recycle arrangement involving two simultaneous gas-liquid exothermic reactions.

The Simplified Tennessee Eastman (TES) model (Ricker 1993) here described is a simplification of the model described in (Downs & Vogel 1993), with only one process unit and eight state variables. This process unit is a combination of a reactor and a separator of the original TE process.

The TES model is also a well-known benchmark process. It is a multi-input multi-output, nonlinear system, open-loop unstable, and contains fast and slow dynamics.

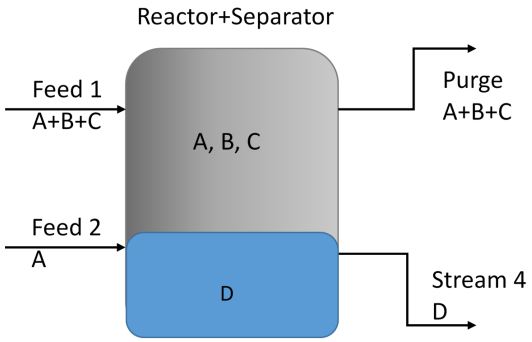


Figure 12.1: Schematic diagram of the TES process.

The process unit has two input components (A and C) and an inert gas (B). An irreversible reaction takes place in the reactor in vapor phase, generating the product D in liquid phase. The reaction is described by the following equation:

$$A + C \longrightarrow D \tag{12.1}$$

The process unit (see Figure 12.1) has two input flows (Feed 1 and Feed 2) and two output flows (Purge and Stream 4). Feed 1 contains A, C, and trace amounts of an inert gas B. Feed 2 contains only A. The purge (i.e., Stream 3) contains A, B and C. The output flow contains the desired product D.

The following assumptions are made (Ricker 1993):

- A and C are non-condensable, and D is a non-volatile liquid.
- The vapor phase consists of A, B, and C. The liquid is pure D. This is because the solubilities of A, B, and C in D are negligible. D has a density (ρ_D) equal to 8.3 kmol/m^3. The gases follow the ideal gas law:

  $$P \cdot V = n \cdot R \cdot T \tag{12.2}$$

  where P is the gas pressure, V is the gas volume, n is the number of gas moles, R is the ideal gas constant ($R = 8.31451$ J/(mol·K)) and T is the gas temperature.
- The reaction rate (R_D) depends only on the partial pressures of A (P_A) and C (P_C). Independent controls (not shown) maintain isothermal operation. The temperature in the reactor (T_r) is then a known model parameter. R_D in $kmol/h$ is obtained from the following equation

$$R_D = k_0 \cdot P_A^{0.5} \cdot P_C^{0.4} \tag{12.3}$$

where $k_0 = 0.00117$ is a constant for the assumed isothermal operation.

– The molar flow rates (units: kmol/h) of streams 1 and 2 are linear functions of valve position:

$$F_i = F_{i,max} \cdot \frac{X_i}{100} \qquad \text{for} \quad i = 1, 2 \tag{12.4}$$

where $0 \leq X_i \leq 100$ is the i-th valve position (percentage open), $F_{1,max} = 330.46$ kmol/h and $F_{2,max} = 22.46$ kmol/h.

– The molar flow rates (units: kmol/h) of streams 3 and 4 are nonlinear functions of the valve position (X_i) and the system pressure (P):

$$F_i = \frac{X_i}{100} \cdot C_{vi} \cdot \sqrt{P - 100} \qquad \text{for} \quad i = 3, 4 \tag{12.5}$$

where $0 \leq X_i \leq 100$ is the i-th valve position (percentage open), $C_{v3} = 0.00352$ and $C_{v4} = 0.0417$.

– A valve position (X_i) responds gradually to a change in its command signal u_i as follows:

$$\tau_V \cdot \frac{dX_i}{dt} = u_i - X_i \qquad \text{for} \quad i = 1, ..., 4 \tag{12.6}$$

where $\tau_V = 10/3600$ (units: h) is a valve time constant, identical for all four valves.

12.2 Task 1

Write the mathematical equations describing the process unit, which is composed of the reactor and separator.

The liquid density of component D (ρ_D) is equal to 8.3 kmol/m^3. The volume of the process unit (V_r) is a model parameter of value 122 m^3. The temperature of the reactor (T_r) is a model parameter of value 373 K. The values of the initial molar holdups of components A, B, C and D are listed in Table 12.1.

Table 12.1: Nominal operating conditions.

Symbol	Description	Nominal value	Units
N_A	Molar holdup of A	44.499999	kmol
N_B	Molar holdup of B	13.532996	kmol
N_C	Molar holdup of C	36.64788	kmol
N_D	Molar holdup of D	110.0	kmol
X_1	Feed 1 valve position	60.953273	%
X_2	Feed 2 valve position	25.0223	%
X_3	Purge valve position	39.2577	%
X_4	Product valve position	44.030	%
u_1	Manip. var. corresponding to valve 1	60.953273	%
u_2	Manip. var. corresponding to valve 2	25.0223	%
u_3	Manip. var. corresponding to purge	39.2577	%
u_4	Manip. var. corresponding to product valve	44.1767	%
$F_1.conc[1]$	Mole fraction A in feed 1	0.485	–
$F_1.conc[2]$	Mole fraction B in feed 1	0.005	–

12.3 Solution to Task 1

The conservation law of the chemical species gives the following four equations:

$$\frac{dN_A}{dt} = F_1.conc_A \cdot F_1 + F_2 - F_3.conc_A \cdot F_3 - R_D \tag{12.7}$$

$$\frac{dN_B}{dt} = F_1.conc_B \cdot F_1 - F_3.conc_B \cdot F_3 \tag{12.8}$$

$$\frac{dN_C}{dt} = F_1.conc_C \cdot F_1 - F_3.conc_C \cdot F_3 - R_D \tag{12.9}$$

$$\frac{dN_D}{dt} = R_D - F_4 \tag{12.10}$$

where N_i, with $i = A, B, C, D$, is the molar hold up in kmol of component i in the vessel; F_j, with $j = 1, 2, 3, 4$, is the molar flow in kmol/h in stream j; $F_j.conc_i$ is the concentration of component i in the stream j; and R_D is the production rate of component D.

The partial pressure of components A, B and C is obtained from the ideal gas equation:

$$p_i = \frac{N_i \cdot R_i \cdot T_r}{V_{vap}} \tag{12.11}$$

where p_i, $i = A, B, C$, is the partial pressure of component i, N_i is the molar hold up of component i, T_r is the reactor temperature (a model parameter) and V_{vap} is the vapor volume in the vessel.

The liquid volume (V_{liq}) is computed taking into account that D is the only liquid with a density ρ_D (model parameter):

$$V_{liq} = \frac{N_D}{\rho_D} \tag{12.12}$$

The vapor volume (V_{vap}) is obtained from the following equation:

$$V_r = V_{liq} + V_{vap} \tag{12.13}$$

where V_r is the vessel volume (model parameter).

The total pressure (P) is obtained from the ideal gas equation, assuming equilibrium between the gas and liquid phase:

$$P = \frac{(N_A + N_B + N_C) \cdot R \cdot T_r}{V_{vap}} \tag{12.14}$$

The molar fraction in the purge ($F_3.conc_i$, $i = A, B, C$) is obtained from the following equation:

$$F_3.conc_i = \frac{p_i}{P} \tag{12.15}$$

The reaction rate (R_D) will be zero if p_A and p_C are less or equal to zero. If it is not the case, R_D is computed from the following equation:

$$R_D = k_0 \cdot p_A^{1.2} \cdot p_C^{0.4} \tag{12.16}$$

where k_0 is a constant equal to 0.00117.

12.4 Task 2

Program a library in Modelica to describe the TES model. The values of the variables at the nominal operating conditions are listed in Table 12.1. The library has to include the following classes:

- A record describing the fluid properties.
- A connector class.
- A class describing the input valve.
- A class describing the output valve.
- A class describing the process unit.
- A class describing the complete model. Name it `ReactorOpenLoop`. This model will be exported as an FMU and has to be described accordingly. This is, the causality of the model has to be set explicitly, declaring as input/output the input/output variables. This model can then be embedded within a Simulink block.

12.5 Solution to Task 2

The Modelica library is shown in Modelica Codes 12.1–12.8.

```
package TESimplified "The Simplified Tennessee Eastman model"

record Media "Media components"
   constant Integer A = 1;
   constant Integer B = 2;
   constant Integer C = 3;
   constant Integer D = 4;
   constant Integer Vapor[:] = {1,2,3};
   constant Integer Liquid[:] = {4};
   constant Integer All[:] = 1:4;
   constant Integer NComp = size(All,1);
   constant Integer NVap = size(Vapor,1);
   constant Integer NLiq = size(Liquid,1);
end Media;
```

Modelica Code 12.1: Simplified Tennessee Eastman model (1/8).

```
model Reactor "Reactor-separator"
  Media Component;
  constant Integer NumChemReac = 1;
  // Constant of perfect gases
  constant Real Rkcal(unit="cal/(mol.K)") = 1.987;
  constant Real RkJ(unit="J/(mol.K)") = 8.31451;
  parameter Real Vr(unit="m3") = 122 "Reactor volume";
  parameter Real v[Component.NComp,NumChemReac] =
                [-1;  0;  -1;  1] "Stoichiometric coefficients";
  parameter Real rho[Component.NComp]( unit="kmol/m3") =
                {0,0,0,8.3} "Density of liquids";
  parameter Real k0 = 0.00117 "Constant for the isothermal operation";
  parameter Real Tr(unit="K") = 373 "Reactor Fixed Temperature";
  Real N[Component.NComp](unit="kmol",
         start={44.499999,13.5329,36.64788,110},
         fixed=true) "Total molar holdup";
  Real p[Component.NVap]( unit="kPa") "Partial pressure";
  Real P( unit="kPa") "Total pressure";
  Real R[NumChemReac]( unit="kmol/hour") "Reaction rate";
  Real Vliq(unit="m3") "Liquid volume";
  Real Vvap(unit="m3") "Vapor volume";

  pCon F1(nComp = Component.NComp)
    annotation (Placement(
      transformation(extent={{-79.5,21.9},{-59.5,41.9}})));

  pCon F2(nComp = Component.NComp)
    annotation (Placement(
      transformation(extent={{-79.7,-10.7},{-59.7,9.3}})));

  pCon F3(nComp = Component.NComp)
    annotation (Placement(
      transformation(extent={{60,22},{80,42}})));

  pCon F4(nComp = Component.NComp)
    annotation (Placement(
      transformation(extent={{60.3,-35.2},{80.3,-15.2}})));
```

Modelica Code 12.2: Simplified Tennessee Eastman model (2/8).

```
equation
  // Molar balance
  for i in Component.All loop
    der(N[i]) = F1.conc[i]*F1.flowM + F2.conc[i]*F2.flowM +
                F3.conc[i]*F3.flowM + F4.conc[i]*F4.flowM +
                sum(v[i, j]*R[j] for j in 1:NumChemReac);
  end for;
  // Total pressure assuming equilibrium vapor and liquid phase
  P = sum(N[i] for i in Component.Vapor) * RkJ * Tr / Vvap;
  // Partial pressure
  for i in Component.Vapor loop
    p[i] = N[i] * RkJ * Tr / Vvap;
  end for;
  // Liquid volume
  Vliq = sum(N[i] / rho[i] for i in Component.Liquid);
  // Vapor volume
  Vr = Vliq + Vvap;
  // Molar fraction of vapor
  for i in Component.Vapor loop
    F3.conc[i] = p[i] / P;
  end for;
  F3.conc[4] = 0;
  // Reaction rate
  R[1] = if noEvent(p[Component.A] > 0 and p[Component.C] > 0)
         then k0 * p[Component.A]^1.2 * p[Component.C]^0.4
         else 0;
  F1.P = P;
  F2.P = P;
  F3.P = P;
  F4.P = P;
  F4.conc = {0, 0, 0,1};
  if P > 3000 then
    terminate("High reactor pressure!!!");
  end if;
```

Modelica Code 12.3: Simplified Tennessee Eastman model (3/8).

```
  annotation (
    Diagram(coordinateSystem(preserveAspectRatio=false,
            extent={{-100,-100},{100,100}}), graphics),
    Icon(graphics={
          Line(
            points={{-60,50},{-58,60},{-54,66},{-48,70},
                    {-44,72},{-40,72}},
            color={0,0,255}, smooth=Smooth.None),
          Line(
            points={{-60,50},{-60,-30}},
            color={0,0,255}, smooth=Smooth.None),
          Line(
            points={{-10,-11},{-8,-1},{-4,5},{2,9},
                    {6,11},{10,11}},
            color={0,0,255}, smooth=Smooth.None,
            origin={-49.1,-40.3}, rotation=90),
          Line(
            points={{60.2,50.1},{58.2,60.1},{54.2,66.1},
                    {48.2,70.1},{44.2,72.1},{40.2,72.1}},
            color={0,0,255}, smooth=Smooth.None),
          Line(
            points={{59.8,-27.9},{57.8,-37.9},{54,-44},
                    {47.8,-47.9},{43.8,-49.9},{39.8,-49.9}},
            color={0,0,255}, smooth=Smooth.None),
          Line(
            points={{-40,72},{40,72}},
            color={0,0,255}, smooth=Smooth.None),
          Line(
            points={{-40,-50},{40,-50}},
            color={0,0,255}, smooth=Smooth.None),
          Line(
            points={{60.1,51.9},{60.1,-28.1}},
            color={0,0,255}, smooth=Smooth.None),
          Line(
            points={{-60,-4.1},{60.2,-4.1}},
            color={0,0,255}, smooth=Smooth.None),
          Text(
            extent={{-19.8,47.7},{21.3,28.4}},
            lineColor={0,0,255}, textString="Vapor"),
          Text(
            extent={{-18.7,-25},{22.4,-44.3}},
            lineColor={0,0,255}, textString="Liquid") } ) );
end Reactor;
```

Modelica Code 12.4: Simplified Tennessee Eastman model (4/8).

```
connector pCon
   parameter Integer nComp = 1 "Number of components";
   Real conc[ nComp](unit="1") "Concentration of each component";
   Real P( unit = "kPa") "Pressure";
   // Output Flow negative - Input flow positive
   flow Real flowM(unit="mol.t-1") "Molar flow of each component";
end pCon;


model SourceValveO "Output source valve"
  //Output flow negative - input flow positive
  pCon OutCon( nComp = nComp)
      annotation (Placement(transformation(
            extent={{-79.6,-10},{-59.6,10}}),
          iconTransformation(extent={{-79.6,-10},{-59.6,10}})));
  parameter Integer nComp = 1;
  parameter Real Tv(unit="h") = 0.00277778;
  parameter Real Cv(unit="1") = 0.352;
  input Real u;
  Real X;
  Real P;
  Real conc[nComp];
equation
  der(X)*Tv = u-X;
  OutCon.flowM =  if noEvent(P-100>0) then X/100*Cv*sqrt(P-100) else 0;
  OutCon.conc = conc;
  OutCon.P = P;
  annotation (
      Icon(coordinateSystem(preserveAspectRatio=false,
            extent={{-100,-100},{100,100}}, grid={0.1,0.1}),
        graphics={
          Polygon(
            points={{-60,40},{-60,-40},{0,0},{-60,40}},
            lineColor={0,0,255}, smooth=Smooth.None,
            fillColor={0,0,255}, fillPattern=FillPattern.Solid),
          Polygon(
            points={{60,40},{60,-40},{0,0},{60,40}},
            lineColor={0,0,255}, smooth=Smooth.None,
            fillColor={0,0,255}, fillPattern=FillPattern.Solid),
          Rectangle(
            extent={{-22.6,56},{25.4,50}}, lineColor={0,0,255},
            fillColor={0,0,255}, fillPattern=FillPattern.Solid),
          Rectangle(
            extent={{-2.4,56},{3.6,0}}, lineColor={0,0,255},
            fillColor={0,0,255}, fillPattern=FillPattern.Solid)}),
    Diagram(coordinateSystem(preserveAspectRatio=false,
          extent={{-100,-100},{100,100}}, grid={0.1,0.1}),
          graphics));
end SourceValveO;
```

Modelica Code 12.5: Simplified Tennessee Eastman model (5/8).

```
model SourceValveI "Input source valve"
  parameter Integer nComp = 1;
  parameter Real Tv( unit = "h") = 0.00277778;
  parameter Real Fmax( unit = "kmol/h") = 330.46;
  pCon OutCon( nComp = nComp)
      annotation (Placement(
        transformation(extent={{59.9,-10},{79.9,10}}),
        iconTransformation(extent={{59.9,-10},{79.9,10}})));
  input Real u;
  Real X;
  Real P;
  parameter Real conc[nComp];
equation
  der(X)*Tv = u-X;
  OutCon.flowM = -Fmax*X/100;
  OutCon.conc = conc;
  OutCon.P = P;
  annotation (Placement(
        transformation(extent={{60,-10},{80,10}}),
        iconTransformation(extent={{60,-10},{80,10}})),
    Icon(coordinateSystem(preserveAspectRatio=false,
         extent={{-100,-100},{100,100}}, grid={1,1}),
         graphics={
           Polygon(
             points={{-60,40},{-60,-40},{0,0},{-60,40}},
             lineColor={0,0,255}, smooth=Smooth.None,
             fillColor={0,0,255}, fillPattern=FillPattern.Solid),
           Polygon(
             points={{60,40},{60,-40},{0,0},{60,40}},
             lineColor={0,0,255}, smooth=Smooth.None,
             fillColor={0,0,255}, fillPattern=FillPattern.Solid),
           Rectangle(
             extent={{-22.6,56},{25.4,50}},
             lineColor={0,0,255}, fillColor={0,0,255},
             fillPattern=FillPattern.Solid),
           Rectangle(
             extent={{-2.4,56},{3.6,0}},
             lineColor={0,0,255}, fillColor={0,0,255},
             fillPattern=FillPattern.Solid)}),
    Diagram(coordinateSystem(preserveAspectRatio=false,
            extent={{-100,-100},{100,100}}, grid={1,1})));
end SourceValveI;
```

Modelica Code 12.6: Simplified Tennessee Eastman model (6/8).

```
model ReactorOpenLoop
  parameter Real u1S = 60.95327;
  parameter Real u2S = 25.023;
  parameter Real u3S = 39.25777;
  parameter Real u4S = 47.030248;
  input Real u1(start=60.95327);
  input Real u2(start=25.023);
  input Real u3(start=39.25777);
  input Real u4(start=44.1767);
  Real F1;
  Real F2;
  Real F3;
  output Real F4(displayUnit="kmol/hour") "Flow in stream 4";
  output Real P(displayUnit="kPa") "Total pressure";
  output Real Vl;
  output Real yA3;
  Real yB3;
  Real yC3;
  Real C;
  Reactor reactorSeparator
    annotation (Placement(transformation(
      extent={{12,2},{38,38}})));
  TESimplified.SourceValveI sourceValve1(nComp=4,
      conc={0.485,0.005,0.51,0}, X(start=60.953271, fixed=true))
    annotation (Placement(visible = true, transformation(
      extent = {{-14, 24}, {6, 44}}, rotation = 0)));
  TESimplified.SourceValveI sourceValve2(nComp=4,
      Fmax=22.46, conc={1,0,0,0}, X(start=25.02, fixed=true))
    annotation (Placement(visible = true, transformation(
      extent = {{-14, 2}, {6, 22}}, rotation = 0)));
  TESimplified.SourceValveO Purge(nComp=4,
      X(start=39.25777, fixed=true))
    annotation (Placement(visible=true, transformation(
      extent={{44,26},{64,46}}, rotation=0)));
  TESimplified.SourceValveO sourceValve4(Cv=4.17, nComp=4,
      X(start=47.0302048, fixed=true))
    annotation (Placement(transformation(
      extent={{44,2},{64,22}})));
equation
  connect(reactorSeparator.F3, Purge.OutCon)
      annotation (Line(points={{34.1,25.76},{43.05,25.76},
                               {43.05,36},{47.04,36}}));
  connect(sourceValve1.OutCon, reactorSeparator.F1)
      annotation (Line(points={{2.99,34},{6,34},
                               {6,25.742},{15.965,25.742}}));
  connect(sourceValve2.OutCon, reactorSeparator.F2)
      annotation (Line(points={{2.99,12},{8,12},
                               {8,19.874},{15.939,19.874}}));
  u1 = sourceValve1.u;
  u2 = sourceValve2.u;
  u3 = Purge.u;
  u4 = sourceValve4.u;
  F1 = reactorSeparator.F1.flowM;
  F2 = reactorSeparator.F2.flowM;
```

Modelica Code 12.7: Simplified Tennessee Eastman model (7/8).

```
  F3 = -reactorSeparator.F3.flowM;
  F4 = -reactorSeparator.F4.flowM;
  P  = reactorSeparator.P;
  Vl = 100*reactorSeparator.Vliq/30;
  yA3 = reactorSeparator.F3.conc[1]*100;
  yB3 = reactorSeparator.F3.conc[2]*100;
  yC3 = reactorSeparator.F3.conc[3]*100;
  C = F3/F4*(2.206*yA3+6.177*yC3)/100;
  connect(sourceValve4.OutCon, reactorSeparator.F4)
       annotation (Line(points={{47.04,12},{40,12},
             {40,15.464},{34.139,15.464}}, color={0,0,0}));
  annotation (
     Diagram(coordinateSystem(preserveAspectRatio=false,
             extent={{-100,-100},{100,100}})),
     experiment(StopTime = 60, StartTime = 0, Tolerance = 1e-06,
                Interval = 0.12),
    __Dymola_experimentSetupOutput(textual=true, derivatives=false));
end ReactorOpenLoop;


annotation (
    uses(Modelica(version="3.2.2")),
    experiment(StopTime=3),
    __Dymola_experimentSetupOutput,
    Documentation(info="<html>
 <H2 align=center>TESimplified - The Simplified Tennessee Eastman Process</H2>

<b>Author</b><br>
<i>Carla Martin</i><br>
Department of Computer Science and Automatic Control, UNED<br>
Madrid, Spain <br>
email:<A HREF=\"mailto:carla@dia.uned.es\">carla@dia.uned.es</A><br>
<br>
<p>
 The Simplified Tennessee Eastman (TES) model (Ricker,1993)
 is a simplification of the model described in (Downs
and Vogel, 1993), with only one process unit and eight
state variables.
<br>
This process unit is a combination of a
reactor and a separator of the original TE process.
<br>
The TES model is also a well-known benchmark
process. It is a multi-input multi-output, nonlinear system,
open-loop unstable, and contains fast and low dynamics.
</p>
<p><H2 align=left>References</H4></p>
<p>Ricker, N.L. (1993): <i>Model predictive control of a continuous,
nonlinear, two-phase reactor.</i> Journal of Process Control, 2, 109-123.</p>
<p>Downs, J.J. and Vogel, E.F. (1993): <i>A plant-wide industrial
process control problem.</i> Computers and Chemical
Engineering, 17(3), 245255.</p>
</HTML> "));

end TESimplified;
```

Modelica Code 12.8: Simplified Tennessee Eastman model (8/8).

12.6 Task 3

The `ReactorOpenLoop` model is a description of the model in open loop, this is, without any controller. This system should be controlled in order to get the desired response. The controller model may be described in Modelica or using another simulation tool (e.g., Matlab/Simulink).

In this task, we are going to describe the controller using Simulink blocks and embed the `ReactorOpenLoop` model within a Simulink block. To this end, we will export the `ReactorOpenLoop` model as a Functional Mockup Unit (FMU).

Any Modelica model can be generated as an FMU using the OpenModelica or Dymola environments. This FMU can be embedded into other environments, such as Matlab/Simulink. The variables declared as inputs/outputs in the Modelica model will be the inputs/outputs of the generated Simulink block.

Therefore, we have to declare the manipulated variables of the `ReactorOpenLoop` model (u_1, u_2, u_3 and u_4) as inputs, and the controlled variables (F_4, P, V_{liq}, and the concentration of component A in the purge denoted by $F_3.conc_A$) as outputs.

To generate the FMU using OpenModelica, the Modelica model has to be opened in OMEdit. Then, by clicking on *FMI / Export FMU*, the FMU is generated in the working directory. The working directory can be changed by clicking on *Tools / Options*.

With release R2017b and beyond, Simulink supports simulation and integration workflows using the Functional Mockup Interface (FMI). The FMI import block allows you to import FMU version 1.0 or version 2.0 into a Simulink model and supports the following use cases: model exchange, and co-simulation.

Finally, control the system in the nominal operating point (see Table 12.1) using the four discrete-time PI and the proportional controller described in the paper (Ricker 1993) (see Figure 12.2). These PI follow the following velocity form equation, with a sampling period $\Delta t = 0.1$ h.

$$\Delta u_n = u_n - u_{n-1} = K_c \cdot \left[e_n - e_{n-1} + \frac{\Delta t}{\tau_1} \cdot e_n \right] \tag{12.17}$$

where e_n is the signal error, and K_c and τ_n are the controller parameter. The value of these parameters for each control loop is shown in Table 12.2.

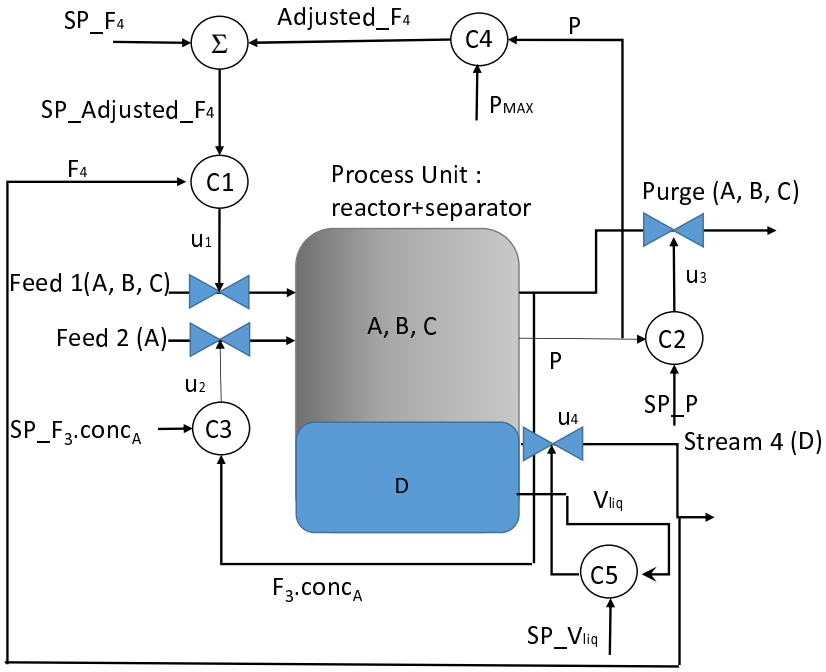


Figure 12.2: Multiloop control strategy for the TES process.

Table 12.2: Control parameters.

Control loop	**Manipulated variable**	**Controlled variable**	K_c	τ_1
C1	u_1	F_4	0.1	1.0
C2	u_3	P	-0.25	1.5
C3	u_2	$F_3.conc_A$	2.0	3.0
C4	Adjusted set-point for F_4	P	0.7	3.0
C5	u_4	V_{liq}	-1.4	–

12.7 Solution to Task 3

The Simulink model of the controlled plant is shown in Figure 12.3 and the plots obtained simulating the model with Matlab/Simulink are shown in Figure 12.4.

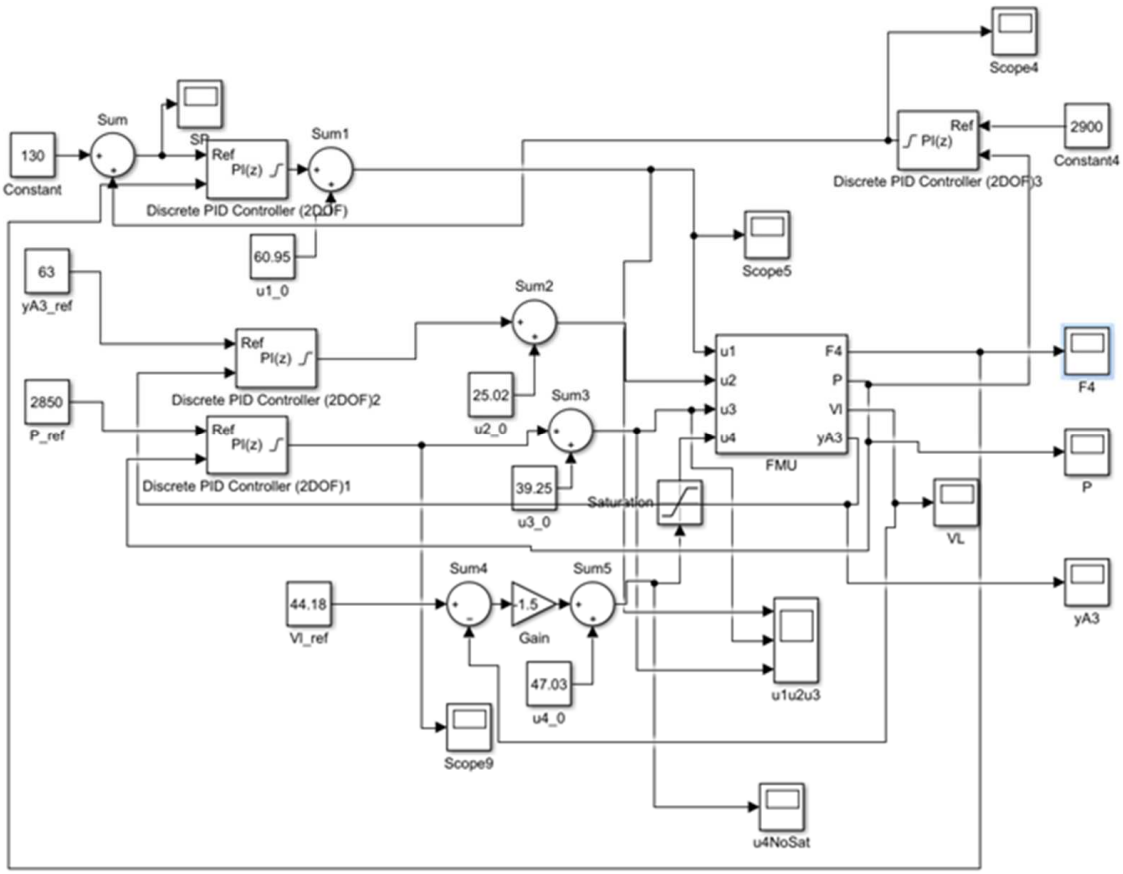


Figure 12.3: Simulink TES process in closed loop.

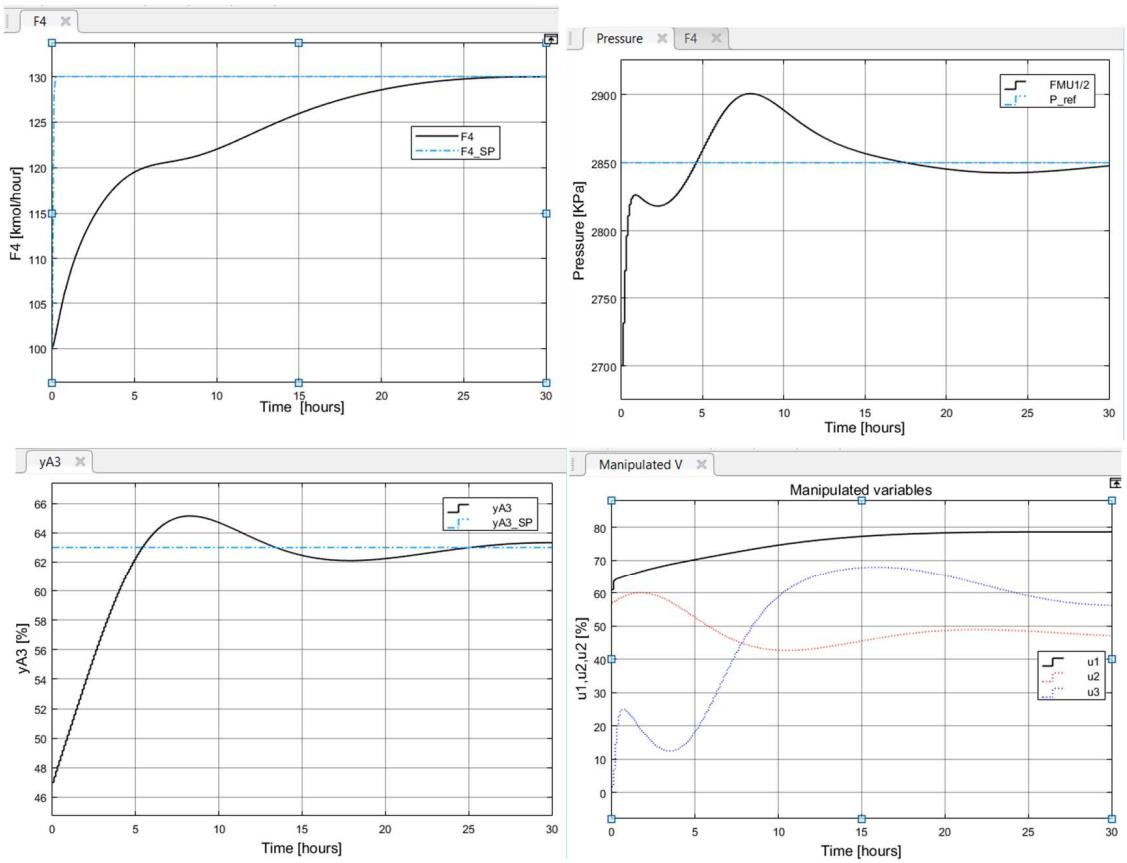


Figure 12.4: Plots obtained simulating the closed-loop TES model.

PEM fuel cell

Purpose of this assignment

- Practice modeling PEM fuel cells, which implies modeling gas and liquid diffusion in porous media, electrochemical reactions, and electric circuits.
- Use spatial discretization to solve PDEs in Modelica.

13.1 System description

Fuel cells are electrochemical devices that transform chemical energy into electric energy. In operation, unlike batteries, fuel cells are fed continuously with fuel. A fuel cell, which is fed with hydrogen and oxygen to obtain water and electricity as products, is depicted in Figure 13.1.

The internal-combustion engines used in automobiles are also continuously fed with diesel, gasoline, etc. The difference between internal-combustion engines and fuel cells is related with the use of the chemical energy. The burning reaction in internal-combustion engines is a fast oxidation, whose reaction heat is partially transformed into mechanical energy. On the contrary, the reaction in hydrogen fuel cells is performed in two separated steps, as described in (13.1) and (13.2), which allows to transform chemical energy into electrical energy efficiently.

$$H_2 \rightleftharpoons 2H^+ + 2e^- \tag{13.1}$$

$$\frac{1}{2}O_2 + 2H^+ + 2e^- \rightleftharpoons H_2O \tag{13.2}$$

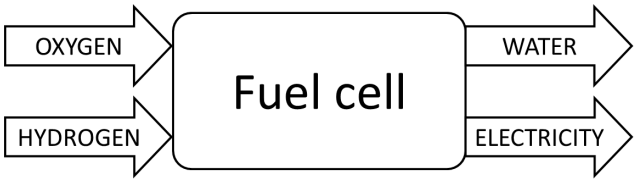


Figure 13.1: Products and reactants in hydrogen fuel cells.

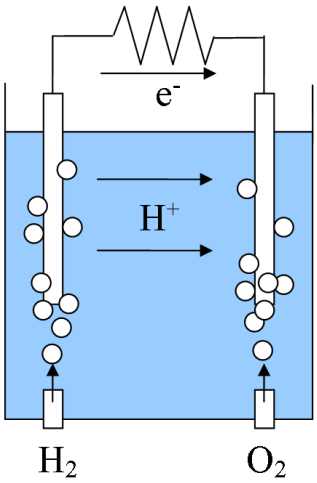


Figure 13.2: Hydrogen fuel cell with liquid electrolyte.

The operation scheme of the first hydrogen fuel cell is shown in Figure 13.2. The reaction in the left electrode (anode) is $H_2 \rightleftharpoons 2H^+ + 2e^-$, whereas the reaction in the right electrode (cathode) is $\frac{1}{2}O_2 + 2H^+ + 2e^- \rightleftharpoons H_2O$. The electrons generated in the reaction are conducted through an external circuit, which allows to use the electrical energy. The protons generated in the anode are conducted to the cathode through the aqueous electrolyte, in order to complete the electrochemical reaction.

There exist several types of fuel cells, which are classified attending to their fuel, constitutive materials, and design. The type of fuel cell considered in this assignment is named **PEM fuel cell**. PEM stands for Proton Exchange Membrane or Polymeric Electrolyte Membrane. PEM fuel cells have a solid polymer electrolyte, and use pure hydrogen, and oxygen (air or pure oxygen) as fuel.

Solid polymer electrolytes have some advantages over liquid electrolytes. The most important advantage of polymeric materials in this context is their high proton conductivity. Dupont's NAFION, and the membranes manufactured by Dow Chemical, are examples of widely-used commercial proton exchange membranes.

The structure of a PEM fuel cell (PEMFC) is schematically represented in Figure 13.3. The PEMFC is composed of the following fundamental parts: the active layers, the diffusion layers, the terminals of the anode and the cathode, and the

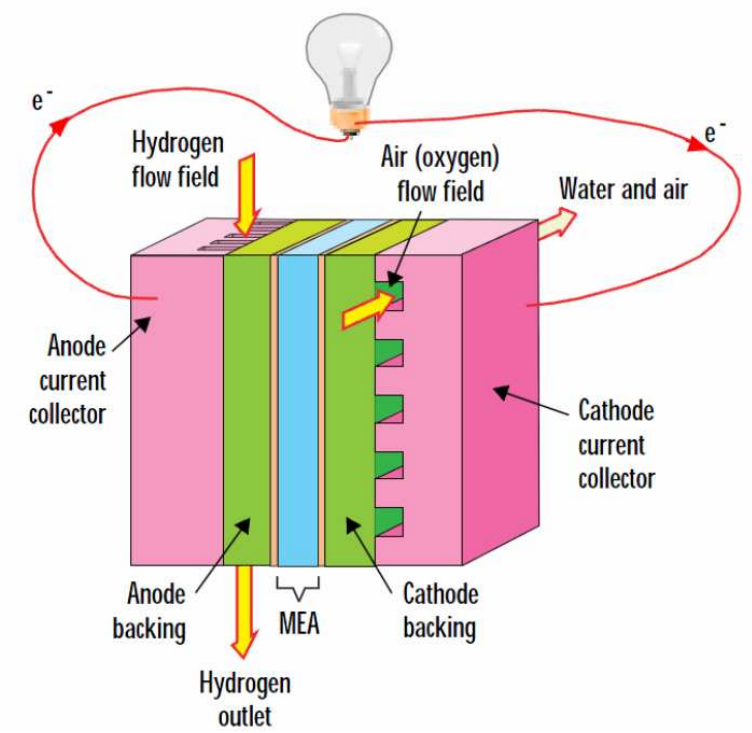


Figure 13.3: Schematic diagram of a PEM fuel cell.

polymeric membrane that separates them. The membrane, active layers and diffusion layers are usually named MEA (Membrane Electrode Assembly). In addition, the PEMFC is composed of a set of auxiliary components that are necessary for its operation, such as the seals, the refrigeration system, the humidification system and the voltage (or electrical current) control. The fundamental parts are briefly described below.

- **Membrane**. The membrane is a solid polymer electrolyte, usually sulfonated polytetrafluoroethylene (PTFE), which is placed between the catalyst layers of the anode and cathode. The free movement of protons in the electrolyte structure, from anode to cathode, while the electrolyte is well hydrated, is the main property of the membrane. In addition, the membrane does not allow the crossover of hydrogen and oxygen.

- **Active layers**. The active layer, also named catalyst layer, is a thin layer where the electrochemical reactions are carried out. It is located between the membrane and the diffusion layer. The active layer has usually three components: catalyst (e.g., platinum), liquid electrolyte, and an electrical conductor (usually carbon powder) used as physical support. The ink obtained of mixing these components is deposited on the surface of the diffusion layer of the anode and cathode, or on the membrane surfaces.

– **Diffusion layers**. The gas diffusion layers (GDL) are fabricated using porous material, which allows the diffusion of gases and water to the catalyst layer. High electric conductivity is a very important property of the diffusion layer, as it has to allow the movement of the electrons between the collector plates and the catalyst layers. The diffusion layer is often fabricated on carbon, paper or cloth, due to their good conductivity and corrosion resistance. Metallic foams are also used for this propose.

– **Collector plates**. The collector plates maintain the fuel cell structure, and distribute the gases to the MEA. They are one of the most expensive components of the PEMFC, as they are usually made of graphite, or machined metals. Some important physical properties of the plates are high electrical and thermal conductivities, and impermeability to gases and liquid water. The plates are usually designed with channels, which allow the adequate distribution of gases and the evacuation of excess water.

Modeling and simulation have been widely employed in the analysis of PEMFC operation. For example, the diffusion of gases and liquid water in the porous media is extensively studied, since the access of the fuel to the catalyst layer, in the presence of liquid water, is one of the most important topics. The dependence of the membrane conductivity on the water concentration, and the influence of other PEMFC design parameters on the electrical performance have also a great interest.

Further reading. Excellent references on fuel cells are (*Fuel Cell Handbook* 2004, Larminie & Dicks 2000, O'Hayre et al. 2006). Modeling of PEMFC is discussed in (Springer & Zawodzinsky 1991, Bernardi & Verbrugge 1992, Bevers et al. 1997, Broka & Ekdunge 1997, Rubio et al. 2010). These suggested readings are not necessary to complete the proposed tasks.

13.2 Outline of the assignment

Since the most important phenomena of the PEMFC occur in its cathode, only this part of the cell will be considered. The relevant species in each layer of the cathode are shown in Figure 13.4. The transport phenomena taken into account in each layer are summarized in Table 13.1. The physical-chemical phenomena modeled in this assignment are itemized below.

– Transport of liquid-phase and gas-phase water in the diffusion layer and the active layer (also referred to as catalyst layer).

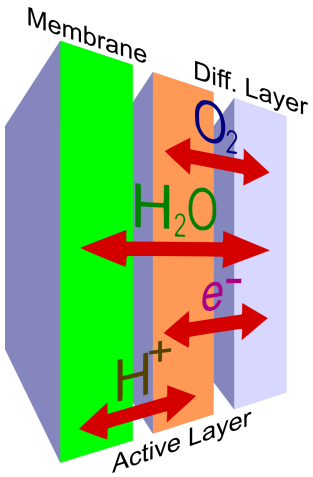


Figure 13.4: Species considered in the model of the PEMFC cathode.

Table 13.1: Transport phenomena considered in each layer.

Phenomenon	**Membrane**	**Catalyst layer**	**Diffusion layer**
Steam water diffusion	X	X	X
Liquid water diffusion	X	X	X
Oxygen diffusion	–	X	X
Protonic conduction	X	X	–
Electronic conduction	–	X	X

– Transport of oxygen in the diffusion layer and the active layer.

– Electrical conduction in the active layer and the diffusion layer.

– Proton conduction in the membrane and the active layer.

– Electrochemical reaction in the active layer.

The proposed modeling tasks have been arranged in order of increasing complexity, as described below.

Task 1 *Diffusion of a gas in a porous media.* Porous materials that allow gas diffusion are often used in making the PEMFC electrodes. The objective of this task is to model the one-dimensional diffusion of gas in a porous material, with the aim of analyzing the spatial distribution of the gas pressure and molar flow rate, in response to changes in the gas pressure applied at one end of the material. This task is an introduction to the control volume technique, which is applied in the other tasks to model the dependence of physical quantities with respect to the spatial coordinates.

Task 2 *Diffusion of a binary mixture of gases (oxygen, and steam water), and a liquid (liquid water) in porous media.* The model developed in Task 1 is extended to describe the molar balances and the diffusion of oxygen, steam water and liquid water. As in Task 1, it is assumed that the diffusion of the species occurs only in one direction and the porous medium is divided into control volumes along that axis. The balances of steam water and liquid water due to the evaporation and condensation processes are taken into account.

Task 3 *Electric conduction and electrochemical reaction.* The objective is twofold. Firstly, to model the electron and proton conduction in the membrane, the catalyst layer and the diffusion layer of the cathode. Secondly, to model the electrochemical reaction that takes place in the catalyst layer.

Combining these models with the models developed in the previous tasks, the *complete model of the PEM fuel cell* is obtained. Finally, the model of the *electrical circuit to polarize the PEMFC model* is developed.

13.3 Task 1

The objective is to develop a one-dimensional model of gas diffusion in a porous material, to calculate the gas pressure and molar flow rate as a function of the spatial coordinate (x-axis) and time. The steps to develop the model are detailed below.

1. **Spatial discretization**. Divide the porous material into N equal control volumes (see the example shown in Figure 13.5). Let V_{VC} and L_{VC} represent the volume, and the length in the x-direction, of the control volume.

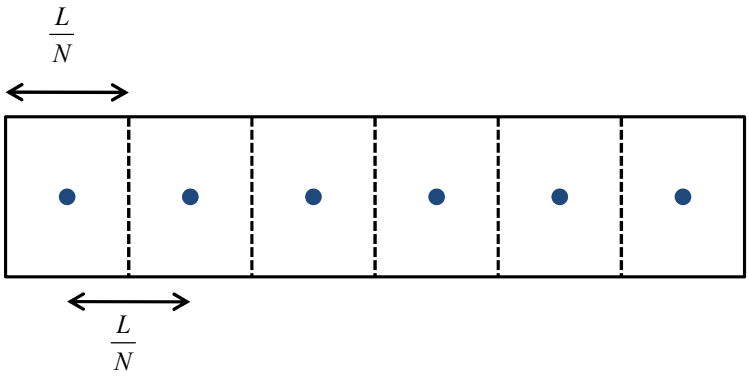


Figure 13.5: Porous medium divided into $N = 6$ control volumes of equal size.

2. **Stirred tank approximation**. Assume that the intensive properties of the gas are time-dependent, and spatially uniform inside each control volume.

The gas is allowed to flow through the interface between contiguous control volumes. Each interface between contiguous control volumes defines a control plane for gas transport. The gas that flows through a control plane has the same properties as the gas contained inside the upstream control volume.

3. **Control planes**. The cross-sectional area of a control plane is S_{VC}.

$$V_{VC} = S_{VC} \cdot L_{VC} \tag{13.3}$$

4. **Mole balance**. The moles of gas stored in a control volume depend on the gas transport through the control planes of the aforesaid control volume (see Figure 13.6).

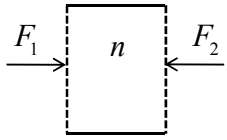


Figure 13.6: Gas mole balance over the control volume, $\frac{dn}{dt} = F_1 + F_2$.

5. **Equation of state of an ideal gas**. Assume that the gas behaves ideally.

$$p \cdot V = n \cdot R \cdot T \tag{13.4}$$

6. **Gas volume in porous media**. The volume (V) occupied by the gas in a control volume is calculated as the product of the medium porosity (ε) and the volume of the control volume (V_{VC}). A constant porosity of the medium is assumed.

$$V = \varepsilon \cdot V_{VC} \tag{13.5}$$

7. **Isothermal behavior** Assume that the temperature of the gas contained in a control volume is a constant, T_0.

$$T = T_0 \tag{13.6}$$

8. **Gas transport**. The gas transport through a control plane depends on the properties in the control volumes that share that control plane (see Figure 13.7).

   Let's assume that the relevant mass transfer mechanisms are Knudsen flux and ordinary diffusion. The relationship between the gradient of gas pressure

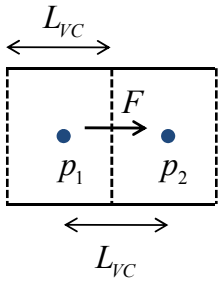


Figure 13.7: Molar flow rate (F) through the control plane.

(∇p) within the porous medium and the molar gas flow rate per area unit (J) of the porous medium, in both diffusion mechanisms, is

$$J = -\frac{\varepsilon}{\tau^2} \cdot \frac{D}{R \cdot T} \cdot \nabla p \tag{13.7}$$

where the parameter τ (units: unitless) represents the tortuosity of the medium, and D (units: m^2/s) is the coefficient corresponding to the diffusion mechanism or mechanisms. If only Knudsen diffusion is considered, D is the Knudsen diffusion coefficient. Similarly for ordinary diffusion. If both diffusion mechanisms are considered, the value of D is calculated from the following expression:

$$\frac{1}{D} = \frac{1}{D_{ord}} + \frac{1}{D_K} \tag{13.8}$$

Eq. (13.7) is applied to calculate the gas flow rate due to diffusion through the control plane that separates two control volumes.

$$\frac{F}{S_{VC}} = \frac{\varepsilon}{\tau^2} \cdot \frac{D}{R \cdot T} \cdot \frac{p_1 - p_2}{L_{VC}} \tag{13.9}$$

where F (units: mol/s) is the molar flow rate that is diffused from the control volume with the pressure p_1 to the control volume with pressure p_2 (see again Figure 13.7). The cross-sectional area of the control plane through which the gas flow occurs is S_{VC} (units: m^2).

9. **Connecting control volumes and transport phenomena**. The model is formulated by discretization of the material into control volumes. Contiguous control volumes are connected by a model of the gas flow through the control plane. Figure 13.8 shows a model composed of control volumes (labeled as "VC"), connected to each other by a transport phenomenon model (labeled as "FT"). In this case, the transport phenomenon is the gas diffusion.

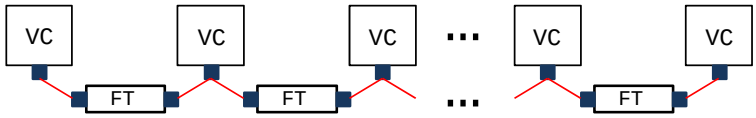


Figure 13.8: Model composed connecting control volumes and transport phenomena.

In Figure 13.8, the filled squares represent the connectors. The control volume model has a connector, while the diffusion model has two connectors. A possible selection of the connector variables is the following:

- Across variables: pressure (p) and temperature (T) of the gas within the control volume.
- Through variable: input molar flow rate (F).

10. **Boundary conditions**. The values of the gas pressure at the ends of the medium are imposed as a boundary condition. This is modeled by connecting pressure sources, and transport phenomena describing diffusion through a distance $L_{VC}/2$, to the control volumes located at the ends. The diagram of the complete model is shown in Figure 13.9.

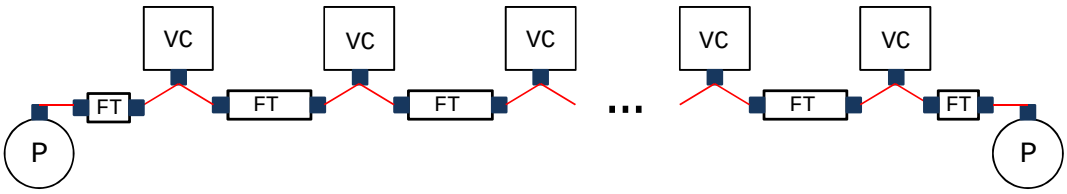


Figure 13.9: Porous medium with boundary conditions.

The **proposed task** consists in developing the Modelica models enumerated below.

1. The connector.
2. The control volume.
3. The transport phenomenon.
4. The pressure source.
5. The complete model, including the boundary conditions (see again Figure 13.9).

Divide the porous media into $N = 20$ control volumes. The model parameters are listed in Table 13.2. Observe that, as $S_{VC} = 1$ m^2, the molar flow rate is calculated per unit area.

Table 13.2: Parameters used in Task 1.

Quantity	**Meaning**	**Value**
T_0	Temperature	300 K
ε	Porosity	0.1
R	Ideal gas constant	8.31447 J/mol/K
L	Total length of the layer	0.003 m
τ	Tortuosity	5
D	Diffusion coefficient	2.5e-2 m^2/s
S_{VC}	Diffusion cross-sectional area	1 m^2

Simulate the complete model to calculate the evolution of the gas pressures and the molar flow rates. The initial pressure of the gas into the porous media is zero. The boundary conditions for the gas pressure are 10^5 Pa (=1 bar) and $2 \cdot 10^5$ Pa (=2 bars). Plot the pressure and the molar flow rate in the control volumes number 1, 5, 10, 15 and 20.

13.4 Solution to Task 1

The models are described in Modelica Code 13.1 and 13.2. The pressure and the molar flow rate in the control volumes number 1, 5, 10, 15 and 20 are plotted in Figures 13.10 – 13.12.

```
connector P_F
  Modelica.SIunits.Pressure P "Gas pressure";
  Modelica.SIunits.Temperature T "Temperature";
  flow Real F  (unit="mol/s")   "Molar flow rate of gas";
end P_F;

model VC
  Modelica.SIunits.Volume V;
  Modelica.SIunits.Volume Vvc "Volume of the control volume";
  Real n( unit="mol");
  parameter Modelica.SIunits.Length Lvc "Length of the control volume";
  parameter Modelica.SIunits.Area Svc "Cross-sectional area of control volume";
  parameter Modelica.SIunits.Temp_K To;
  parameter Real epsilon "Porosity of the medium";
  constant Real R=8.31447( unit="J/mol/K")   "Ideal gas constant";

  P_F p_f;
equation
  p_f.F = der(n);
  p_f.P*V = n*R*p_f.T;
  V = epsilon*Vvc;
  Vvc = Svc*Lvc;
  p_f.T = To;
end VC;

model FT
  parameter Modelica.SIunits.Length Lvc "Length of the control volume";
  parameter Modelica.SIunits.DiffusionCoefficient D "Diffusion coeff.";
  parameter Real epsilon "Porosity of the medium";
  parameter Real tau "Tortuosity of the medium";
  parameter Modelica.SIunits.Area Svc "Area of the control volume";
  constant Real R=8.31447( unit="J/mol/K")   "Ideal gas constant";

  P_F p_f_1;
  P_F p_f_2;
equation
  p_f_2.F = if (p_f_2.P > p_f_1.P) and p_f_2.P > 0
      then (Svc*epsilon*D*(p_f_2.P - p_f_1.P))/(tau^2*R*p_f_2.T)
      else if (p_f_1.P > p_f_2.P) and p_f_1.P > 0
          then (Svc*epsilon*D*(p_f_2.P - p_f_1.P))/(tau^2*R*p_f_2.T)
          else 0;
  p_f_1.F = -p_f_2.F;
end FT;

model Source_P
  parameter Modelica.SIunits.Pressure Po;
  parameter Modelica.SIunits.Temperature To;

  P_F p_f;
equation
  p_f.P = Po;
  p_f.T = To;
end Source_P;
```

Modelica Code 13.1: Gas diffusion in porous medium (1/2).

```
model Diffusion_Layer
  parameter Integer N=20;
  parameter Modelica.SIunits.Length L=0.003 "Total length of the layer";
  parameter Modelica.SIunits.Temp_K To=300;
  parameter Modelica.SIunits.DiffusionCoefficient D=2.5e-2 "Diffusion coeff.";
  parameter Real epsilon=0.1 "Porosity of the medium";
  parameter Modelica.SIunits.Area Svc =1 "Area of the control volume";
  parameter Real tau=5 "Tortuosity of the medium";
  parameter Modelica.SIunits.Length l=L/N;
  parameter Modelica.SIunits.Pressure Pres_1 =1e5 "Boundary pressure 1";
  parameter Modelica.SIunits.Pressure Pres_2 =2e5 "Boundary pressure 2";
  parameter Real lv_ft[N + 1]=
       {if i == 1 or i == (N + 1) then l/2 else l    for i in 1:(N + 1)};

  FT f_t[N + 1] (
      Lvc=lv_ft,
      Svc=fill(Svc, N + 1),
      D=fill(D, N + 1),
      epsilon=fill(epsilon, N + 1),
      tau=fill(tau, N + 1));

  VC v_c[N] (
      Lvc=fill(l, N),
      Svc=fill(Svc, N),
      To=fill(To, N),
      epsilon=fill(epsilon, N));

  Source_P p1(Po=Pres_1, To=To);
  Source_P p2(Po=Pres_2, To=To);

equation
  connect(p1.p_f, f_t[1].p_f_1);

  for conta in 1:N loop
    connect(f_t[conta].p_f_2, v_c[conta].p_f);
    connect(v_c[conta].p_f, f_t[conta + 1].p_f_1);
  end for;

  connect(f_t[N + 1].p_f_2, p2.p_f);

end Diffusion_Layer;
```

Modelica Code 13.2: Gas diffusion in porous medium (2/2).

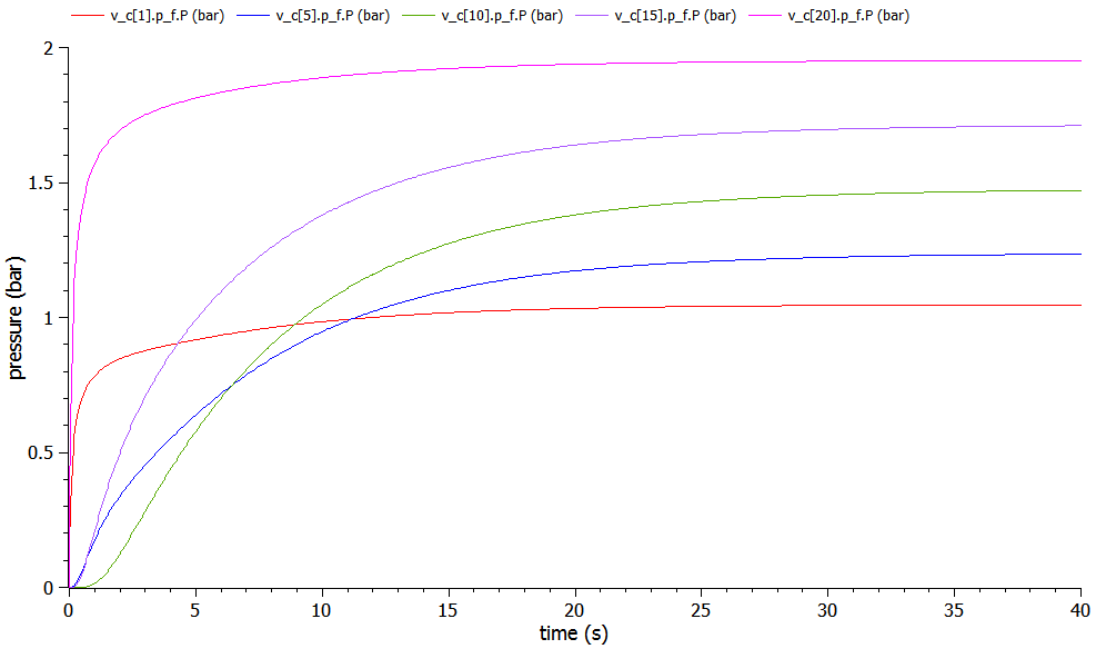


Figure 13.10: Pressure in the control volumes number 1, 5, 10, 15 and 20.

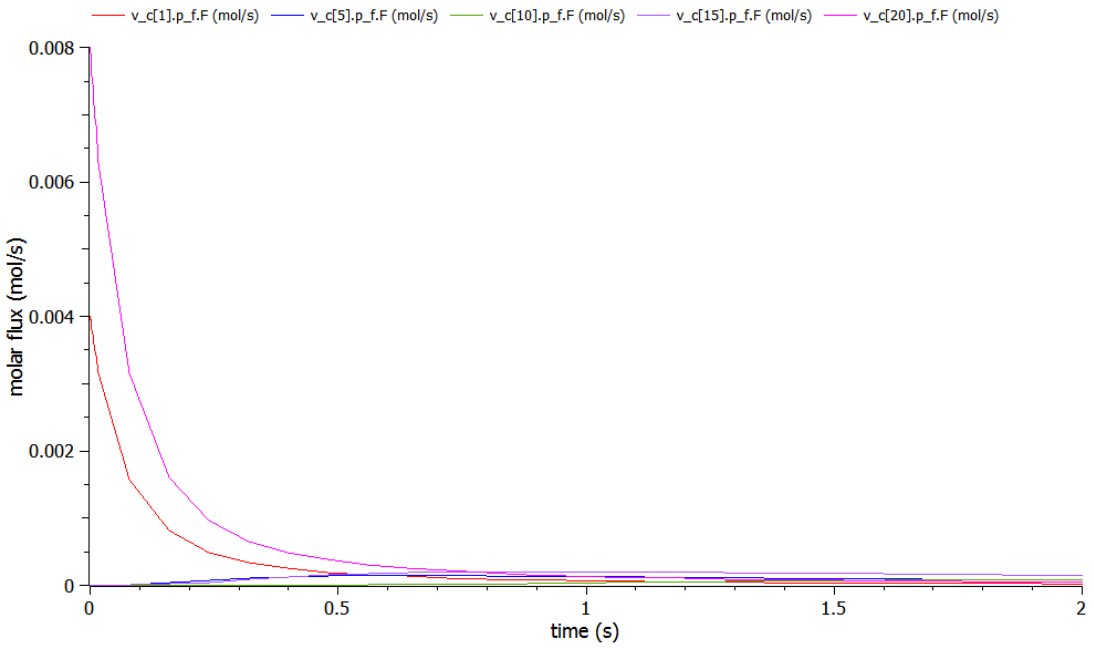


Figure 13.11: Molar flow rate in the control volumes number 1, 5, 10, 15 and 20.

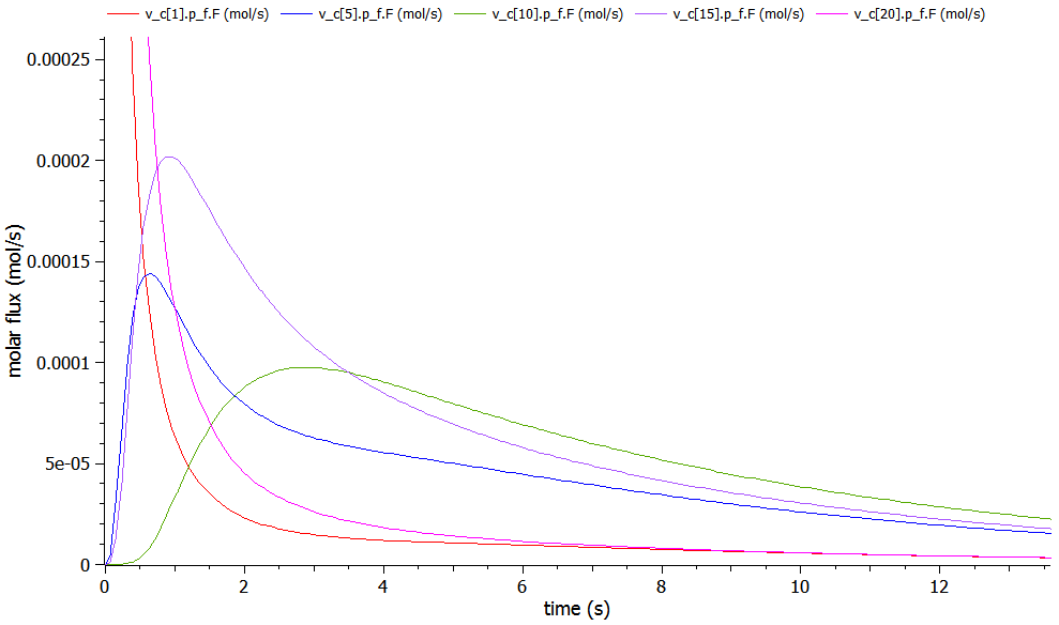


Figure 13.12: Detail of the molar flow rate in the control volumes number 1, 5, 10, 15 and 20.

13.5 Task 2

A relevant phenomenon that occurs in the PEMFC cathode is the interaction of two gaseous species: oxygen and steam water. Also, in PEMFC operated at temperatures below 100 ^{0}C, the presence of liquid water plays a fundamental role that must also be taken into account.

The objective of this task is to extend the model developed in Task 1, by considering a second gaseous species and the liquid phase. The phenomena modeled are the diffusion of the species, and the balance of steam water and liquid water due to evaporation and condensation processes. As in Task 1, it is assumed that the diffusion of the species only occurs along the x axis, and the porous medium is divided into control volumes along that axis.

The **control volume** represents a fraction of the porous medium, that contains a mixture of two gaseous species (oxygen and steam water) and liquid water. The model of the control volume comprises: the molar balances of oxygen, steam water and liquid water; the state equation of the gas mixture; the energy balance; the expression to calculate the volume occupied by the gas mixture; and the equations that describe water evaporation and condensation. This is described in detail below.

1. **Molar balance**. The molar balance of each species in a control volume is described as follows:

$$\frac{dn_{O_2}}{dt} = F_{O_2} \tag{13.10}$$

$$\frac{dn_{H_2O,g}}{dt} = F_{H_2O,g} + G_{H_2O,g} \tag{13.11}$$

$$\frac{dn_{H_2O,l}}{dt} = F_{H_2O,l} + G_{H_2O,l} \tag{13.12}$$

where n_{O_2}, $n_{H_2O,g}$ and $n_{H_2O,l}$ represent the number of moles (units: mol) of oxygen, steam water and liquid water contained in the control volume; F_{O_2}, $F_{H_2O,g}$ and $F_{H_2O,l}$ represent the molar flow rate (units: mol/s) of oxygen, steam water and liquid water entering the control volume; and $G_{H_2O,g}$ and $G_{H_2O,l}$ represent the moles of steam water and liquid water generated per time unit (units: mol/s) in the control volume due to phase transition.

2. **Mass and volume of liquid water**. The moles ($n_{H_2O,l}$) and mass ($m_{H_2O,l}$) of liquid water are related by Eq. (13.13), where $M_{H_2O} = 18$ gr/mol is the molar mass of water.

$$n_{H_2O,l} = \frac{m_{H_2O,l}}{M_{H_2O}} \tag{13.13}$$

The volume ($V_{H_2O,l}$) and mass of liquid water are related by Eq. (13.14), where $\rho_{H_2O,l}$ represents the liquid water density.

$$V_{H_2O,l} = \frac{m_{H_2O,l}}{\rho_{H_2O,l}} \tag{13.14}$$

3. **Equation of state for an ideal gas mixture**. Assuming an ideal mixture, the state equations of the gas species are the following:

$$p_{O_2} \cdot V_g = n_{O_2} \cdot R \cdot T_g \tag{13.15}$$

$$p_{H_2O,g} \cdot V_g = n_{H_2O,g} \cdot R \cdot T_g \tag{13.16}$$

where V_g is the total volume (units: m^3) occupied by the gas mixture; T_g is the temperature (units: K) of the gas mixture; and p_{O_2} and $p_{H_2O,g}$ are the partial pressures (units: Pa) of oxygen and steam water respectively. The *partial pressure* of a gas is defined as the pressure that the gas would exert if it were alone in all the volume.

As we are assuming that the gas mixture is ideal, the total pressure (p_g) of the gas mixture is the sum of the partial pressures of the components (Dalton's Law of partial pressures).

$$p_g = p_{O_2} + p_{H_2O,g} \tag{13.17}$$

4. **Gas volume in a porous media**. The porous media is composed of solid material and pores. The volume of the solid material (V_s) plus the volume of the pores (V_{pore}) is equal to the total volume (V_{VC}) of the control volume. The gas mixture and the liquid water can only occupy the pore volume. In particular, the volume occupied by the gas mixture (V_g) is equal to the total pore volume (V_{pore}) minus the volume occupied by liquid water ($V_{H_2O,l}$).

$$\begin{aligned} V_{VC} &= V_{pore} + V_s & (13.18)\\ V_{pore} &= V_g + V_{H_2O,l} & (13.19) \end{aligned}$$

Unitless parameters that describe the volume ratio occupied by pores, solid material, the gas mixture, and liquid water are defined as follows:

$$\varepsilon_{pore} = \frac{V_{pore}}{V_{VC}} \qquad \varepsilon_s = \frac{V_s}{V_{VC}} \qquad \varepsilon_g = \frac{V_g}{V_{VC}} \qquad \varepsilon_{H_2O,l} = \frac{V_{H_2O,l}}{V_{VC}} \tag{13.20}$$

These parameters satisfy the following relationships:

$$\begin{aligned} \varepsilon_{pore} + \varepsilon_s &= 1 & (13.21)\\ \varepsilon_{pore} &= \varepsilon_g + \varepsilon_{H_2O,l} & (13.22) \end{aligned}$$

5. **Water load**. The water load (χ) is a unitless quantity that represents the relationship between the masses of liquid water and solid inside the control volume.

$$\chi = \frac{m_{H_2O,l}}{m_s} \tag{13.23}$$

There is a maximum limit for the water load (χ_{max}) in a porous medium. This limit is reached when the pore volume is completely filled with liquid water

($V_{H_2O,l} = V_{pore}$) and, consequently, the gas volume is zero ($V_g = 0$). As the maximum mass of liquid water inside the porous material is:

$$m_{H_2O,l,\max} = \rho_{H_2O,l} \cdot V_{pore} \tag{13.24}$$

the maximum value of the water load is:

$$\chi_{\max} = \frac{m_{H_2O,l,\max}}{m_s} = \frac{\rho_{H_2O,l} \cdot V_{pore}}{m_s} \tag{13.25}$$

The **relative water load** (χ_{rel}) provides a good indication of the amount of liquid water contained inside the porous medium. It is defined as the water load (χ) divided by the maximum water load ($\chi_{\max}$).

$$\chi_{rel} = \frac{\chi}{\chi_{\max}} \tag{13.26}$$

6. **Mass of solid material**. The mass of solid material (m_s) inside the control volume is related with the density of the porous material (ρ_s), as shown in Eq. (13.27).

$$m_s = V_{VC} \cdot \rho_s \tag{13.27}$$

7. **Isothermal behavior**. The temperatures of the gas mixture and the liquid water are assumed to be equal and time-independent. The constant temperature of the gas mixture and the liquid water is referred to as T_0.

$$T_g = T_0 \tag{13.28}$$
$$T_{H_2O,l} = T_0 \tag{13.29}$$

8. **Water evaporation and condensation**. The molar flow rate of liquid water ($G_{H_2O,l}$, units: mol/s) generated inside the control volume due to the condensation of steam water can be calculated as follows:

$$G_{H_2O,l} = V_{VC} \cdot \frac{\alpha_v \cdot \beta}{R \cdot T_g} \cdot \left(p_{H_2O,g} - p^{sat}_{H_2O}\right) \tag{13.30}$$

where V_{VC} (units: m^3) is the volume of the control volume; α_v (units: m^2/m^3) and β (units: m/s) are two parameters referred to as *specific condensation surface* and *mass transfer coefficient* respectively; R (units: Pa·m^3/mol/K) is

the ideal gas constant; T_g (units: K) is the gas temperature; and $p_{H_2O,g}$ and $p^{sat}_{H_2O}$ (units: Pa) are the partial pressure of steam water and the saturation pressure of water respectively.

The saturation pressure of water ($p^{sat}_{H_2O,g}$) depends mainly on temperature (T_g)

$$p^{sat}_{H_2O} = p^{sat}_{0_{H_2O}} \cdot \exp\left[\left(\frac{1}{T^{sat}_0} - \frac{1}{T_g}\right) \cdot \frac{L_v \cdot M_{H_2O}}{R}\right] \tag{13.31}$$

where $p^{sat}_{0_{H_2O}}$ and T^{sat}_0 are known parameters; and the evaporation molar enthalpy (L_v, units: J/mol) can be estimated by using the expression written below, where the T_g temperature is expressed in Kelvin.

$$\begin{aligned} L_v \;=\; & 1.73287 \cdot 10^6 + 1.03001 \cdot 10^{-4} \cdot T_g - 4.47755 \cdot 10^1 \cdot T_g^2 + \\ & 7.6629 \cdot 10^{-2} \cdot T_g^3 - 5.5058 \cdot 10^{-5} \cdot T_g^4 \end{aligned} \tag{13.32}$$

During the phase transition, each mole of steam water produces one mole of liquid water, and vice versa. The molar flow rate of steam water generated inside the control volume due to evaporation is:

$$G_{H_2O,g} = -G_{H_2O,l} \tag{13.33}$$

Compiling the above equations, the **complete model of the control volume** shown in Table 13.3 is obtained.

In order to assess whether this set of equations completely describes the control volume, let's analyze its computational causality. To this end, we set the computational causality of the interface variables and select the state variables as described below.

- The quantities that describe the molar flow rate entering the control volume are input variables: F_{O_2}, $F_{H_2O,g}$ and $F_{H_2O,l}$.
- The moles of the species are state variables: n_{O_2}, $n_{H_2O,g}$ and $n_{H_2O,l}$.
- The following parameters have known constant values: M_{H_2O}, $\rho_{H_2O,l}$, ρ_s, R, V_{VC}, ε_{pore}, T_0, α_v, β, $p^{sat}_{0_{H_2O}}$ and T^{sat}_0.
- The following variables are evaluated from the model equations: $\frac{dn_{O_2}}{dt}$, $\frac{dn_{H_2O,g}}{dt}$, $\frac{dn_{H_2O,l}}{dt}$, $G_{H_2O,g}$, $G_{H_2O,l}$, $m_{H_2O,l}$, m_s, $V_{H_2O,l}$, V_g, V_{pore}, ε_g, T_g, p_{O_2}, $p_{H_2O,g}$, $p^{sat}_{H_2O}$, χ, $\chi_{\text{máx}}$, χ_{rel} and L_v.

Table 13.3: Complete model of the control volume.

$$\frac{dn_{O_2}}{dt} = F_{O_2} \tag{13.34}$$

$$\frac{dn_{H_2O,g}}{dt} = F_{H_2O,g} + G_{H_2O,g} \tag{13.35}$$

$$\frac{dn_{H_2O,l}}{dt} = F_{H_2O,l} + G_{H_2O,l} \tag{13.36}$$

$$n_{H_2O,l} = \frac{m_{H_2O,l}}{M_{H_2O}} \tag{13.37}$$

$$V_{H_2O,l} = \frac{m_{H_2O,l}}{\rho_{H_2O,l}} \tag{13.38}$$

$$p_{O_2} \cdot V_g = n_{O_2} \cdot R \cdot T_g \tag{13.39}$$

$$p_{H_2O,g} \cdot V_g = n_{H_2O,g} \cdot R \cdot T_g \tag{13.40}$$

$$T_g = T_0 \tag{13.41}$$

$$\varepsilon_{pore} = \frac{V_{pore}}{V_{VC}} \tag{13.42}$$

$$\varepsilon_g = \frac{V_g}{V_{VC}} \tag{13.43}$$

$$V_{pore} = V_g + V_{H_2O,l} \tag{13.44}$$

$$\chi = \frac{m_{H_2O,l}}{m_s} \tag{13.45}$$

$$\chi_{\max} = \frac{\rho_{H_2O,l} \cdot V_{pore}}{m_s} \tag{13.46}$$

$$\chi_{rel} = \frac{\chi}{\chi_{\max}} \tag{13.47}$$

$$m_s = V_{VC} \cdot \rho_s \tag{13.48}$$

$$G_{H_2O,l} = V_{VC} \cdot \frac{\alpha_v \cdot \beta}{R \cdot T_g} \cdot \left(p_{H_2O,g} - p^{sat}_{H_2O}\right) \tag{13.49}$$

$$G_{H_2O,g} = -G_{H_2O,l} \tag{13.50}$$

$$p^{sat}_{H_2O} = p^{sat}_{0_{H_2O}} \cdot \exp\left[\left(\frac{1}{T^{sat}_0} - \frac{1}{T_g}\right)\frac{L_v \cdot M_{H_2O}}{R}\right] \tag{13.51}$$

$$L_v = 1.73287 \cdot 10^6 + 1.03001 \cdot 10^{-4} \cdot T_g - 4.47755 \cdot 10^1 \cdot T_g^2 + 7.6629 \cdot 10^{-2} \cdot T_g^3 - 5.5058 \cdot 10^{-5} \cdot T_g^4 \tag{13.52}$$

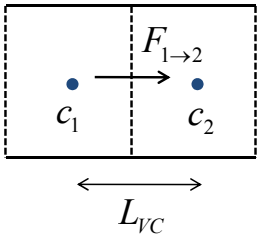


Figure 13.13: Diffusion of liquid water produced by concentration gradient.

An ordering of the model, with the variable to evaluate from each equation written within square brackets, is shown in Table 13.4.

Once the control volume is modeled, the next step is to describe the **diffusion of liquid and gas** between adjacent control volumes. This is described in detail below.

1. **Diffusion of liquid water**. The molar concentration of liquid water ($c_{H_2O,l}$) within a control volume is calculated by dividing the moles of liquid water ($n_{H_2O,l}$) contained in the control volume by the volume of the control volume (V_{VC}).

$$c_{H_2O,l} = \frac{n_{H_2O,l}}{V_{VC}} \tag{13.72}$$

   Consider a control plane with area S_{VC} that separates two control volumes with molar concentrations of water c_1 and c_2 (units: mol/m^3), respectively. The molar flow rate of liquid water ($F_{1\to2}$), through the control plane is (see Figure 13.13):

$$F_{1\to2} = S_{VC} \cdot D_{H_2O,l} \cdot \frac{c_1 - c_2}{L_{VC}} \tag{13.73}$$

   where $F_{1\to2}$ (units: mol/s) is positive while the flow goes from the control volume with concentration c_1 to the control volume with concentration c_2. The symbol $D_{H_2O,l}$ represents the diffusion coefficient (units: m^2/s).

2. **Diffusion of the gaseous species**. The following expression describes the mass transport due to ordinary diffusion and Knudsen diffusion of two gaseous species in a porous media

$$\nabla p_j = -\frac{R \cdot T_g}{\frac{\varepsilon_g}{\tau^2}} \cdot J_j \cdot \left(\frac{1}{D_{jK}} + \frac{1}{D_{ij}}\right) \qquad \text{for } j:1,2; \quad i \neq j \tag{13.74}$$

Table 13.4: Sorted model of the control volume.

$$[T_g] = T_0 \tag{13.53}$$

$$[L_v] = 1.73287 \cdot 10^6 + 1.03001 \cdot 10^{-4} \cdot T_g - 4.47755 \cdot 10^1 \cdot T_g^2 + 7.6629 \cdot 10^{-2} \cdot T_g^3 - 5.5058 \cdot 10^{-5} \cdot T_g^4 \tag{13.54}$$

$$n_{H_2O,l} = \frac{[m_{H_2O,l}]}{M_{H_2O}} \tag{13.55}$$

$$[V_{H_2O,l}] = \frac{m_{H_2O,l}}{\rho_{H_2O,l}} \tag{13.56}$$

$$\varepsilon_{pore} = \frac{[V_{pore}]}{V_{VC}} \tag{13.57}$$

$$V_{pore} = [V_g] + V_{H_2O,l} \tag{13.58}$$

$$[\varepsilon_g] = \frac{V_g}{V_{VC}} \tag{13.59}$$

$$[p_{O_2}] \cdot V_g = n_{O_2} \cdot R \cdot T_g \tag{13.60}$$

$$[p_{H_2O,g}] \cdot V_g = n_{H_2O,g} \cdot R \cdot T_g \tag{13.61}$$

$$[m_s] = V_{VC} \cdot \rho_s \tag{13.62}$$

$$[\chi] = \frac{m_{H_2O,l}}{m_s} \tag{13.63}$$

$$[\chi_{\max}] = \frac{\rho_{H_2O,l} \cdot V_{pore}}{m_s} \tag{13.64}$$

$$[\chi_{rel}] = \frac{\chi}{\chi_{\max}} \tag{13.65}$$

$$\left[p_{H_2O}^{sat}\right] = p_{0_{H_2O}}^{sat} \cdot \exp\left[\left(\frac{1}{T_0^{sat}} - \frac{1}{T_g}\right)\frac{L_v \cdot M_{H_2O}}{R}\right] \tag{13.66}$$

$$[G_{H_2O,l}] = V_{VC} \cdot \frac{\alpha_v \cdot \beta}{R \cdot T_g} \cdot \left(p_{H_2O,g} - p_{H_2O}^{sat}\right) \tag{13.67}$$

$$[G_{H_2O,g}] = -G_{H_2O,l} \tag{13.68}$$

$$\left[\frac{dn_{O_2}}{dt}\right] = F_{O_2} \tag{13.69}$$

$$\left[\frac{dn_{H_2O,g}}{dt}\right] = F_{H_2O,g} + G_{H_2O,g} \tag{13.70}$$

$$\left[\frac{dn_{H_2O,l}}{dt}\right] = F_{H_2O,l} + G_{H_2O,l} \tag{13.71}$$

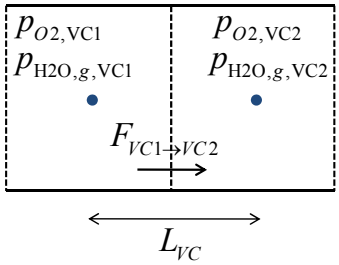


Figure 13.14: Diffusion of a binary gaseous mixture between adjacent control volumes.

where τ (units: unitless) is the tortuosity of porous medium; ε_g (units: unitless) is the fraction of volume that can be occupied by the gas, as described by Eq. (13.20); p_j (units: Pa) is the partial pressure of the j gaseous species; T_g (units: K) is the temperature of the gas mixture; J_j is the molar flux (units: mol/s/m^2) of the j gaseous species; D_{jK} (units: m^2/s) is the Knudsen diffusion coefficient of the j gaseous species; and D_{ij} (units: m^2/s) is the binary diffusion coefficient of the i and j gaseous species.

The temperature and pressure dependence of the D_{ij} diffusion coefficient can be described as shown in Eq. (13.75), where D_{ij} is the diffusion coefficient at the p_g pressure and the T_g temperature, provided that the value of the diffusion coefficient at the reference pressure and temperature (p_g^{ref}, T_g^{ref}) is D_{ij}^{ref}.

$$D_{ij} = D_{ij}^{ref} \frac{p_g}{p_g^{ref}} \left(\frac{T_g}{T_g^{ref}} \right)^{1.5} \tag{13.75}$$

Eq. (13.74) can be particularized to the case where the mixture is composed of oxygen and steam water:

$$\frac{dp_{O_2}}{dx} = -\frac{R \cdot T_g}{\frac{\varepsilon_g}{\tau^2}} \cdot J_{O_2} \cdot \left(\frac{1}{D_{O_2 K}} + \frac{1}{D_{O_2/H_2O,g}} \right) \tag{13.76}$$

$$\frac{dp_{H_2O,g}}{dx} = -\frac{R \cdot T_g}{\frac{\varepsilon_g}{\tau^2}} \cdot J_{H_2O,g} \cdot \left(\frac{1}{D_{H_2O,gK}} + \frac{1}{D_{O_2/H_2O,g}} \right) \tag{13.77}$$

where $D_{O_2/H_2O,g}$ is the binary diffusion coefficient of oxygen and steam water; and D_{O_2K} and $D_{H_2O,gK}$ are the Knudsen diffusion coefficients of oxygen and steam water, respectively.

Consider two adjacent control volumes, referred to as VC1 and VC2, as shown in Figure 13.14. The partial pressures of oxygen are represented as $p_{O2,VC1}$ and $p_{O2,VC2}$; and the partial pressures of steam water as $p_{H2O,g,VC1}$ and $p_{H2O,g,VC2}$. The molar flux of oxygen, represented as $F_{VC1 \to VC2,O2}$, is positive while the

diffusion flow of oxygen exits VC1 and enters VC2. The same criterion is applied to the molar flux of steam water, $F_{VC1\rightarrow VC2,H2O,g}$.

Spatial discretization of Eqs. (13.76) and (13.77) is carried out to calculate the molar flux (number of moles per unit area per unit time) by diffusion between adjacent control volumes. The following equations describe the diffusion of oxygen and steam water in a porous material.

$$p_{O2} = \max\left\{p_{O2,VC1}, p_{O2,VC2}\right\} \tag{13.78}$$

$$p_{H2O,g} = \max\left\{p_{H2O,g,VC1}, p_{H2O,g,VC2}\right\} \tag{13.79}$$

$$p_g = p_{O2} + p_{H2O,g} \tag{13.80}$$

$$T_g = T_0 \tag{13.81}$$

$$D_{O_2/H_2O,g} = D^{ref}_{O_2/H_2O,g} \cdot \frac{p_g}{p_g^{ref}} \cdot \left(\frac{T_g}{T_g^{ref}}\right)^{1.5} \tag{13.82}$$

$$\frac{p_{O_2,VC1} - p_{O_2,VC2}}{L_{VC}} = \frac{R \cdot T_g}{\frac{\varepsilon_g}{\tau^2}} \cdot J_{O_2} \cdot \left(\frac{1}{D_{O_2K}} + \frac{1}{D_{O_2/H_2O,g}}\right) \tag{13.83}$$

$$\frac{p_{H_2O,g,VC1} - p_{H_2O,g,VC2}}{L_{VC}} = \frac{R \cdot T_g}{\frac{\varepsilon_g}{\tau^2}} \cdot J_{H_2O,g} \cdot \left(\frac{1}{D_{H_2O,gK}} + \frac{1}{D_{O_2/H_2O,g}}\right) \tag{13.84}$$

$$F_{VC1\rightarrow VC2,O2} = J_{O_2} \cdot S_{VC} \tag{13.85}$$

$$F_{VC1\rightarrow VC2,H2O,g} = J_{H_2O,g} \cdot S_{VC} \tag{13.86}$$

Observe that, if the partial pressures in the control volumes are known variables (e.g., they are calculated from the control volume model), and the T_0, $D^{ref}_{O_2/H_2O,g}$, p_g^{ref}, T_g^{ref}, D_{O_2K}, $D_{H_2O,gK}$, L_{VC}, S_{VC}, R, ε_g and τ parameters have known values, then the p_{O2}, $p_{H2O,g}$, p_g, T_g, $D_{O_2/H_2O,g}$, J_{O_2}, $J_{H_2O,g}$, $F_{VC1\rightarrow VC2,O2}$ and $F_{VC1\rightarrow VC2,H2O,g}$ variables can be calculated, one after another, from Eqs. (13.78) – (13.86).

The **proposed task** consists in developing the Modelica models enumerated below.

1. The connector.
2. The control volume.
3. The species transport.
4. The boundary conditions for the diffusion layer model. This is, a model describing the evolution in time of the species pressure and concentrations.

Table 13.5: Parameters of the model defined in Task 2.

Quantity	**Symbol**	**Value**
Temperature	T_0	300 K
Molar mass of water	M_{H_2O}	18 gr/mol
Water density	$\rho_{H_2O,l}$	1000 kg/m^3
Solid density	ρ_s	727 kg/m^3
Ideal gas constant	R	8.31447 J/mol/K
Condensation area	α_v	10^-9 m^2
Transfer coefficient	β	0.001 m/s
Binary diffusion coefficient	$D_{O_2/H_2O,g}$	$2.82 \cdot 10^{-3}$ m^2/s
Knudsen diffusion coefficient of oxygen	D_{O_2K}	$7.853 \cdot 10^{-3}$ m^2/s
Knudsen diffusion coefficient of steam water	$D_{H_2O,gK}$	$1.047 \cdot 10^{-3}$ m^2/s
Diffusion coefficient of liquid water	$D_{H_2O,l}$	$2.5 \cdot 10^{-4}$ m^2/s
Porosity	ε_{pore}	0.2
Length of the layer	L	0.01 m
Tortuosity	τ	5
Medium area	S_{VC}	1 m^2
Saturation pressure of water T_0^{sat}	$p^{sat}_{0_{H_2O}}$	3169 Pa
Reference temperature for saturation pressure	T_0^{sat}	298.16 K

Table 13.6: Initial conditions used in Task 2.

Quantity	**Boundaries**	**Inside the layer**
Oxygen pressure	1 bar / 2 bars	0 bar
Steam water pressure	1 bar / 1 bar	0 bar
Liquid water load	0 / 0	0.03

Table 13.7: Boundary conditions used in Task 2.

Quantity	**Left end**	**Right end**
Oxygen pressure	1 bar	2 bars
Steam water pressure	1 bar	1 bar
Liquid water load	0	0

Instantiating and connecting the models previously defined, compose a model of a layer and its terminal boundary conditions. To this end, discretize the porous media into $N = 20$ control volumes. The model parameters are defined in Table 13.5. Notice that, as $S_{VC} = 1$ m^2, the molar flux is equal the molar flow rate.

Simulate the model of the diffusion layer to calculate the evolution of the gas species pressures, the liquid water load, and the molar flux of all species in each control volume. The initial and the boundary conditions are described in Tables 13.6 and 13.7. Plot the pressures of the gas species, the liquid water load, and the molar flux of all species at the control volumes number 1, 5, 10, 15, and 20.

13.6 Solution to Task 2

The connector is defined in Modelica Code 13.3, the control volume in Modelica Code 13.4, the diffusion transport in Modelica Code 13.5, and the boundary conditions and the complete model of the diffusion layer in Modelica Code 13.6.

The pressures of oxygen and steam water in the control volumes number 1, 5, 10, 15 and 20 are plotted in Figures 13.15 and 13.16. The liquid water load in the control volumes number 1, 5, 10, 15 and 20 is represented in Figure 13.17. The molar fluxes of steam water, liquid water and oxygen in the control volumes number 1, 5, 10, 15 and 20 are shown in Figures 13.18 – 13.20.

```
connector P_F2
  Modelica.SIunits.Temperature Tg;
  Modelica.SIunits.Pressure p_O2 "Pressure of oxygen";
  Modelica.SIunits.Pressure p_H2O_g "Pressure of steam water";
  flow Real n_H2O_l (unit="mol")   "Moles of liquid water";
  Real E_g "Portion of pores occupied by the gas";
  flow Real F_O2 (unit="mol/s") "Molar flow rate of oxygen";
  flow Real F_H2O_g (unit="mol/s") "Molar flow rate of steam water";
  flow Real F_H2O_l (unit="mol/s") "Molar flow rate of liquid water";
end P_F2;
```

Modelica Code 13.3: Diffusion of binary gas mixture and liquid in porous medium (1/4).

```
model VC2
  parameter Real M_H2O=0.018  (unit="kg/mol")   "Molar mass of water";
  parameter Modelica.SIunits.Density rho_H2O_l=1000  "Liquid water density";
  outer parameter Real R  (unit="J/K/mol")   "Ideal gases constant";
  outer parameter Modelica.SIunits.Density rho_s "Solid density";
  outer parameter Real Vvc  (unit="m3")   "Volume of the control volume";
  outer parameter Real Epores "Porosity of the medium";
  outer parameter Real To   (unit="K")    "Temperature";
  outer parameter Real alfa_v  (unit="m2")   "Condensation specific surface";
  outer parameter Real beta   (unit="m/s")   "Mass transfer coefficient";
  outer parameter Real To_sat  (unit="K")    "Saturation temperature";
  outer parameter Real P0_H2O_sat  (unit="Pa")
         "Water saturation pressure at To_sat";
  outer parameter Real Xo "Initial liquid water load";
  Real n_O2  (unit="mol")    "Moles of O2";
  Real n_H2O_g  (unit="mol")    "Moles of steam water";
  Real G_H2O_g  (unit="mol/s")    "Steam water generated";
  Real G_H2O_l  (unit="mol/s")    "Liquid water generated";
  Real m_H2O_l  (unit="kg")   "Mass of steam water";
  Real m_s  (unit="kg")    "Mass of the solid";
  Real V_H2O_l  (unit="m3")    "Volume of liquid water";
  Real V_g  (unit="m3")   "Volume of the gas";
  Real Vpores  (unit="m3")    "Volume of pores";
  Real X "Liquid water load";
  Real Xmax "Maximum liquid water load";
  Real Xrel "Relative liquid water load";
  Real Lv  (unit="J/mol")   "Evaporation molar enthalpy";
  Real P_H2O_sat  (unit="Pa")    "Water saturation pressure";

  P_F2 p_f;
equation
  p_f.F_O2 = der(n_O2);
  p_f.F_H2O_g + G_H2O_g = der(n_H2O_g);
  p_f.F_H2O_l + G_H2O_l = der(p_f.n_H2O_l);
  p_f.n_H2O_l = m_H2O_l/M_H2O;
  V_H2O_l = m_H2O_l/rho_H2O_l;
  p_f.p_O2*V_g = n_O2*R*p_f.Tg;
  p_f.p_H2O_g*V_g = n_H2O_g*R*p_f.Tg;
  p_f.Tg = To;
  Epores = Vpores/Vvc;
  Vpores = V_g + V_H2O_l;
  p_f.E_g = V_g/Vvc;
  X = m_H2O_l/m_s;
  Xmax = rho_H2O_l*Vpores/m_s;
  Xrel = X/Xmax;
  m_s = Vvc*rho_s;
  G_H2O_l =  if  p_f.p_H2O_g >  0 or  X >  0 then  Vvc*alfa_v*beta*(p_f.p_H2O_g -
P_H2O_sat)/(R*p_f.Tg) else 0;
  G_H2O_g = -G_H2O_l;
  P_H2O_sat = P0_H2O_sat*exp(((1/To_sat) - (1/p_f.Tg))*Lv*M_H2O/R);
  Lv =  1.73287e6 +  1.03001e-4*p_f.Tg -  4.47755e1*p_f.Tg^2 +7.6629e-2*p_f.Tg^3 -
5.5058e-5*p_f.Tg^4;
initial equation
  X = Xo;
end VC2;
```

Modelica Code 13.4: Diffusion of binary gas mixture and liquid in porous medium (2/4).

```
model FT2
  Real pO2  (unit="Pa")    "Partial pressure of oxygen";
  Real pH2O_g  (unit="Pa")    "Partial pressure of steam water";
  Real p_g  (unit="Pa")    "Total pressure of the gases";
  Real J_O2  (unit="mol/s/m2")   "Flux per area unit of oxygen";
  Real J_H2O_g  (unit="mol/s/m2")   "Flux per area unit of steam water";
  Real D_O2_H2O_g  (unit="m2/s")   "Binary diffusion coefficient";
  Real Egmin "Free liquid portion of pores";

  outer parameter Real D_O2_H2O_g_ref  (unit="m2/s")
         "Binary diffusion coefficient";
  outer parameter Real p_g_ref  (unit="Pa")
         "Reference pressure for binary diffusion";
  outer parameter Real Tg_ref  (unit="K")
         "Reference temperature for binary diffusion";
  outer parameter Real To  (unit="K")    "Temperature of medium";
  outer parameter Real D_H2O_g_k  (unit="m2/s")
         "Knudsen diffusion coefficient of steam water";
  outer parameter Real D_O2_k  (unit="m2/s")
         "Knudsen diffusion coefficient of oxygen";
  outer parameter Real D_H2O_l  (unit="m2/s")
         "Diffusion coefficient of liquid water";
  outer parameter Real tau "Tortuosity";
  outer parameter Real Svc  (unit="m2") "Control volume area";
  outer parameter Real R  (unit="J/K/mol")    "Ideal gas constant";
  parameter Real Lvc  (unit="J/mol")    "Evaporation molar enthalpy";

  P_F2 p_f_1;
  P_F2 p_f_2;
equation
  p_f_1.F_H2O_l = D_H2O_l*(p_f_1.n_H2O_l - p_f_2.n_H2O_l)/(Lvc^2);
  pO2 = max(p_f_1.p_O2, p_f_2.p_O2);
  pH2O_g = max(p_f_1.p_H2O_g, p_f_2.p_H2O_g);
  p_g = pO2 + pH2O_g;
  D_O2_H2O_g = D_O2_H2O_g_ref*(p_g/p_g_ref)*(p_f_2.Tg/Tg_ref)^1.5;
  Egmin = max(0, min(p_f_1.E_g, p_f_2.E_g));
  J_O2 = if (p_g > 0)
     then (Egmin/tau^2)*(p_f_1.p_O2 - p_f_2.p_O2)/(Lvc*R*p_f_2.Tg*((1/D_O2_k)
          + (1/D_O2_H2O_g))) else 0;
  J_H2O_g = if (p_g > 0)
      then (Egmin/tau^2)*(p_f_1.p_H2O_g - p_f_2.p_H2O_g)/(Lvc*R*p_f_2.Tg*((1/D_O2_k)
        + (  1/D_O2_H2O_g))) else 0;
  p_f_2.F_O2 = -J_O2*Svc;
  p_f_2.F_H2O_g = -J_H2O_g*Svc;
  p_f_1.F_O2 = -p_f_2.F_O2;
  p_f_1.F_H2O_g = -p_f_2.F_H2O_g;
  p_f_1.F_H2O_l = -p_f_2.F_H2O_l;
end FT2;
```

Modelica Code 13.5: Diffusion of binary gas mixture and liquid in porous medium (3/4).

```
model Source_P2
  parameter Real presion_O2  (unit="Pa")
        "Boundary condition of oxygen pressure";
  parameter Real presion_H2O_g  (unit="Pa")
        "Boundary condition of steam water pressure";
  outer parameter Real To  (unit="K")    "Boundary condition of temperature";

  P_F2 p_f;
equation
  p_f.p_O2 = presion_O2;
  p_f.p_H2O_g = presion_H2O_g;
  p_f.n_H2O_l = 0;
  p_f.Tg = To;
  p_f.E_g = 1;
end Source_P2;


model Diffusion_Layer2
  parameter Integer N=20;
  parameter Real L=0.01  (unit="m")   "Length of the layer";
  parameter Real l=L/(N)  (unit="m")    "Length of the control volume";
  inner parameter Real Svc=1 "Medium area";
  inner parameter Real Vvc=Svc*l "Volume of the control volume";
  inner parameter Real To=300 "Temperature of the medium";
  inner parameter Real rho_s=727 "Density of the solid";
  inner parameter Real R=8.31447 "Ideal gas constant";
  inner parameter Real alfa_v=1e-9 "Condensation specific area";
  inner parameter Real beta=0.001 "Mass transfer coefficient";
  inner parameter Real D_H2O_g_k=1.047e-3 "Knudsen diff. coeff. steam water";
  inner parameter Real D_O2_k=7.853e-3 "Knudsen diff. coeff. of oxygen";
  inner parameter Real D_O2_H2O_g_ref=2.82e-3 "Binary diffusion coefficient";
  inner parameter Real p_g_ref=1e5 "Reference pressure of binary diffusion";
  inner parameter Real Tg_ref=308.1 "Reference pressure of binary diffusion";
  inner parameter Real D_H2O_l=2.5e-4 "Diffusion coefficient of liquid water";
  inner parameter Real Epores=0.2 "Porosity of the medium";
  inner parameter Real tau=5 "Tortuosity";
  inner parameter Real To_sat=298.16 "Saturation temperature";
  inner parameter Real P0_H2O_sat=3169 "Saturation pressure of water at To_sat";
  inner parameter Real Xo=0.03 "Initial liquid water load";

  parameter Real lv_ft[N + 1]={if i == 1 or i == (N + 1)
        then l/2 else l for i in 1:(N + 1)};

  VC2 v_c[N];
  FT2 f_t[N + 1](Lvc=lv_ft);
  Source_P2 p1(presion_O2=1e5, presion_H2O_g=1e5);
  Source_P2 p2(presion_O2=2e5, presion_H2O_g=1e5);
equation
  connect(p1.p_f, f_t[1].p_f_1);
  for conta in 1:N loop
    connect(f_t[conta].p_f_2, v_c[conta].p_f);
    connect(v_c[conta].p_f, f_t[conta + 1].p_f_1);
  end for;
  connect(f_t[N + 1].p_f_2, p2.p_f);
end Diffusion_Layer2;
```

Modelica Code 13.6: Diffusion of binary gas mixture and liquid in porous medium (4/4).

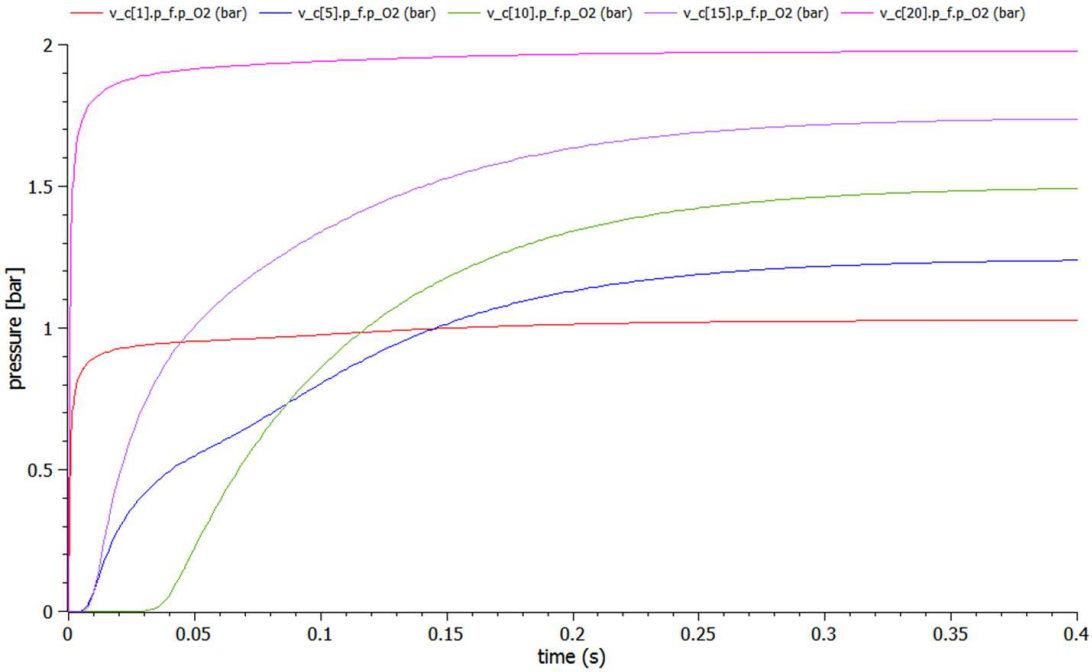


Figure 13.15: Pressure of oxygen in the control volumes number 1, 5, 10, 15 and 20.

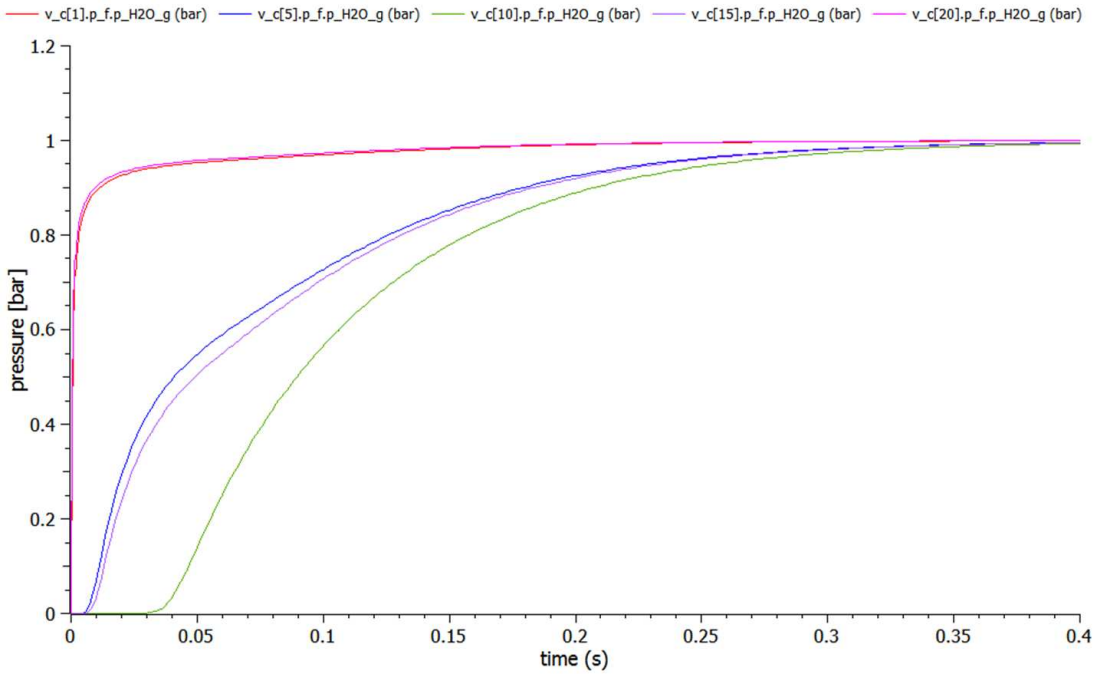


Figure 13.16: Pressure of steam water in the control volumes number 1, 5, 10, 15 and 20.

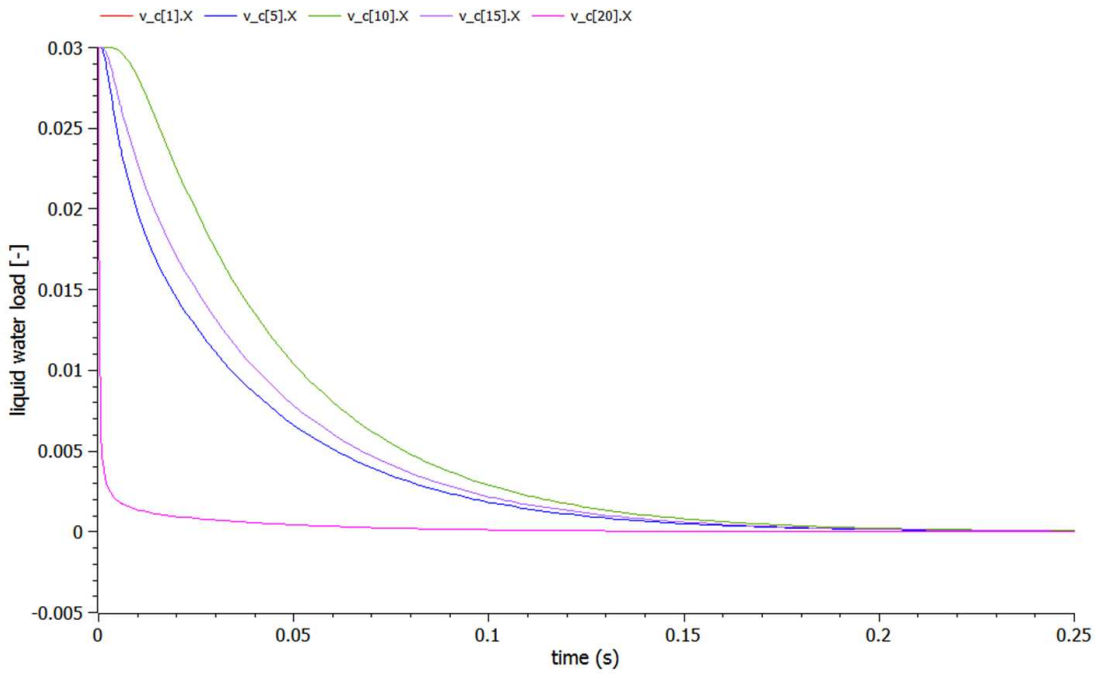


Figure 13.17: Liquid water load in the control volumes number 1, 5, 10, 15 and 20.

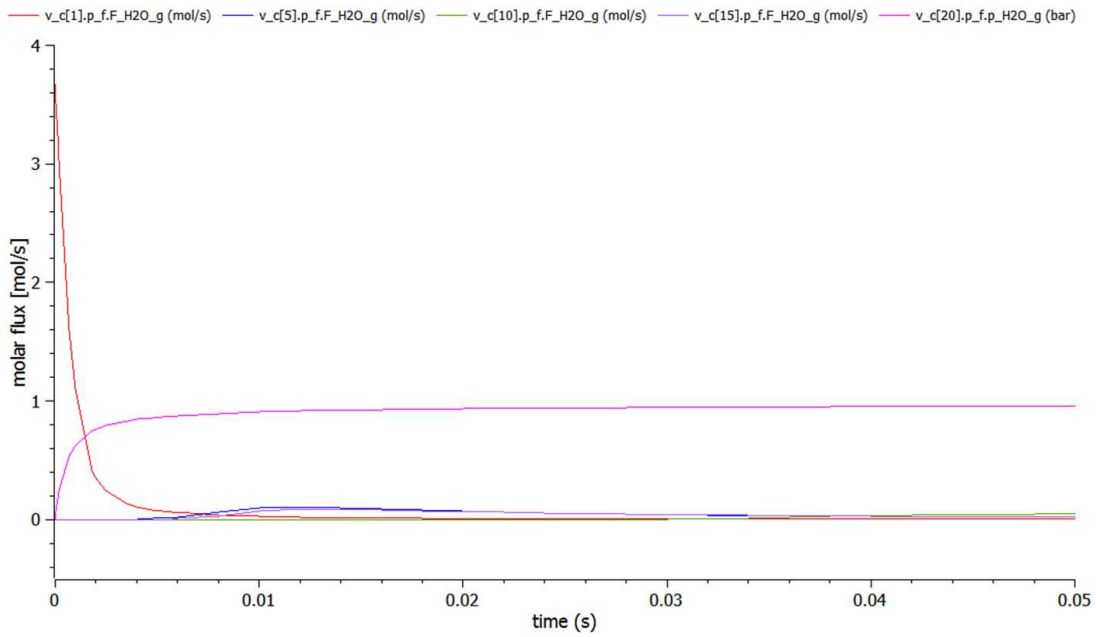


Figure 13.18: Molar flux of steam water in the control volumes number 1, 5, 10, 15 and 20.

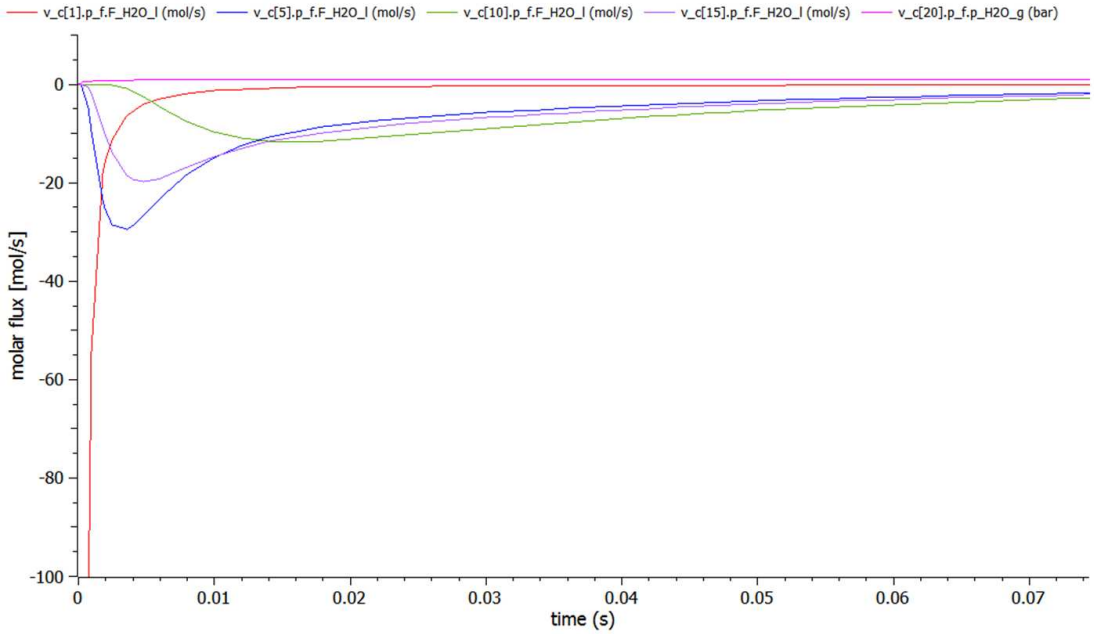


Figure 13.19: Molar flux of liquid water in the control volumes number 1, 5, 10, 15 and 20.

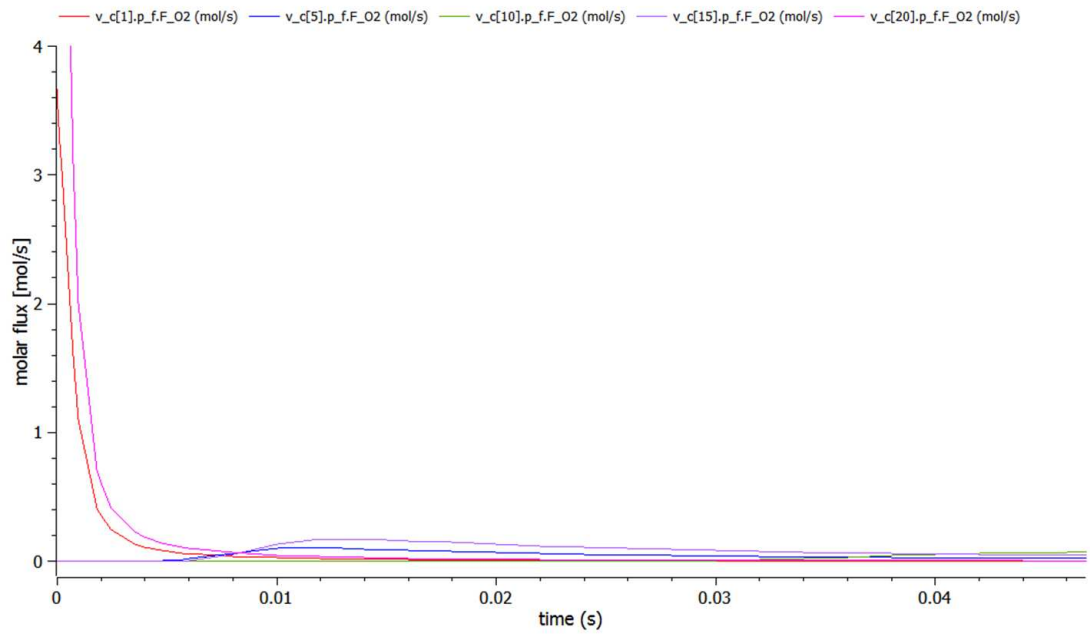


Figure 13.20: Molar flux of oxygen in the control volumes number 1, 5, 10, 15 and 20.

13.7 Task 3

To recap on what has been discussed so far, the most relevant phenomena in each layer of the fuel cell are enumerated below (see again Table 13.1).

- *Membrane*. Water diffusion in liquid and steam phases, and protonic conduction.
- *Catalyst layer of the cathode*. Water diffusion in liquid and steam phases, oxygen diffusion, electronic and protonic conduction, and electrochemical reaction.
- *Diffusion layer of the cathode*. Water diffusion in liquid and steam phases, oxygen diffusion, and electronic conduction.

The diffusion of water in liquid and steam phases, and the diffusion of oxygen, as well as the molar balance of these species in a control volume, were modeled in Task 2. The objective now is to develop the following models.

1. *Electronic and protonic conduction*. These models, used in conjunction with the models developed in Task 2, will allow to describe the behavior of the membrane and the diffusion layer.
2. *Electrochemical reaction that takes place in the catalyst layer*. This new model is coupled to the transport model proposed in Task 2. On the one hand, the electrochemical reaction rate depends on the oxygen partial pressure. On the other hand, the oxygen consumed in the reaction and the generated steam water intervene in the molar balances of these species.
3. *Membrane, and catalyst layer and diffusion layer of the cathode*. Each layer will be discretized into control volumes. A component describing the transport phenomena will be connected between adjacent control volumes, as represented in Figure 13.8.
4. *Electrical interfaces of membrane and cathode*. These interface models will allow to connect the three-layer model to external electric components.
5. *Complete PEM fuel cell*. It will be composed by connecting previously defined models, as shown in Figure 13.21.
6. *PEMFC connected to external load resistance*. This electrical circuit (see Figure 13.22) will allow to simulate the polarization curves of the fuel cell.

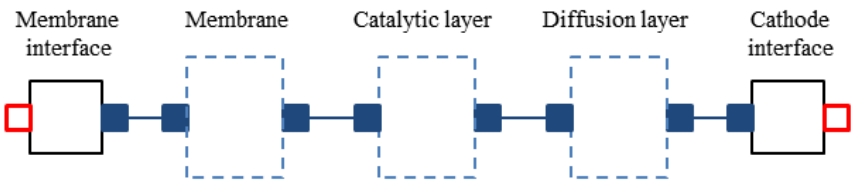


Figure 13.21: Electrical model of the fuel cell.

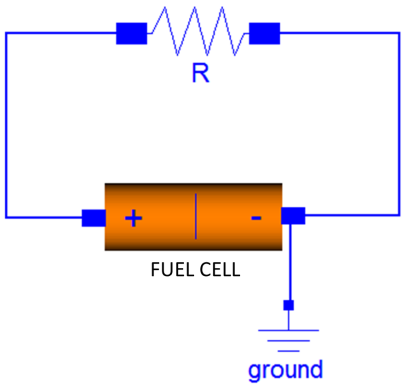


Figure 13.22: PEMFC model connected to external load resistance.

These new models are explained in detail below.

1. **Electronic and protonic transport**. Electrons are transported through the solid material, and protons through the electrolyte. In general, the voltages in the solid material and the electrolyte are different, and depend on the spatial coordinate. These voltages are represented as v_e and v_p, respectively. The e subscript refers to the electronic conductor material and the p subscript refers to the protonic conductor material.

   The protonic current density (j_p) is proportional to the voltage gradient in the protonic conductor medium as described in Eq. (13.87), where K_p is the protonic conductivity and ε_m is the volume of the protonic conductor.

$$j_p = -K_p \cdot \varepsilon_m \cdot \frac{dv_p}{dx} \tag{13.87}$$

   As described in Eq. (13.88), the electronic current density (j_e) is proportional to the voltage gradient in the solid material, where σ_e is the electronic conductivity in the solid medium and ε_s is its porosity.

$$j_e = -\sigma_e \cdot \varepsilon_s \cdot \frac{dv_e}{dx} \tag{13.88}$$

Table 13.8: Conductivity parameters of the layers ($\epsilon \approx 0$).

	Membrane	**Catalyst layer**	**Diffusion layer**
Protonic conductivity	$K_{p,mem}$	$K_{p,cat}$	ϵ
Electronic conductivity	ϵ	$\sigma_{e,cat}$	$\sigma_{e,dif}$

The following two assumptions are made:

a) The membrane is composed of a polymeric material that conducts protons, but does not conduct electrons.

b) The diffusion layer is a matrix of electrically conductive porous material that does not transport protons.

The conductivity parameters of the layers are shown in Table 13.8. The ϵ conductivity represents a small value near zero.

Eqs. (13.87) and (13.88) allow to calculate the current density of protons and electrons through the control plane that separates two adjacent control volumes named 1 and 2.

$$j_p = K_p \cdot \varepsilon_m \cdot \frac{v_{p,1} - v_{p,2}}{L_{VC}} \tag{13.89}$$

$$j_e = \sigma_e \cdot \varepsilon_s \cdot \frac{v_{e,1} - v_{e,2}}{L_{VC}} \tag{13.90}$$

The currents (i_p, i_e) are calculated taking into account the cross-sectional area of the control plane (S_{VC}). They are positive while exit control volume 1 and enter control volume 2.

$$\frac{i_p}{S_{VC}} = K_p \cdot \varepsilon_m \cdot \frac{v_{p,1} - v_{p,2}}{L_{VC}} \tag{13.91}$$

$$\frac{i_e}{S_{VC}} = \sigma_e \cdot \varepsilon_s \cdot \frac{v_{e,1} - v_{e,2}}{L_{VC}} \tag{13.92}$$

2. **Control volume**. This model describes the balances of electrons and protons, and the generation due to the electrochemical reaction. Additional hypotheses are made to describe the voltages of the electronic and protonic conductors.

   a) **Balances of electrons and protons**. It is assumed that there is no accumulation of electrons and protons in the control volume. Therefore,

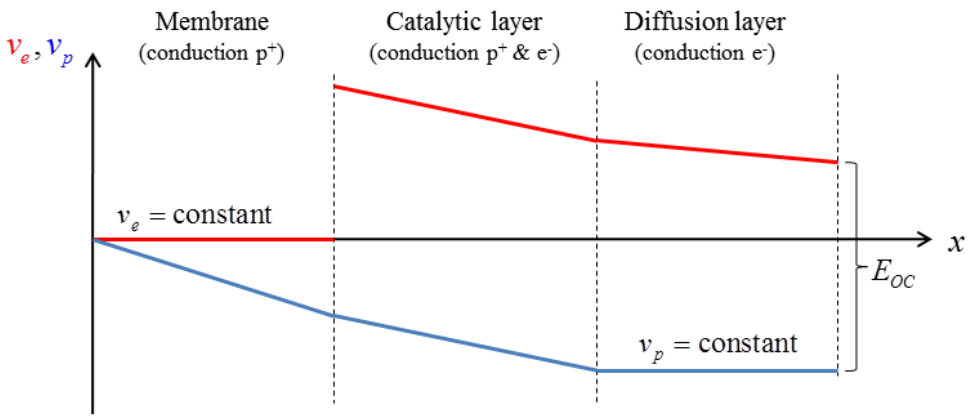


Figure 13.23: Solid voltage (v_e) and electrolyte voltage (v_p), not drawn to scale.

the net electronic current that enters through the control planes, plus the electrons generated in the control volume per time unit, is zero. The same applies to the net protonic current and generation inside the control volume.

$$0 = i_e + G_e \tag{13.93}$$
$$0 = i_p + G_p \tag{13.94}$$

As there is no electrochemical reaction in the membrane and diffuser layer, the electron or proton generation in the control volumes of these layers is zero ($G_e = G_p = 0$). This implies zero net incoming protonic and electronic currents to these control volumes ($0 = i_e = i_p$).

b) **Solid and electrolyte voltages**. The model should make it possible to calculate, for each control volume, the voltage of the electronic conductor (v_e) and protonic conductor (v_p). The electron and proton balances formulated above are used for this purpose. The following assumptions are also made (see Figure 13.23).

1) The voltage for electronic conduction (v_e) is constant along the membrane.
2) The voltage for protonic conduction (v_p) is constant along the diffusion layer.
3) In the external interface of the membrane (left-hand end of Figure 13.23), the voltage of the electronic conductor is equal to the voltage of the protonic conductor.

$$v_e = v_p \tag{13.95}$$

4) In the external interface of the diffusion layer (right-hand end of Figure 13.23), it is satisfied:

$$v_e - v_p = E_{OC} \tag{13.96}$$

c) **Electrochemical reaction**. The following electrochemical reaction takes place inside the catalyst layer of the cathode:

$$4H^+ + O_2 + 4e^- \longleftrightarrow 2H_2O \tag{13.97}$$

The electrons produced per unit of time can be calculated as shown in Eq. (13.98), where a_{act} is the specific area of the catalyst layer, and B the Tafel slope. The reference partial pressure of oxygen is represented as $p^0_{O_2}$. The number of protons and electrons generated per unit of time are equal, as described in Eq. (13.99).

$$G_e = -a_{act} \cdot i_0^{ref} \cdot \frac{p_{O_2}}{p^0_{O_2}} \cdot e^{\frac{E_{OC}-v_e+v_p}{B}} \tag{13.98}$$

$$G_p = G_e \tag{13.99}$$

The moles of oxygen (G_{reac,O_2}) and steam water ($G_{reac,H_2O,g}$) generated per unit of time are related with the electrons produced per unit of time as described below, where F represents the Faraday constant (96.485 C/mol).

$$G_{reac,O_2} = \frac{1}{4 \cdot F} \cdot G_e \tag{13.100}$$

$$G_{reac,H_2O,g} = -\frac{1}{2 \cdot F} \cdot G_e \tag{13.101}$$

d) **Molar balances of oxygen and steam water**. The balances, taking into account the component of generation due to the electrochemical reaction, are written as follows.

$$\frac{dn_{O_2}}{dt} = F_{O_2} + G_{reac,O_2} \tag{13.102}$$

$$\frac{dn_{H_2O,g}}{dt} = F_{H_2O,g} + G_{H_2O,g} + G_{reac,H_2O,g} \tag{13.103}$$

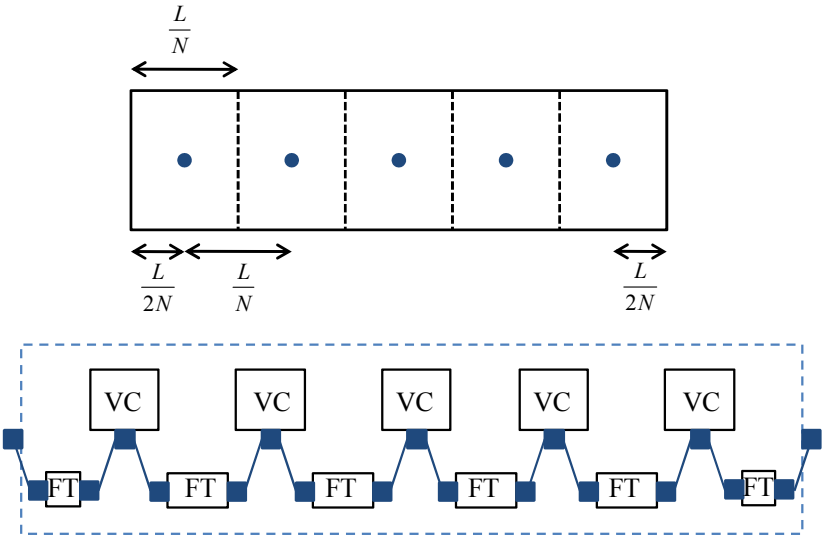


Figure 13.24: Spatial discretization and layer model.

3. **Model of the PEM fuel cell**. The model diagram is shown in Figure 13.21. It is composed of the layer models (membrane, catalyst and diffusion layers), and the interface models of membrane and cathode.

   a) **Connectors**. Two classes of connectors have been used in the model shown in Figure 13.21.

   – *Standard electrical connector of Modelica*. The connector represented as a empty square is the standard electrical connector of Modelica, which defines an across variable (voltage, v) and a through variable (current, i).

   – *Connector of layer models*. The connector represented as a filled square describes the solid and electrolyte voltages (v_e and v_p), the electronic and protonic currents (i_e and i_p), the partial pressures of oxygen and steam water, the load of liquid water, the molar flow rates of oxygen, steam water and liquid water, and the portion of pores occupied by the gas.

   The through variables (currents and molar flow rates) have positive sign while enter into the component.

   b) **Layer models**. The layers are discretized following the same procedure as in the previous tasks. An example of spatial discretization of a layer is shown in the upper part of Figure 13.24. The layer, of length L, is divided into $N = 5$ control volumes of equal size. The corresponding model, shown in the lower part of the figure, is composed of the connection of control volumes (labeled as VC) and transport phenomena (labeled as FT).

c) **Membrane and cathode interfaces**. These models describe the relationships among the terminal variables of the Modelica electrical connector (v, i) and the layer model connector (v_e, v_p, i_e, i_p).

- *Membrane interface* (see Figure 13.23):

$$v = v_e \tag{13.104}$$
$$v_e = v_p \tag{13.105}$$
$$i + i_e + i_p = 0 \tag{13.106}$$

- *Cathode interface*:

$$v = v_e \tag{13.107}$$
$$v_e - v_p = E_{OC} \tag{13.108}$$
$$i + i_e + i_p = 0 \tag{13.109}$$

In addition, these interface models impose the boundary conditions on the partial pressure of oxygen, and the molar flow rates of liquid and steam water.

The proposed **modeling tasks** consist in developing the Modelica models enumerated below.

1. The connector of the layer models.
2. Control volumes of the membrane, catalyst and diffusion layers.
3. Transport phenomena in the membrane, catalyst and diffusion layers. As oxygen diffusion is not considered in the membrane, there will be no binary gas diffusion in this layer.
4. Membrane, catalyst and diffusion layers. Discretize each layer into 20 equal control volumes.
5. Interface models to connect the membrane and the diffusion layer to external electrical components, which have the standard electrical connectors of Modelica. These interfaces also impose boundary conditions to the partial pressure of oxygen, and the molar fluxes of steam and liquid water. See Tables 13.9 and 13.10.

Table 13.9: Boundary conditions at the end of the membrane.

Quantity	**Symbol**	**Value**
Partial pressure of oxygen	p_{O_2}	1 Pa
Steam water flux	$j_{H_2O,g}$	0 $\text{mol}\cdot\text{m}^{-2}\cdot\text{s}^{-1}$
Liquid water flux	$j_{H_2O,l}$	0 $\text{mol}\cdot\text{m}^{-2}\cdot\text{s}^{-1}$

Table 13.10: Boundary conditions at the end of the catalyst layer.

Quantity	**Symbol**	**Value**
Partial pressure of oxygen	p_{O_2}	$1 \cdot 10^5$ Pa
Molar flux of steam water	$j_{H_2O,g}$	0 $\text{mol}\cdot\text{m}^{-2}\cdot\text{s}^{-1}$
Molar flux of liquid water	$j_{H_2O,l}$	0 $\text{mol}\cdot\text{m}^{-2}\cdot\text{s}^{-1}$

Table 13.11: Initial value of state variables in connectors.

Quantity	**Symbol**	**Value**
Pressure of oxygen	p_{O_2}	$1 \cdot 10^5$ Pa
Pressure of steam water	$p_{H_2O,g}$	$1 \cdot 10^3$ Pa
Load of liquid water	χ	$1 \cdot 10^{-2}$ $\text{kg}_{H_2O}/\text{kg}_{solid}$
Electronic voltage	v_e	0.5 V
Protonic voltage	v_p	-0.5 V

6. Complete model of the fuel cell. Suggested initial values for the state variables of the connectors are given in Table 13.11. The parameter values are specified in Table 13.12 (catalyst layer), Table 13.13 (diffusion layer) and Table 13.14 (membrane).

7. Connect the fuel cell model in series with an electrical resistor. Select the membrane interface as the ground node. The circuit diagram is shown in Figure 13.22.

The following three **simulation tasks** are proposed.

1. Simulate the *polarization curve* of the fuel cell to analyze the effect of the parameters shown in Table 13.15 on the electrical behavior of the fuel cell.

   The polarization curve, also known as I/V curve, is obtained by simulating the fuel cell voltage and current for different values of the external resistance. The polarization curve should show the maximum current range. In our case, the curve must reach a minimum voltage value of $0.3 - 0.4$ V. The value of each point of the polarization curve (voltage and current) will be obtained after simulating during 10 s.

2. When the current produced by the fuel cell and the oxygen consumption are high, the phenomenon known as *mass transport defect* represents a problem in the fuel cell operation. When it occurs, the oxygen pressure drops significantly in the catalytic layer, causing the voltage and current to drop.

   To study this phenomenon, analyze the effect of the design parameters listed in Table 13.16 on the oxygen pressure along the spatial dimension perpendicular to the catalyst layer, once the stationary state is reached for a high current.

3. Another relevant phenomenon is the *flooding of the fuel cell cathode*. In this case, the liquid water generated by the electrochemical reaction is accumulated in the porous material, making difficult for the oxygen to enter the catalyst layer. This phenomenon produces a mass defect that decreases the fuel cell performance.

   To study this phenomenon, analyze the effect of the parameters shown in Table 13.17 on the dynamics of the flooding process. Plot the time evolution of the fuel cell voltage, under high-current operating conditions.

Table 13.12: Parameters of the catalyst layer.

Quantity	**Symbol**	**Value**
Temperature	T_g	340 K
Total length of the layer	L	$4 \cdot 10^{-5}$ m
Tortuosity	τ	5
Area of the control volume	S_{VC}	1 m^2
Molar mass of water	M_{H_2O}	0.018 kg/mol
Liquid water density	$\rho_{H_2O,l}$	972 kg/m^3
Solid density	ρ_s	4000 kg/m^3
Electrolyte density	ρ_e	2000 kg/m^3
Ideal gas constant	R	8.31447 J/mol/K
Condensation specific surface	α_v	10^{-2} m^2
Mass transfer coefficient	β	0.001 m/s
Binary diffusion coefficient	$D_{O_2/H_2O,g}$	$2.82 \cdot 10^{-3}$ m^2/s
Reference pressure for binary diffusion	p_g^{ref}	$1 \cdot 10^5$ Pa
Reference temperature for binary diffusion	T_g^{ref}	308.1 K
Diffusion coefficient of liquid water	$D_{H_2O,l}$	$2.5 \cdot 10^{-11}$ m^2/s
Knudsen diffusion coefficient of oxygen	D_{O_2K}	$7.853 \cdot 10^{-3}$ m^2/s
Knudsen diffusion coefficient of steam water	$D_{H_2O,gK}$	$1.047 \cdot 10^{-3}$ m^2/s
Volume rate of the solid	ε_s	0.6
Volume rate of the electrolyte	ε_e	0.2
Water saturation pressure at T_0^{sat}	$p_{0_{H_2O}}^{sat}$	3169 Pa
Saturation temperature	T_0^{sat}	298.16 K
Specific area of catalyst $\times$ Ref. exchange current	$a_{act} \cdot i_0^{ref}$	1.2 A/m^3
Tafel slope	B	0.04 V
Reference partial pressure of oxygen	$p_{O_2}^0$	$1 \cdot 10^5$ Pa
Solid conductivity	$\sigma_{e,cat}$	1 S·m^{-1}
Protonic conductivity of electrolyte	$K_{p,cat}$	0.1 S·m^{-1}

Table 13.13: Parameters of the diffusion layer.

Quantity	**Symbol**	**Value**
Temperature	T_g	340 K
Total length of the layer	L	$1.6 \cdot 10^{-3}$ m
Tortuosity	τ	1
Area of the control volume	S_{VC}	1 m^2
Molar mass of water	M_{H_2O}	0.018 kg/mol
Liquid water density	$\rho_{H_2O,l}$	972 kg/m^3
Solid density	ρ_s	4000 kg/m^3
Ideal gas constant	R	8.31447 J/mol/K
Condensation specific surface	α_v	10^{-2} m^2
Mass transfer coefficient	β	0.001 m/s
Binary diffusion coefficient	$D_{O_2/H_2O,g}$	$2.82 \cdot 10^{-3}$ m^2/s
Reference pressure for binary diffusion	p_g^{ref}	$1 \cdot 10^5$ Pa
Reference temperature for binary diffusion	T_g^{ref}	308.1 K
Diffusion coefficient of liquid water	$D_{H_2O,l}$	$3.5 \cdot 10^{-11}$ m^2/s
Knudsen diffusion coefficient of oxygen	D_{O_2K}	$7.853 \cdot 10^{-3}$ m^2/s
Knudsen diffusion coefficient of steam water	$D_{H_2O,gK}$	$1.047 \cdot 10^{-3}$ m^2/s
Volume rate of the solid	ε_s	0.4
Water saturation pressure at T_0^{sat}	$p_{0_{H_2O}}^{sat}$	3169 Pa
Saturation temperature	T_0^{sat}	298.16 K
Solid conductivity	$\sigma_{e,dif}$	$1{\cdot}10^4$ $S{\cdot}m^{-1}$
Protonic conductivity of electrolyte	$K_{p,dif}$	$1{\cdot}10^{-7}$ $S{\cdot}m^{-1}$

Table 13.14: Parameters of the membrane.

Quantity	**Symbol**	**Value**
Temperature	T_g	340 K
Total length of the layer	L	$8 \cdot 10^{-5}$ m
Tortuosity	τ	1
Area of the control volume	S_{VC}	1 m^2
Molar mass of water	M_{H_2O}	0.018 kg/mol
Liquid water density	$\rho_{H_2O,l}$	972 kg/m^3
Electrolyte density	ρ_e	2000 kg/m^3
Ideal gas constant	R	8.31447 J/mol/K
Condensation specific surface	α_v	10^{-2} m^2
Mass transfer coefficient	β	0.001 m/s
Diffusion coefficient of liquid water	$D_{H_2O,l}$	$5.5 \cdot 10^{-6}$ m^2/s
Knudsen diffusion coefficient of steam water	$D_{H_2O,gK}$	$1 \cdot 10^{-6}$ m^2/s
Volume rate of the electrolyte	ε_e	0.72
Water saturation pressure at T_0^{sat}	$p_{0_{H_2O}}^{sat}$	3169 Pa
Saturation temperature	T_0^{sat}	298.16 K
Solid conductivity	$\sigma_{e,mem}$	$1{\cdot}10^{-6}$ S·m^{-1}
Protonic conductivity of electrolyte	$K_{p,mem}$	10 S·m^{-1}

Table 13.15: Parameters to analyze in the Simulation Task 1.

Quantity	**Symbol**	**Layer**
Layer length	L	3 layers
Cat. area × exchange current	$a_{act} \cdot i_0^{ref}$	Catalyst layer
Tafel slope	B	Catalyst layer
Conductivity of the solid	σ_e	Catalyst and diffusion layers
Protonic conductivity	K_p	Catalyst layer and membrane

Table 13.16: Parameters to analyze in the Simulation Task 2.

Quantity	**Symbol**	**Layer**
Layer length	L	Catalyst layer
Oxygen pressure	p_{O_2}	Cathode interface
Porosity of the solid	ε_s	Catalyst layer
Electrolyte porosity	ε_e	Catalyst layer
Binary diffusion coeff.	$D_{O_2/H_2O,g}$	Catalyst layer
Knudsen diffusion coeff. of oxygen	D_{O_2K}	Catalyst layer
Knudsen diffusion coeff. of steam water	$D_{H_2O,gK}$	Catalyst layer

Table 13.17: Parameters to analyze in the Simulation Task 3.

Quantity	**Symbol**	**Layer**
Layer length	L	3 layers
Porosity of the solid	ε_s	Diffusion layer
Temperature	T_g	Complete cell

13.8 Solution to Task 3

The proposed modeling tanks are solved in Modelica Code 13.7 – 13.28. The results of the three simulation tasks are plotted in Figures 13.25 – 13.45.

```
connector P_F3
  Modelica.SIunits.Pressure p_O2(start=1e5) "Pressure of oxygen";
  Modelica.SIunits.Pressure p_H2O_g(start=1e3) "Pressure of steam water";
  Real X(start=1e-2) "Liquid water load";
  Modelica.SIunits.Voltage ve(start=0.5) "Solid voltage";
  Modelica.SIunits.Voltage vp(start=-0.5) "Electrolyte voltage";
  flow Real F_O2(unit="mol/s") "Molar flow rate of oxygen";
  flow Real F_H2O_g(unit="mol/s") "Molar flow rate of steam water";
  flow Real F_H2O_l(unit="mol/s") "Molar flow rate of liquid water";
  flow Modelica.SIunits.Current ie "Electronic current";
  flow Modelica.SIunits.Current ip "Protonic current";

  Real E_g "Portion of pores occupied by the gas";
end P_F3;
```

Modelica Code 13.7: PEM fuel cell (1/22). Connector.

```
model VC_m
  Connectors.P_F3 p;

  parameter Real Vvc = Svc * Lvc  (unit="m3")  "Volume of the control volume";
  parameter Real M_H2O = 0.018  (unit="kg/mol")  "Molar mass of water";
  parameter Real rho_H2O_l = 972  (unit="kg/m3")  "Liquid water density";
  parameter Real Lvc  (unit="m")  "Length of the control volume";
  parameter Real P0_H2O_sat = 3169  (unit="Pa")
        "Water saturation pressure at To_sat";
  parameter Real To_sat = 298.16  (unit="K")  "Saturation temperature";
  parameter Real alfa_v = 1e3  (unit="m2")  "Condensation specific surface";
  parameter Real beta = 0.001  (unit="m/s")  "Mass transfer coefficient";
  parameter Real LIM_XREL = 0.997 "Flooding limit";

  outer parameter Real Svc  (unit="m2")  "Area of control plane ";
  outer parameter Real R  (unit="J/mol/K")  "Ideal gas constant";
  outer parameter Real Tg  (unit="K")  "Temperature";
  outer parameter Real rho_e  (unit="kg/m3")  "Electrolyte density";
  outer parameter Real Ee "Volume fraction occupied by electrolyte";
```

Modelica Code 13.8: PEM fuel cell (2/22). Control volume of membrane.

```
  Real Epores "Volume ratio occupied by pores";
  Real Vpores  (unit="m3")   "Volume of the pores";
  Real G_H2O_g  (unit="mol/s")
        "Moles of steam water generated per unit of time";
  Real G_H2O_l  (unit="mol/s")
        "Moles of liquid water generated per unit of time";
  Real m_H2O_l  (unit="kg")   "Mass of liquid water";
  Real m_s  (unit="kg")   "Mass of the solid";
  Real V_H2O_l  (unit="m3")   "Volume of liquid water";
  Real V_g  (unit="m3")   "Volume of the gas";
  Real Xmax "Maximum liquid water load";
  Real Lv  (unit="J/mol")   "Evaporation molar enthalpy";
  Real P_H2O_sat  (unit="Pa")   "Water saturation pressure at To_sat";
  Real n_H2O_g  (unit="mol")   "Moles of steam water";
  Real n_H2O_l  (unit="mol")   "Moles of liquid water";
  Real Xrel "Relative liquid water load";

equation
  der(n_H2O_g) = p.F_H2O_g + G_H2O_g;
  m_s / M_H2O * der(p.X) = p.F_H2O_l + G_H2O_l;
  n_H2O_l = m_H2O_l / M_H2O;
  V_H2O_l = m_H2O_l / rho_H2O_l;
  p.p_O2 = 1e-3;
  p.p_H2O_g * V_g = n_H2O_g * R * Tg;
  Epores = 1 - Ee;
  Epores = Vpores / Vvc;
  Vpores = V_g + V_H2O_l;
  p.E_g = V_g / Vvc;
  p.X = m_H2O_l / m_s;
  Xmax = rho_H2O_l * Vpores / m_s;
  Xrel = p.X / Xmax;
  m_s = Vvc * rho_e * Ee;
  G_H2O_l = Vvc * alfa_v * beta * (p.p_H2O_g - P_H2O_sat) / (R * Tg);
  G_H2O_g = -G_H2O_l;
  P_H2O_sat = P0_H2O_sat * exp((1 / To_sat - 1 / Tg) * Lv * M_H2O / R);
  Lv = 1.73287e6 + 1.03001e-4 * Tg - 4.47755e1 * Tg ^ 2 + 7.6629e-2 * Tg ^ 3
       - 5.5058e-5 * Tg ^ 4;
  p.ve = 0;
  p.ip = 0;

initial equation
  p.p_H2O_g = 1e3;
  p.X = 1e-2;
end VC_m;
```

Modelica Code 13.9: PEM fuel cell (3/22). Control volume of membrane.

```
model VC_dif
  Connectors.P_F3 p;

  parameter Real Vvc = Svc * Lvc  (unit="m3")  "Volume of the control volume";
  parameter Real M_H2O = 0.018  (unit="kg/mol")  "Molar mass of water";
  parameter Real rho_H2O_l = 972  (unit="kg/m3")  "Liquid water density";
  parameter Real Lvc  (unit="m")  "Length of control volume";
  parameter Real P0_H2O_sat = 3169  (unit="Pa")
        "Water saturation pressure at To_sat";
  parameter Real To_sat = 298.16  (unit="K")  "Saturation temperature";
  parameter Real alfa_v = 1e3  (unit="m2")  "Condensation specific surface";
  parameter Real beta = 0.001  (unit="m/s")  M̈ass transfer coefficient";
  parameter Real LIM_XREL = 0.997 "Limit rate of flooding";

  outer parameter Real Svc  (unit="m2")  "Control plane area";
  outer parameter Real R  (unit="J/mol/K")  Ïdeal gas constant";
  outer parameter Real Tg  (unit="K")  "Temperature";
  outer parameter Real rho_s  (unit="kg/m3")  "Solid density";
  outer parameter Real Es "Fraction of volume occupied by the solid";

  Real Epores "Rate of volume of porous media vs volume of control volume";
  Real Vpores  (unit="m3")  "Volume of the pores";
  Real G_H2O_g  (unit="mol/s")
        "Moles of steam water generated per unit of time";
  Real G_H2O_l  (unit="mol/s")
        "Moles of liquid water generated per unit of time";
  Real m_H2O_l  (unit="kg")  "Mass of liquid water";
  Real m_s  (unit="kg")  "Mass of the solid";
  Real V_H2O_l  (unit="m3")  "Volume of liquid water";
  Real V_g  (unit="m3")  "Volume of the gas";
  Real Xmax "Maximum liquid water load";
  Real Lv  (unit="J/mol")  "Evaporation molar enthalpy";
  Real P_H2O_sat  (unit="Pa")  "Water saturation pressure at To_sat";
  Real n_H2O_g  (unit="mol")  "Moles of steam water";
  Real n_H2O_l  (unit="mol")  "Moles of liquid water";
  Real n_O2  (unit="mol")  "Moles of oxygen";
  Real Xrel "Relative liquid water load";
```

Modelica Code 13.10: PEM fuel cell (4/22). Control volume of diffusion layer.

```
equation
  der(p.p_O2) * V_g - p.p_O2 * m_s / rho_H2O_l * der(p.X) = p.F_O2 * R * Tg;
  der(p.p_H2O_g) * V_g - p.p_H2O_g * m_s / rho_H2O_l * der(p.X) =
          (p.F_H2O_g + G_H2O_g) * R * Tg;
  m_s / M_H2O * der(p.X) = p.F_H2O_l + G_H2O_l;
  n_H2O_l = m_H2O_l / M_H2O;
  V_H2O_l = m_H2O_l / rho_H2O_l;
  p.p_O2 * V_g = n_O2 * R * Tg;
  p.p_H2O_g * V_g = n_H2O_g * R * Tg;
  Epores = 1 - Es;
  Epores = Vpores / Vvc;
  Vpores = V_g + V_H2O_l;
  p.E_g = V_g / Vvc;
  p.X = m_H2O_l / m_s;
  Xmax = rho_H2O_l * Vpores / m_s;
  Xrel = p.X / Xmax;
  m_s = Vvc * (rho_s * Es);
  G_H2O_l = Vvc * alfa_v * beta * (p.p_H2O_g - P_H2O_sat) / (R * Tg);
  G_H2O_g = -G_H2O_l;
  P_H2O_sat = P0_H2O_sat * exp((1 / To_sat - 1 / Tg) * Lv * M_H2O / R);
  Lv = 1.73287e6 + 1.03001e-4 * Tg - 4.47755e1 * Tg ^ 2 + 7.6629e-2 * Tg ^ 3
          - 5.5058e-5 * Tg ^ 4;
  p.ie = 0;
  p.ip = 0;

  when Xrel > LIM_XREL then
    reinit(p.X, Xmax);
    reinit(p.p_O2, 0);
    reinit(p.p_H2O_g, 0);
  end when;

initial equation
  p.p_O2 = 1e5;
  p.p_H2O_g = 1e3;
  p.X = 1e-2;
end VC_dif;
```

Modelica Code 13.11: PEM fuel cell (5/22). Control volume of diffusion layer.

```
model VC_cat
  Connectors.P_F3 p;

  parameter Real Vvc = Svc * Lvc  (unit="m3")  "Volume of the control volume";
  parameter Real M_H2O = 0.018  (unit="kg/mol")  "Molar mass of water";
  parameter Real rho_H2O_l = 972  (unit="kg/m3")  "Liquid water density";
  parameter Real Lvc  (unit="m")  "Length of the control volume";
  parameter Real P0_H2O_sat = 3169  (unit="Pa")
        "Water saturation pressure at To_sat";
  parameter Real To_sat = 298.16  (unit="K")  "Saturation temperature";
  parameter Real alfa_v = 1e3  (unit="m2")  "Condensation specific surface";
  parameter Real beta = 0.001  (unit="m/s")  "Mass transfer coefficient";
  parameter Real a_act_i0ref = 1.2  (unit="A/m3")
          "Specific area of catalytic surface * Reference exchange current";
  parameter Real p0_O2 = 1e5  (unit="Pa")
        "Reference partial pressure of oxygen";
  parameter Real F = 96485  (unit="A*s/mol")  "Faraday constant";
  parameter Real B = 0.04  (unit="V")  "Tafel slope";
  parameter Real LIM_XREL = 0.997 "Flooding maximum limit";

  outer parameter Real Svc  (unit="m2")  "Area of control plane";
  outer parameter Real R  (unit="J/mol/K")  "Ideal gas constant";
  outer parameter Real Tg  (unit="K")  "Temperature";
  outer parameter Real rho_s  (unit="kg/m3")  "Solid density";
  outer parameter Real Es "Volume fraction occupied by the solid";
  outer parameter Real rho_e  (unit="kg/m3")  "Electrolyte density";
  outer parameter Real Ee "Volume fraction occupied by electrolyte";
  outer Real Eoc_  (unit="V")  "Voltage difference Vp-Ve";

  Real Epores "Volume fraction occupied by pores";
  Real Vpores  (unit="m3")  "Volume of the pores";
  Real G_H2O_g  (unit="mol/s")
        "Moles of steam water generated per unit of time";
  Real G_H2O_l  (unit="mol/s")
        "Moles of liquid water generated per unit of time";
  Real m_H2O_l  (unit="kg")  "Mass of liquid water";
  Real m_s  (unit="kg")  "Mass of solid";
  Real V_H2O_l  (unit="m3")  "Volume of liquid water";
  Real V_g  (unit="m3")  "Volume of the gas";
  Real Xmax "Maximum liquid water load";
  Real Lv  (unit="J/mol")  "Evaporation molar enthalpy";
  Real P_H2O_sat  (unit="Pa")  "Water saturation pressure at To_sat";
  Real n_H2O_g  (unit="mol")  "Moles of steam water";
  Real n_H2O_l  (unit="mol")  "Moles of liquid water";
  Real n_O2  (unit="mol")  "Moles of oxygen";
  Real Xrel "Relative liquid water load";
  Real Ge  (unit="A")  "Electrons generated per unit of time in the reaction";
  Real Gp  (unit="A")  "Protons generated per unit of time in reaction";
  Real G_reac_H2O_g  (unit="mol/s")
      "Moles of steam water generated in the reaction per unit of time";
  Real G_reac_O2  (unit="mol/s")
        "Moles of oxygen generated in the reaction per unit of time";
```

Modelica Code 13.12: PEM fuel cell (6/22). Control volume of catalyst layer.

```
equation
  der(p.p_O2) * V_g - p.p_O2 * m_s / rho_H2O_l * der(p.X) =
                                        (p.F_O2 + G_reac_O2) * R * Tg;
  der(p.p_H2O_g) * V_g - p.p_H2O_g * m_s / rho_H2O_l * der(p.X) =
                              (p.F_H2O_g + G_H2O_g + G_reac_H2O_g) * R * Tg;
  m_s / M_H2O * der(p.X) = p.F_H2O_l + G_H2O_l;
  n_H2O_l = m_H2O_l / M_H2O;
  V_H2O_l = m_H2O_l / rho_H2O_l;
  p.p_O2 * V_g = n_O2 * R * Tg;
  p.p_H2O_g * V_g = n_H2O_g * R * Tg;
  Epores = 1 - Ee - Es;
  Epores = Vpores / Vvc;
  Vpores = V_g + V_H2O_l;
  p.E_g = V_g / Vvc;
  p.X = m_H2O_l / m_s;
  Xmax = rho_H2O_l * Vpores / m_s;
  Xrel = p.X / Xmax;
  m_s = Vvc * (rho_e * Ee + rho_s * Es);
  G_H2O_l = if p.p_H2O_g > 0 or p.X > 0
            then Vvc * alfa_v * beta * (p.p_H2O_g - P_H2O_sat) / (R * Tg)
            else 0;
  G_H2O_g = -G_H2O_l;
  P_H2O_sat = P0_H2O_sat * exp((1 / To_sat - 1 / Tg) * Lv * M_H2O / R);
  Lv = 1.73287e6 + 1.03001e-4 * Tg - 4.47755e1 * Tg ^ 2 + 7.6629e-2 * Tg ^ 3
      - 5.5058e-5 * Tg ^ 4;
  Ge = -a_act_i0ref * (p.p_O2 / p0_O2) * exp((Eoc_ - p.ve + p.vp) / B);
  Gp = Ge;
  p.ie - Ge = 0;
  p.ip + Gp = 0;
  G_reac_O2 = Ge / (4 * F);
  G_reac_H2O_g = -Ge / (2 * F);

  when Xrel > LIM_XREL then
    reinit(p.X, Xmax);
    reinit(p.p_O2, 0);
    reinit(p.p_H2O_g, 0);
  end when;

initial equation
  p.p_O2 = 1e5;
  p.p_H2O_g = 1e3;
  p.X = 1e-2;
end VC_cat;
```

Modelica Code 13.13: PEM fuel cell (7/22). Control volume of catalyst layer.

```
model FT_m
  Connectors.P_F3 p1;
  Connectors.P_F3 p2;

  parameter Real Lvc  (unit="m")  "Length of control volume";
  parameter Real Vvc = Svc * Lvc  (unit="m3")  "Volume of control volume";
  parameter Real M_H2O = 0.018  (unit="kg/mol")  "Molar mass of water";
  parameter Real rho_H2O_l = 972  (unit="kg/m3")  "Liquid water density";
  parameter Real D_H2O_l = 5.5e-7  (unit="m2/s")
        "Diffusion coefficient of liquid water";
  parameter Real D_H2O_g_k = 1e-11  (unit="m2/s")
        "Knudsen diffusion coefficient of steam water";
  parameter Real tau = 1 "Tortuosity";
  parameter Real sigma_e = 1e-6  (unit="S/m")  "Solid conductivity";
  parameter Real Kp = 10  (unit="S/m")  "Protonic conductivity of electrolyte";
  outer parameter Real Svc  (unit="m2")  "Area of the control volume";
  outer parameter Real R  (unit="J/mol/K")  "Ideal gas constant";
  outer parameter Real Tg  (unit="K")  "Temperature of the medium";
  outer parameter Real rho_e  (unit="kg/m3")  "Electrolyte density";
  outer parameter Real Ee "Volume fraction of the electrolyte";

  Real J_H2O_g  (unit="mol/s/m2")  "Flux per area unit of steam water";
  Real Egmin "Minimum volume of gas transport";

equation
  Egmin = max(min(p1.E_g, p2.E_g), 0);
  J_H2O_g = if p1.p_H2O_g > 0 or p2.p_H2O_g > 0
     then Egmin * (p1.p_H2O_g - p2.p_H2O_g) * D_H2O_g_k / (Lvc * R * Tg * tau ^ 2)
     else 0;
  p2.F_H2O_g = -J_H2O_g * Svc;
  p1.F_H2O_l = if p1.X > 0 or p2.X > 0
     then D_H2O_l * rho_e * Svc * (p1.X - p2.X) / (Lvc * M_H2O)
     else 0;
  p1.F_O2 = 0;
  p1.F_H2O_g = -p2.F_H2O_g;
  p1.F_H2O_l = -p2.F_H2O_l;
  p1.F_O2 = -p2.F_O2;
  p1.ie = 0;
  p1.ip = Svc * Kp * Ee * (p1.vp - p2.vp) / Lvc;
  p1.ie = -p2.ie;
  p1.ip = -p2.ip;
end FT_m;
```

Modelica Code 13.14: PEM fuel cell (8/22). Transport phenomena in the membrane.

```
model FT_dif
  Connectors.P_F3 p1;
  Connectors.P_F3 p2;
  parameter Real Lvc  (unit="m")  "Length of control volume";
  parameter Real Vvc = Svc * Lvc  (unit="m3")  "Volume of control volume";
  parameter Real M_H2O = 0.018  (unit="kg/mol")  "Molar mass of water";
  parameter Real rho_H2O_l = 972  (unit="kg/m3")  "Liquid water density";
  parameter Real D_H2O_l = 3.5e-7  (unit="m2/s")
        "Diffusion coefficient of liquid water";
  parameter Real D_O2_H2O_g_ref = 2.82e-5  (unit="m2/s")
        "Binary diffusion coefficient";
  parameter Real D_H2O_g_k = 1.047e-6  (unit="m2/s")
        "Knudsen diff. coeff. of steam water";
  parameter Real D_O2_k = 7.853e-7  (unit="m2/s")
        "Knudsen diffusion coefficient of oxygen";
  parameter Real p_g_ref = 1e5  (unit="Pa")
        "Reference pressure for binary diffusion";
  parameter Real Tg_ref = 308.1  (unit="K")
        "Reference temperature for binary diffusion";
  parameter Real tau = 1 "Tortuosity";
  parameter Real sigma_e = 1e4  (unit="S/m")  "Solid conductivity";
  parameter Real Kp = 1e-7  (unit="S/m")
        "Protonic conductivity of electrolyte";
  outer parameter Real Svc  (unit="m2")  "Area of the control volume";
  outer parameter Real R  (unit="J/mol/K")  "Ideal gas constant";
  outer parameter Real Tg  (unit="K")  "Temperature of the medium";
  outer parameter Real rho_s  (unit="kg/m3")  "Solid density";
  outer parameter Real Es "Volume rate of the solid";
  Real pH2O_g  (unit="Pa")  "Partial pressure of steam water";
  Real pO2  (unit="Pa")  "Partial pressure of oxygen";
  Real p_g  (unit="Pa")  "Total pressure of the gases";
  Real J_H2O_g  (unit="mol/s/m2")  "Flux per area unit of steam water";
  Real J_O2  (unit="mol/s/m2")  "Flux per area unit of oxygen";
  Real D_O2_H2O_g  (unit="m2/s")  "Binary diffusion coefficient";
  Real Egmin "Minimum volume of gas transport";
```

Modelica Code 13.15: PEM fuel cell (9/22). Transport phenomena in the diffusion layer.

```
equation
  pO2 = max(p1.p_O2, p2.p_O2);
  pH2O_g = max(p1.p_H2O_g, p2.p_H2O_g);
  p_g = pO2 + pH2O_g;
  Egmin = max(min(p1.E_g, p2.E_g), 0);
  D_O2_H2O_g = D_O2_H2O_g_ref * (p_g / p_g_ref) * (Tg / Tg_ref) ^ 1.5;
  J_O2 = if p_g > 0
     then Egmin / tau ^ 2 * (p1.p_O2 - p2.p_O2) /
               (Lvc * R * Tg * (1 / D_O2_k + 1 / D_O2_H2O_g))
     else 0;
  J_H2O_g = if p_g > 0
     then Egmin / tau ^ 2 * (p1.p_H2O_g - p2.p_H2O_g) /
               ( Lvc * R * Tg * (1 / D_H2O_g_k + 1 / D_O2_H2O_g))
     else 0;
  p2.F_H2O_g = -J_H2O_g * Svc;
  p2.F_O2 = -J_O2 * Svc;
  p1.F_H2O_l = if p1.X > 0 or p2.X > 0
     then D_H2O_l * rho_s * Svc * (p1.X - p2.X) / (Lvc * M_H2O)
     else 0;
  p1.F_H2O_g = -p2.F_H2O_g;
  p1.F_H2O_l = -p2.F_H2O_l;
  p1.F_O2 = -p2.F_O2;
  p1.ie = Svc * sigma_e * Es * (p1.ve - p2.ve) / Lvc;
  p1.ip = Svc * Kp * Es * (p1.vp - p2.vp) / Lvc;
  p1.ie = -p2.ie;
  p1.ip = -p2.ip;
end FT_dif;
```

Modelica Code 13.16: PEM fuel cell (10/22). Transport phenomena in the diffusion layer.

```
model FT_cat
  Connectors.P_F3 p1;
  Connectors.P_F3 p2;
  parameter Real Lvc  (unit="m")   "Length of control volume";
  parameter Real Vvc = Svc * Lvc  (unit="m3")   "Volume of control volume";
  parameter Real M_H2O = 0.018  (unit="kg/mol")   "Molar mass of water";
  parameter Real rho_H2O_l = 972  (unit="kg/m3")   "Liquid water density";
  parameter Real D_H2O_l = 2.5e-7  (unit="m2/s")
        "Diffusion coefficient of liquid water";
  parameter Real D_O2_H2O_g_ref = 2.82e-5  (unit="m2/s")
        "Binary diffusion coefficient";
  parameter Real D_H2O_g_k = 1.047e-6  (unit="m2/s")
        "Knudsen diff. coeff. of steam water";
  parameter Real D_O2_k = 7.853e-7  (unit="m2/s")
        "Knudsen diffusion coefficient of oxygen";
  parameter Real p_g_ref = 1e5  (unit="Pa")
        "Reference pressure for binary diffusion";
  parameter Real Tg_ref = 308.1  (unit="K")
        "Reference temperature for binary diffusion";
  parameter Real tau = 5 "Tortuosity";
  parameter Real sigma_e = 1  (unit="S/m")   "Solid conductivity";
  parameter Real Kp = 0.1  (unit="S/m")
        "Protonic conductivity of electrolyte";
  outer parameter Real Svc  (unit="m2")   "Area of the control plane";
  outer parameter Real R  (unit="J/mol/K")   "Ideal gas constant";
  outer parameter Real Tg  (unit="K")   "Temperature of the medium";
  outer parameter Real rho_e  (unit="kg/m3")"Electrolyte density";
  outer parameter Real rho_s  (unit="kg/m3")"Solid density";
  outer parameter Real Ee "Volume rate of the electrolyte";
  outer parameter Real Es "Volume rate of the solid";
  Real pH2O_g  (unit="Pa")   "Partial pressure of steam water";
  Real pO2  (unit="Pa")   "Partial pressure of oxygen";
  Real p_g  (unit="Pa")   "Total pressure of the gases";
  Real J_H2O_g  (unit="mol/s/m2")   "Flux per area unit of steam water";
  Real J_O2  (unit="mol/s/m2")   "Flux per area unit of oxygen";
  Real D_O2_H2O_g  (unit="m2/s")   "Binary diffusion coefficient";
  Real Egmin "Minimum volume of gas transport";
```

Modelica Code 13.17: PEM fuel cell (11/22). Transport phenomena in the catalyst layer.

```
equation
  pO2 = max(p1.p_O2, p2.p_O2);
  pH2O_g = max(p1.p_H2O_g, p2.p_H2O_g);
  p_g = pO2 + pH2O_g;
  Egmin = max(min(p1.E_g, p2.E_g), 0);
  D_O2_H2O_g = D_O2_H2O_g_ref * (p_g / p_g_ref) * (Tg / Tg_ref) ^ 1.5;
  J_O2 = if p_g > 0 then Egmin / tau ^ 2 * (p1.p_O2 - p2.p_O2) /
            (Lvc * R * Tg * (1 / D_O2_k + 1 / D_O2_H2O_g)) else 0;
  J_H2O_g = if p_g > 0  then Egmin / tau ^ 2 * (p1.p_H2O_g - p2.p_H2O_g) /
           (Lvc * R * Tg * (1 / D_H2O_g_k + 1 / D_O2_H2O_g))  else 0;
  p2.F_H2O_g = -J_H2O_g * Svc;
  p2.F_O2 = -J_O2 * Svc;
  p1.F_H2O_l = if p1.X > 0 or p2.X > 0 then D_H2O_l * (rho_s + rho_e) * Svc *
            (p1.X - p2.X) / (Lvc * M_H2O) else 0;
  p1.F_H2O_g = -p2.F_H2O_g;
  p1.F_H2O_l = -p2.F_H2O_l;
  p1.F_O2 = -p2.F_O2;
  p1.ie = Svc * sigma_e * Es * (p1.ve - p2.ve) / Lvc;
  p1.ip = Svc * Kp * Ee * (p1.vp - p2.vp) / Lvc;
  p1.ie = -p2.ie;
  p1.ip = -p2.ip;
end FT_cat;
```

Modelica Code 13.18: PEM fuel cell (12/22). Transport phenomena in the catalyst layer.

```
model membrane
  Connectors.P_F3 p1;
  Connectors.P_F3 p2;

  parameter Integer N = 20 "Number of control volumes";
  parameter Real L_mem = 8e-5  (unit="m")  "Total length of the layer";
  parameter Real l_mem = L_mem / N  (unit="m")  "Length of the control volume";
  parameter Real dimension_mem[N + 1] =
       {if i == 1 then l_mem / 2 else l_mem for i in 1:N + 1};

  inner parameter Real Svc = 1 "Area of the control volume";
  inner parameter Real R = 8.31447 "Ideal gas constant";
  inner parameter Real Tg = 340 "Temperature of the medium";
  inner parameter Real rho_e = 2000 "Electrolyte density";
  inner parameter Real Ee = 0.72 "Volume rate of the electrolyte";

  VC_m v_c_mem[N](Lvc = fill(l_mem, N));
  FT_m f_t_mem[N + 1](Lvc = dimension_mem);

equation
  connect(f_t_mem[1].p1, p1);

  for conta in 1:N loop
    connect(f_t_mem[conta].p2, v_c_mem[conta].p);
    connect(v_c_mem[conta].p, f_t_mem[conta + 1].p1);
  end for;

  connect(f_t_mem[N + 1].p2, p2);
end membrane;
```

Modelica Code 13.19: PEM fuel cell (13/22). Membrane layer.

```
model diffusion
  Connectors.P_F3 p1;
  Connectors.P_F3 p2;

  parameter Integer N = 20 "Number of control volumes";
  parameter Real L_dif = 1.6e-3  (unit="m")   "Total length of the layer";
  parameter Real l_dif = L_dif / N  (unit="m")   "Length of the control volume";
  parameter Real dimension_dif[N ] =
          {if i == N then l_dif / 2 else l_dif for i in 1:N};

  inner parameter Real Svc = 1 "Area of the control plane";
  inner parameter Real R = 8.31447 "Ideal gas constant";
  inner parameter Real Tg = 340 "Temperature of the medium";
  inner parameter Real rho_s = 4000 "Solid density";
  inner parameter Real Es = 0.1 "Volume rate of the solid";

  VC_dif v_c_dif[N](Lvc = fill(l_dif, N));
  FT_dif f_t_dif[N](Lvc = dimension_dif);

equation
  connect(f_t_dif[1].p1, p1);

  for conta in 1:N - 1 loop
    connect(v_c_dif[conta].p, f_t_dif[conta].p1);
    connect(f_t_dif[conta].p2, v_c_dif[conta + 1].p);
  end for;

  connect(v_c_dif[N].p, f_t_dif[N].p1);
  connect(f_t_dif[N].p2, p2);
end diffusion;
```

Modelica Code 13.20: PEM fuel cell (14/22). Diffusion layer.

```
model catalyst
  Connectors.P_F3 p1;
  Connectors.P_F3 p2;
  Connectors.pin_Eoc pin_Eoc;

  parameter Integer N = 20 "Number of control volumes";
  parameter Real L_cat = 4e-5  (unit="m")  "Total length of the layer";
  parameter Real l_cat = L_cat / N  (unit="m")  "Length of the control volume";
  parameter Real dimension_cat[N + 1] =
        {if i == 1 or i == N + 1 then l_cat / 2 else l_cat for i in 1:N + 1};

  inner parameter Real Svc = 1 "Area of the control plane";
  inner parameter Real R = 8.31447 "Ideal gas constant";
  inner parameter Real Tg = 340 "Temperature of the medium";
  inner parameter Real rho_e = 2000 "Electrolyte density";
  inner parameter Real rho_s = 4000 "Solid density";
  inner parameter Real Ee = 0.2 "Volume rate of the electrolyte";
  inner parameter Real Es = 0.6 "Volume rate of the solid";
  inner Real Eoc_ "Voltage difference Vp-Ve";

  VC_cat v_c_cat[N](Lvc = fill(l_cat, N));
  FT_cat f_t_cat[N](Lvc = fill(l_cat, N));

equation
  connect(v_c_cat[1].p, p1);

  for conta in 1:N - 1 loop
    connect(v_c_cat[conta].p, f_t_cat[conta].p1);
    connect(f_t_cat[conta].p2, v_c_cat[conta + 1].p);
  end for;

  connect(v_c_cat[N].p, f_t_cat[N].p1);
  connect(f_t_cat[N].p2, p2);

  Eoc_ = pin_Eoc.Eoc;
end catalyst;
```

Modelica Code 13.21: PEM fuel cell (15/22). Catalyst layer.

```
connector Pin
  Modelica.SIunits.Voltage v;
  flow Modelica.SIunits.Current i;
end Pin;
```

Modelica Code 13.22: PEM fuel cell (16/22). Standard electrical connector.

```
partial model Ext_Inter
  Connectors.Pin p1;
  Connectors.P_F3 p2;
end Ext_Inter;
```

Modelica Code 13.23: PEM fuel cell (17/22). Composed connector.

```
model IntMembrane
  extends Interfaces.Ext_Inter;
equation
  p1.v = p2.ve;
  p2.ve = p2.vp;
  p1.i + p2.ie + p2.ip = 0;
  p2.p_O2 = 1;
  p2.F_H2O_g = 0;
  p2.F_H2O_l = 0;
  p2.E_g = 1;
end IntMembrane;
```

Modelica Code 13.24: PEM fuel cell (18/22). Interface of membrane.

```
connector pin_Eoc
  Modelica.SIunits.Voltage Eoc "Open circuit voltage";
end pin_Eoc;
```

Modelica Code 13.25: PEM fuel cell (19/22). Open circuit voltage connector.

```
model IntCathode
  extends Interfaces.Ext_Inter;
  Connectors.pin_Eoc pin_Eoc;

  parameter Real R = 8.31447  (unit="J/mol/K")  "Ideal gas constant";
  parameter Real F = 96485  (unit="A*s/mol")  "Faraday constant";
  parameter Real Tg = 340  (unit="K")  "Temperature of the medium";
  parameter Real p_H2 = 100000  (unit="Pa")  "Hydrogen pressure";
  parameter Real p_O2 = 100000  (unit="Pa")  "Oxygen pressure";

equation
  p1.v = p2.ve;
  p2.ve - p2.vp = pin_Eoc.Eoc;
  p1.i + p2.ie + p2.ip = 0;
  p2.p_O2 = p_O2 ;
  p2.F_H2O_g = 0;
  p2.F_H2O_l = 0;
  p2.E_g = 1;
  pin_Eoc.Eoc=1.1-0.9e-3*(Tg - 298)+R * Tg /(2 * F) *
       log(p_H2/100000*(p2.p_O2/1000)^ 0.5/(p2.p_H2O_g/100000));
end IntCathode;
```

Modelica Code 13.26: PEM fuel cell (20/22). Interface of catalyst layer.

```
model FC
  Components.Layers.membrane m;
  Components.Layers.catalyst c;
  Components.Layers.diffusion d;
  Interfaces.External_interface.IntMembrane intm;
  Interfaces.External_interface.IntCathode intc;
  Connectors.Pin p1;
  Connectors.Pin p2;
equation
  connect(p1, intm.p1);
  connect(p2, intc.p1);
  connect(intm.p2, m.p1);
  connect(m.p2, c.p1);
  connect(c.p2, d.p1);
  connect(d.p2, intc.p2);
  connect(c.pin_Eoc, intc.pin_Eoc);
end FC;
```

Modelica Code 13.27: PEM fuel cell (21/22). Complete model of fuel cell.

```
model FC_polar
  Components.Resistance R;
  Components.Ground GND;

  FC FC1;
equation
  connect(FC1.p1, GND.p);
  connect(R.p, FC1.p2);
  connect(R.n, FC1.p1);
end FC_polar;
```

Modelica Code 13.28: PEM fuel cell (22/22). Polarized fuel cell.

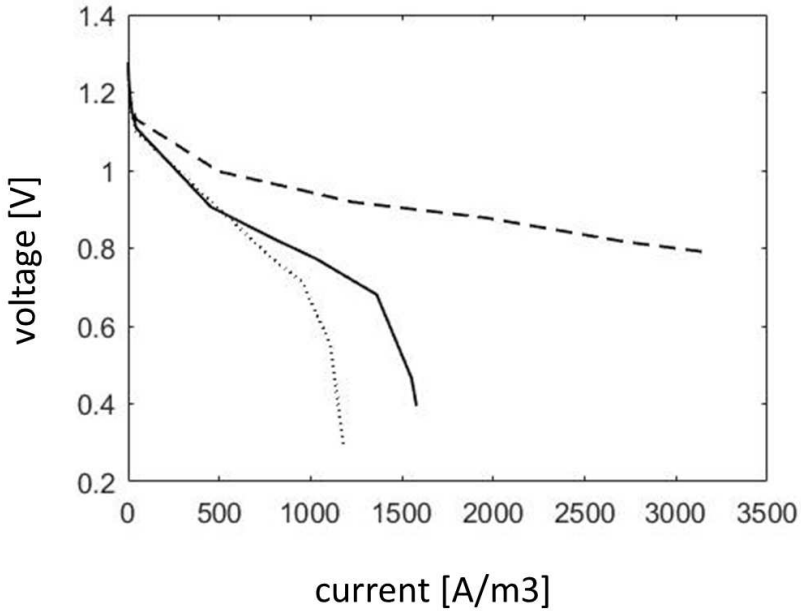


Figure 13.25: Simulation Task 1. Effect of the catalyst layer length (L_{cat}) on the polarization curve: [. . .] 6e-5 m, [——] 4e-5 m and [- - -] 2e-6 m.

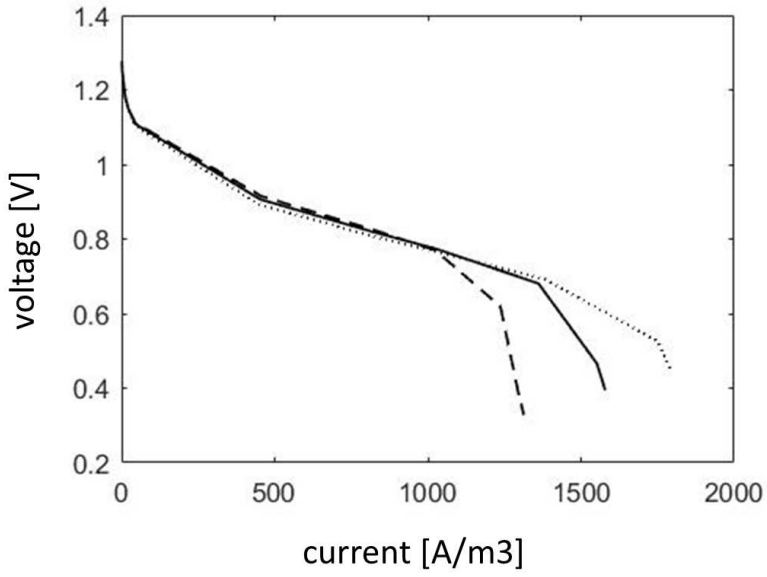


Figure 13.26: Simulation Task 1. Effect of the diffusion layer length (L_{dif}) on the polarization curve: [- - -] 4e-3 m, [——] 1.6e-3 m and [. . .] 0.5e-3 m.

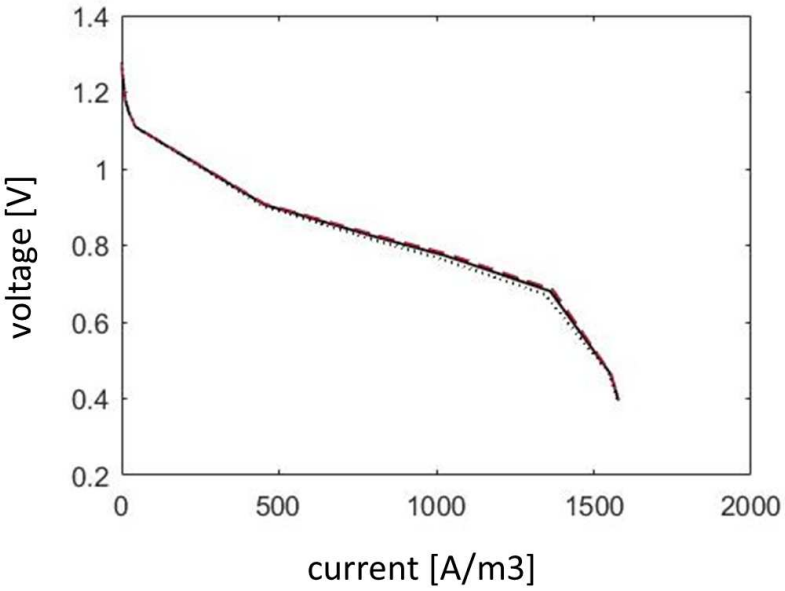


Figure 13.27: Simulation Task 1. Effect of the membrane length (L_{mem}) on the polarization curve: [- - -] 4e-5 m, [——] 8e-5 m and [. . .] 16e-5 m.

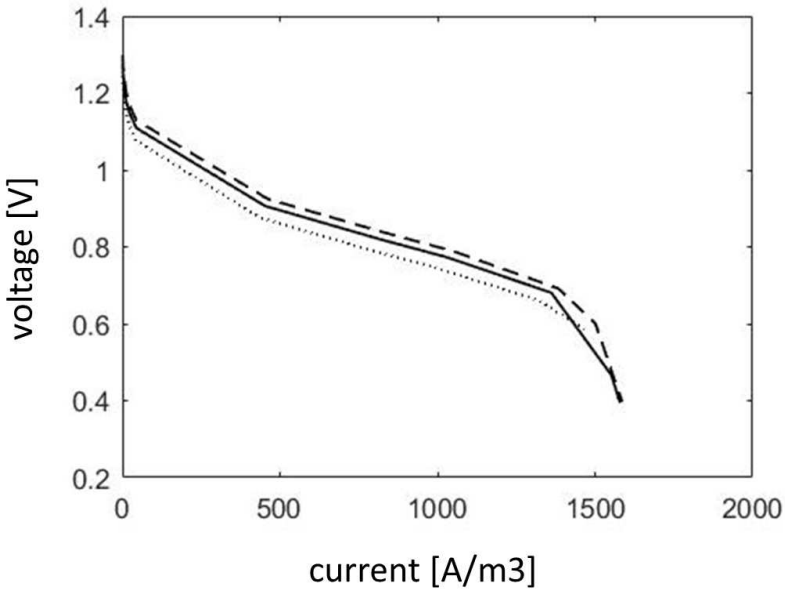


Figure 13.28: Simulation Task 1. Effect of the catalyst area and exchange current ($a_{act} \cdot i_0^{ref}$) on the polarization curve: [- - -] 2 A/m^3, [——] 1.2 A/m^3 and [. . .] 0.5 A/m^3.

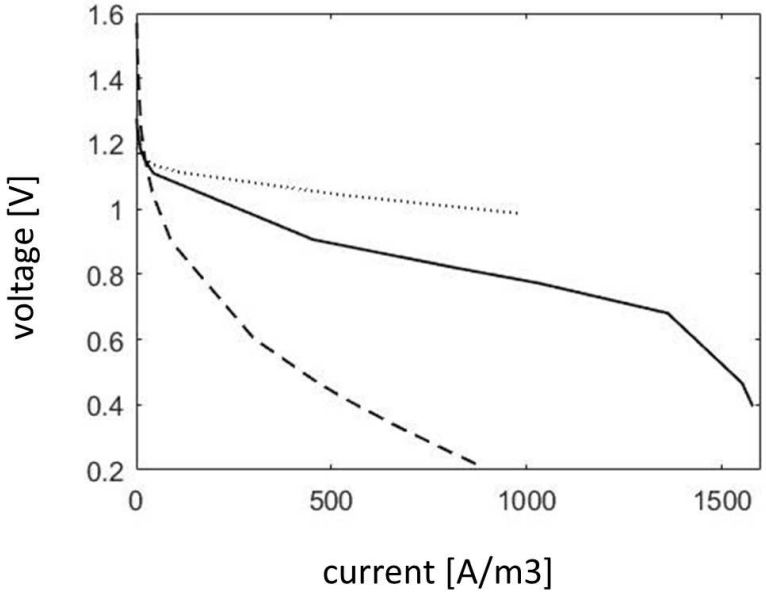


Figure 13.29: Simulation Task 1. Effect of the Tafel slope (B) on the polarization curve: [- - -] 0.15 V, [——] 0.04 V and [. . .] 0.01 V.

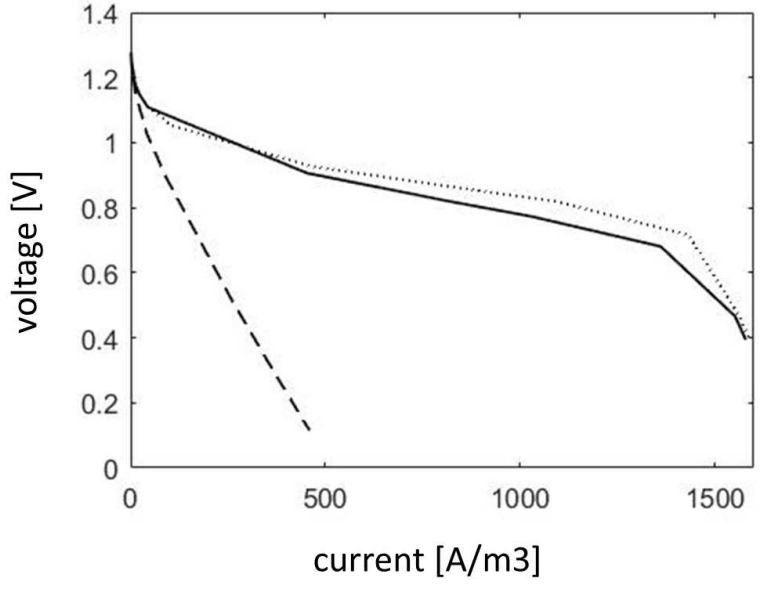


Figure 13.30: Simulation Task 1. Effect of the solid conductivity of the catalyst layer ($\sigma_{e,cat}$) on the polarization curve: [- - -] 0.01 S/m, [——] 1 S/m and [. . .] 100 S/m.

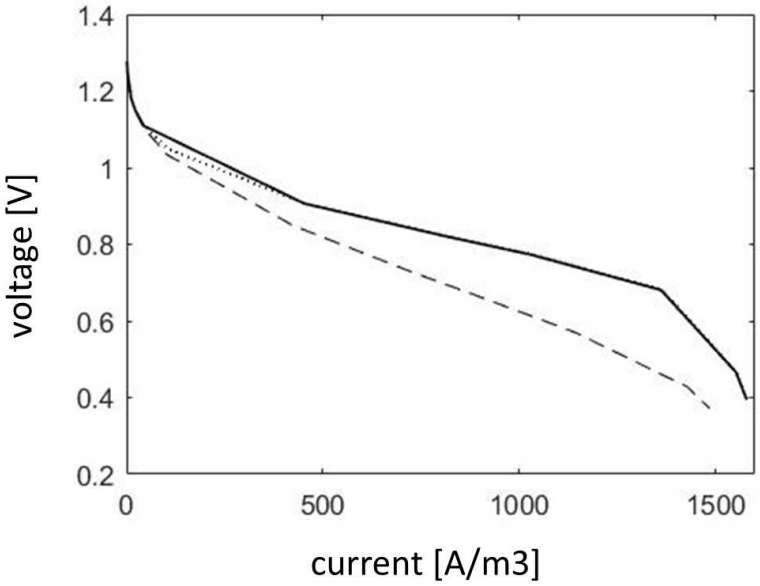


Figure 13.31: Simulation Task 1. Effect of the solid conductivity of the diffusion layer ($\sigma_{e,dif}$) on the polarization curve: [- - -] 100 S/m, [——] 1e4 S/m and [. . .] 1e5 S/m.

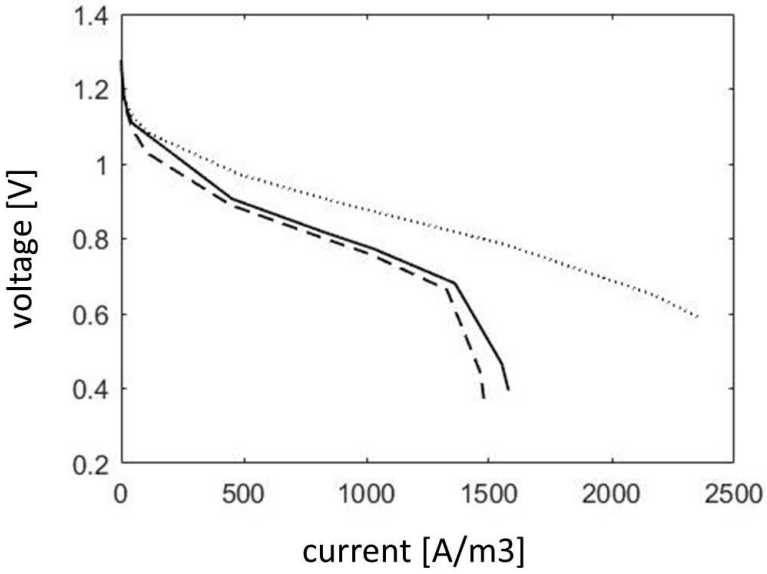


Figure 13.32: Simulation Task 1. Effect of the protonic conductivity of the catalyst layer ($k_{p,cat}$) on the polarization curve: [- - -] 0.05 S/m, [——] 0.1 S/m and [. . .] 1 S/m.

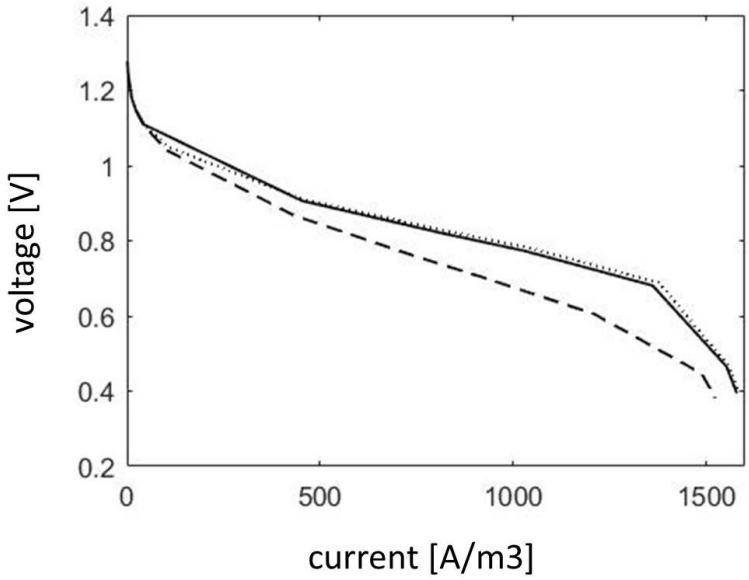


Figure 13.33: Simulation Task 1. Effect of the protonic conductivity of the membrane ($k_{p,mem}$) on the polarization curve: [- - -] 1 S/m, [——] 10 S/m and [. . .] 100 S/m.

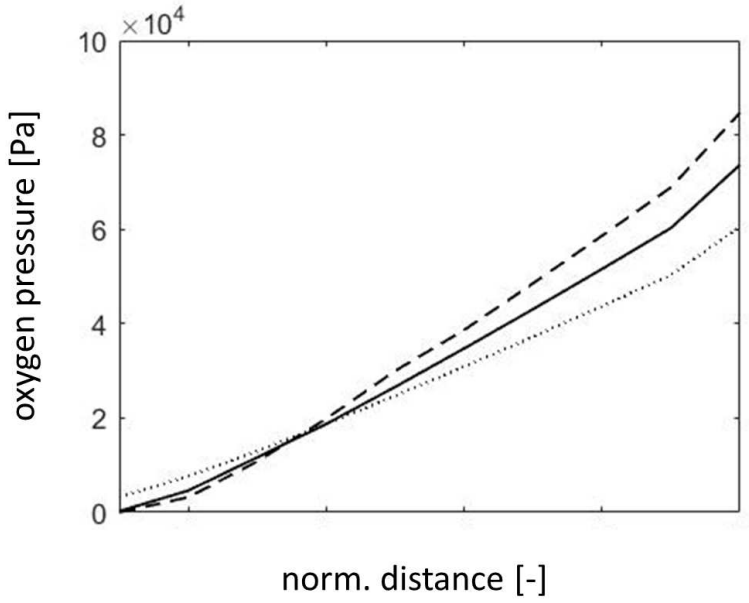


Figure 13.34: Simulation Task 2. Effect of the catalyst layer length (L_{cat}) on the oxygen pressure along the catalyst layer for high current: [- - -] 2e-5 m, [——] 4e-5 m and [. . .] 8e-5 m.

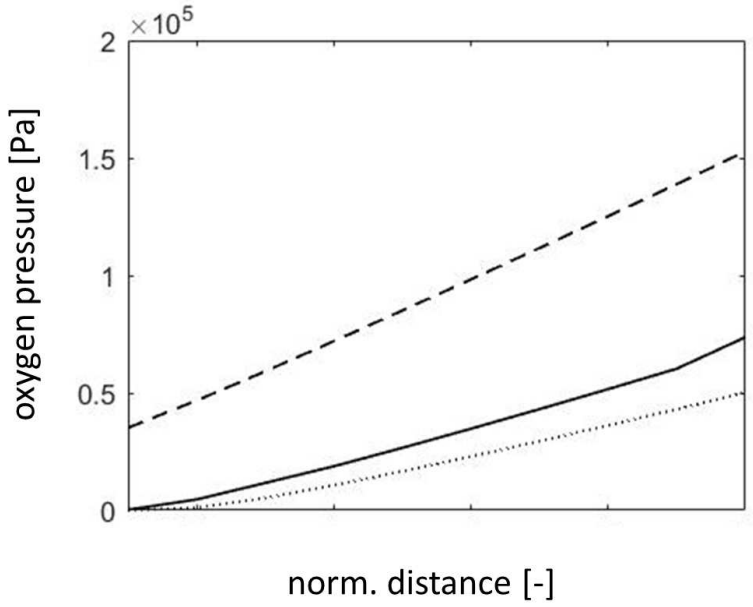


Figure 13.35: Simulation Task 2. Effect of the oxygen pressure used as boundary condition at the cathode interface (p_{O_2}) on the oxygen pressure along the catalyst layer for high current: [- - -] 2e5 Pa, [——] 1e5 Pa and [. . .] 7.5e4 Pa.

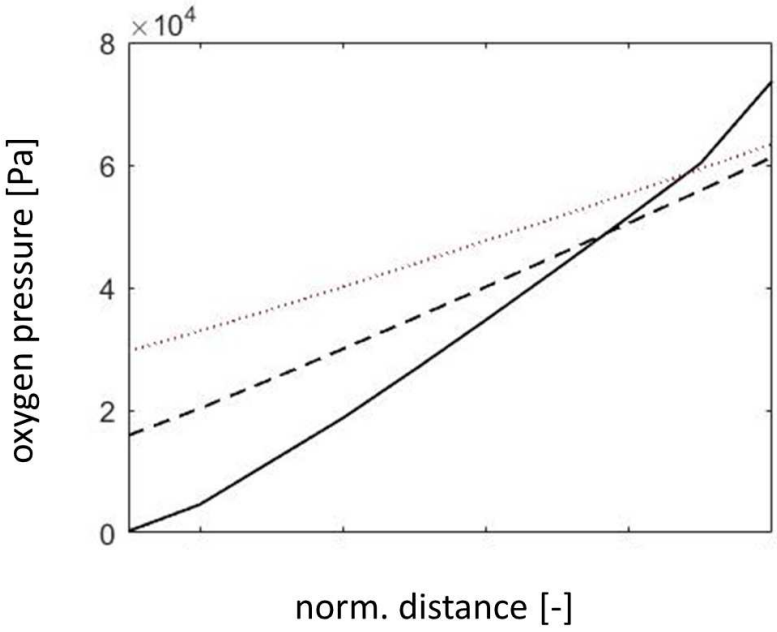


Figure 13.36: Simulation Task 2. Effect of the solid porosity of the catalyst layer (ε_s) on the oxygen pressure along the catalyst layer for high current: [- - -] 0.4, [——] 0.6 and [. . .] 0.3.

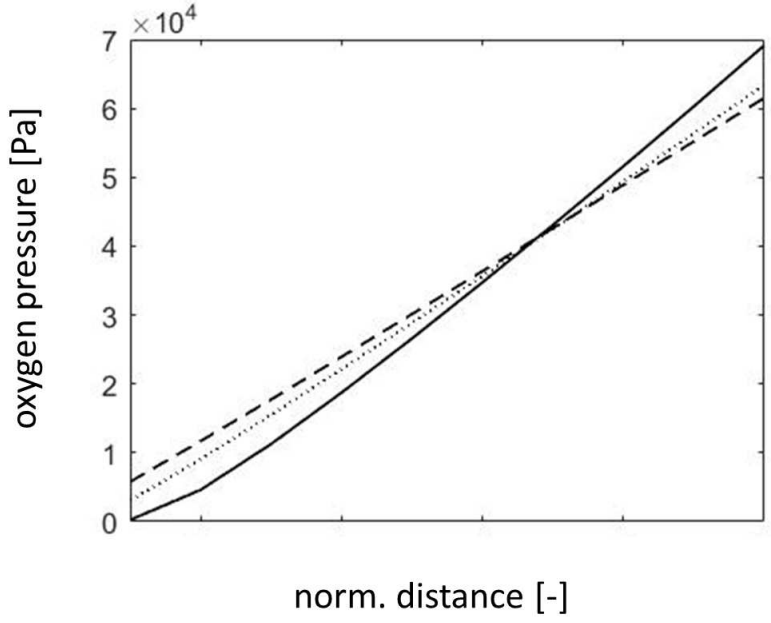


Figure 13.37: Simulation Task 2. Effect of the electrolyte porosity of the catalyst layer (ε_e) on the oxygen pressure along the catalyst layer for high current: [- - -] 0.05, [——] 0.2 and [. . .] 0.1.

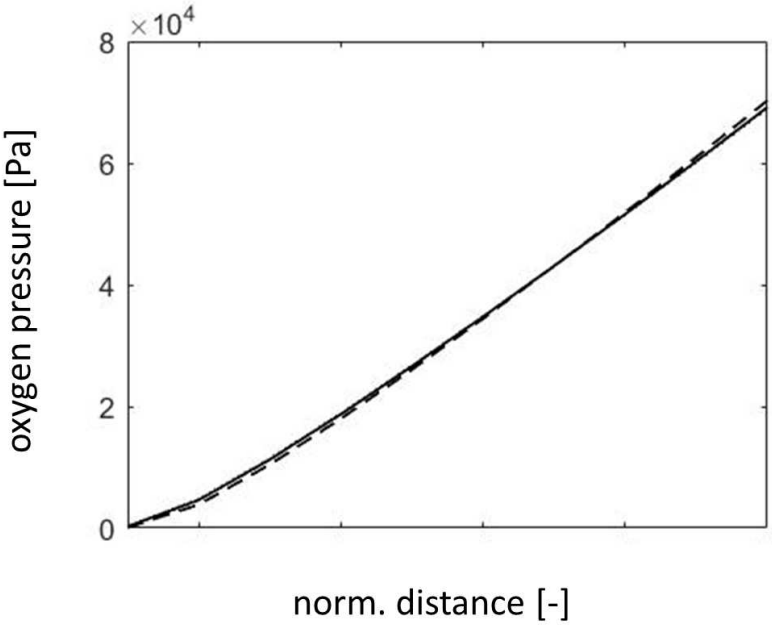


Figure 13.38: Simulation Task 2. Effect of the binary diffusion coefficient ($D_{O_2/H_2O,g}$) on the oxygen pressure along the catalyst layer for high current: [- - -] 5e-6 m^2/s, [——] 2.82e-5 m^2/s and [. . .] 1e-4 m^2/s.

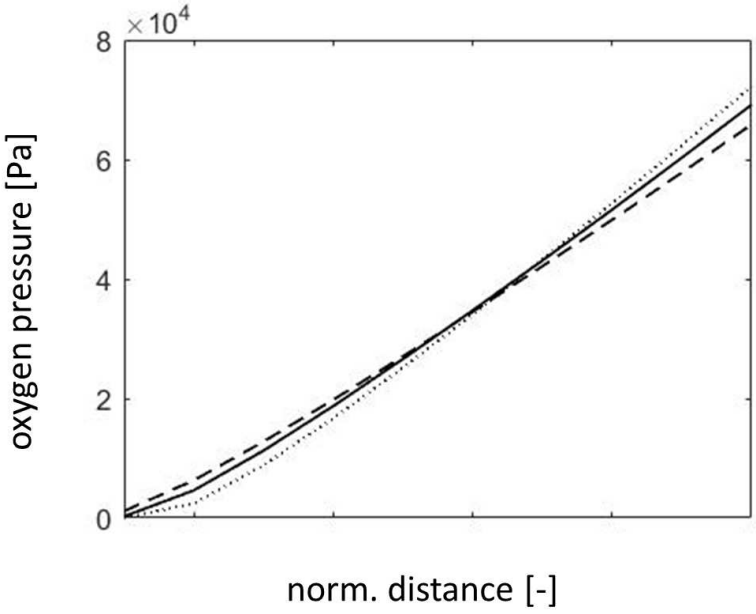


Figure 13.39: Simulation Task 2. Effect of the Knudsen diffusion coefficient of oxygen (D_{O_2K}) on the oxygen pressure along the catalyst layer for high current: [- - -] 1e-6 m^2/s, [——] 7.853e-7 m^2/s and [. . .] 6e-7 m^2/s.

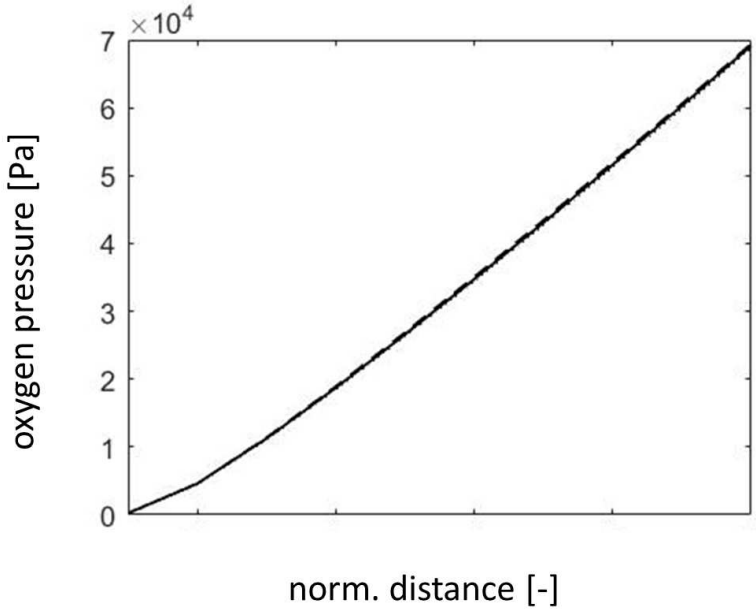


Figure 13.40: Simulation Task 2. Effect of the Knudsen diffusion coefficient of steam water ($D_{H_2O,lK}$) on the oxygen pressure along the catalyst layer for high current: [- - -] 1e-5 m^2/s, [——] 1.047e-6 m^2/s and [. . .] 1e-7 m^2/s.

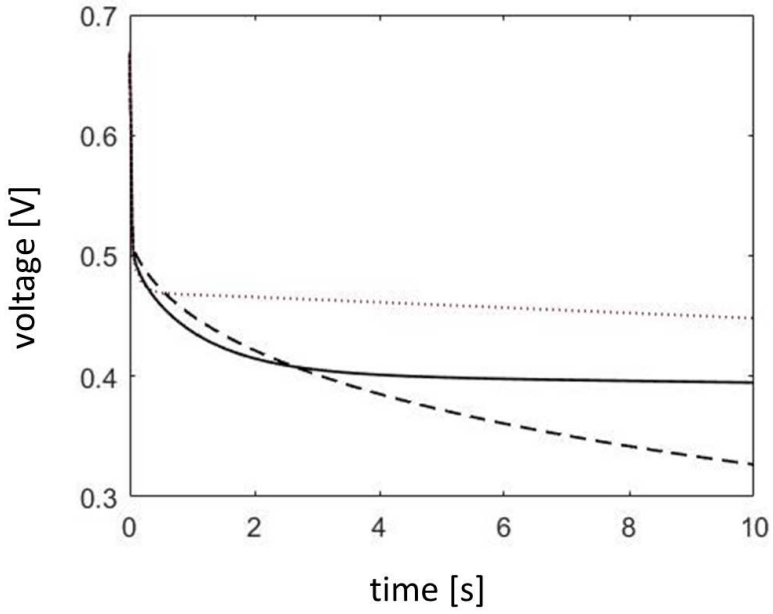


Figure 13.41: Simulation Task 3. Effect of the diffusion layer length (L_{dif}) on the transient response of the fuel cell voltage for high current: [- - -] 0.01 m, [——] 0.0016 m and [. . .] 0.0005 m.

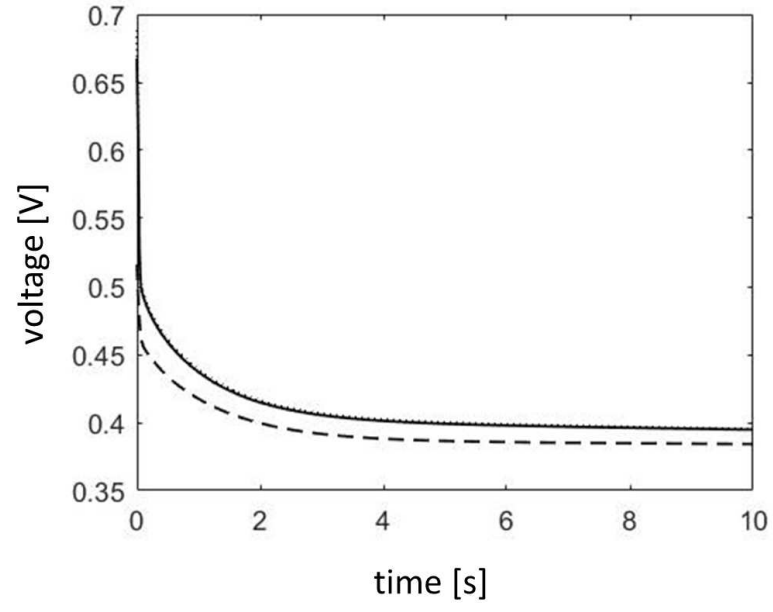


Figure 13.42: Simulation Task 3. Effect of the membrane length (L_{mem}) on the transient response of the fuel cell voltage for high current: [- - -] 8e-4 m, [——] 8e-5 m and [. . .] 8e-6 m.

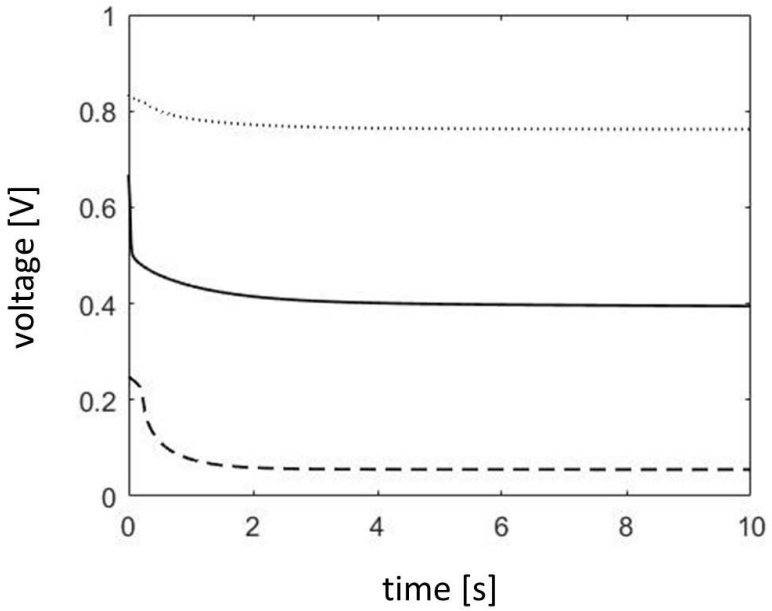


Figure 13.43: Simulation Task 3. Effect of the catalyst layer length (L_{cat}) on the transient response of the fuel cell voltage for high current: [- - -] 4e-4 m, [——] 4e-5 m and [. . .] 4e-6 m.

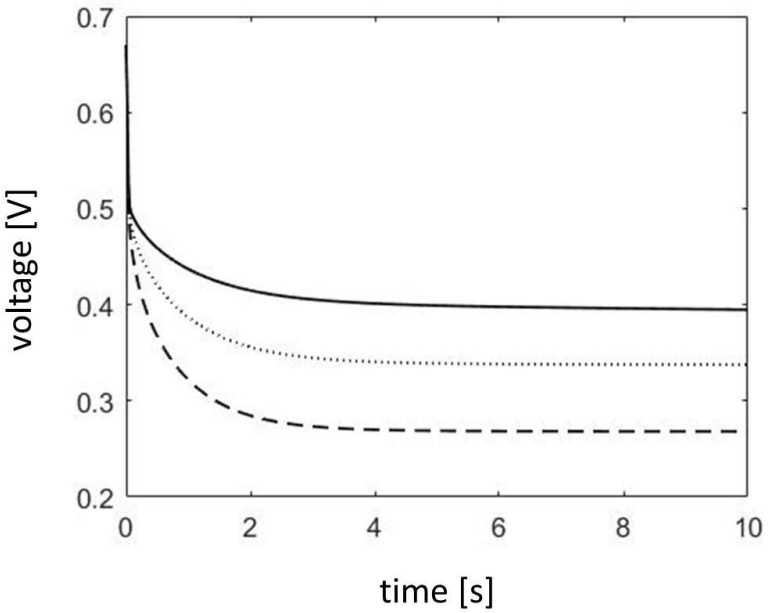


Figure 13.44: Simulation Task 3. Effect of the solid porosity of the diffusion layer (ε_s) on the transient response of the fuel cell voltage for high current: [- - -] 0.7, [. . .] 0.5 and [——] 0.1.

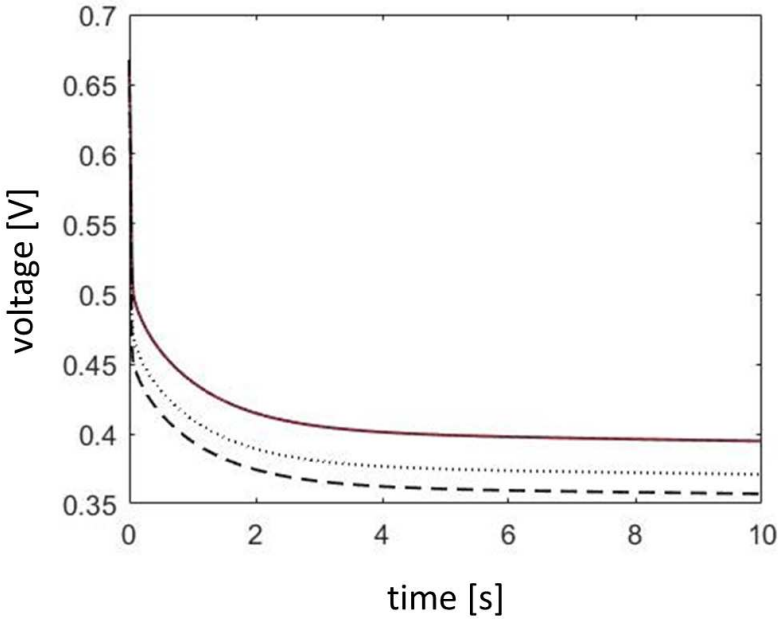


Figure 13.45: Simulation Task 3. Effect of temperature of the three layers (T_g) on the transient response of the fuel cell voltage for high current: [- - -] 390 K, [. . .] 370 K and [——] 340 K.

Bibliography

Abdel-Rahman, A. (2008), On the atmospheric dispersion and gaussian plume model, *in* '2^{nd} Intl. Conference on waste management, water-pollution air pollution, indoor climate, WWA'08'.

Bernardi, D. M. & Verbrugge, M. W. (1992), 'A mathematical model of the solid-polymer-electrolyte fuel cell', *J. Electrochem. Soc.* (139), 2477–2491.

Bevers, D., Wöhr, M. & Yasuda, K. (1997), 'Simulation of polymer electrolyte fuel cell electrode', *J. Appl. Electrochem. Soc.* (27), 1254–1264.

Broenink, J. F. (1999), *Introduction to Physical Systems Modelling with Bond Graphs*. Accessed January, 2018.
URL: *www.ram.ewi.utwente.nl/bnk/papers/BondGraphsV2.pdf*

Broka, K. & Ekdunge, P. (1997), 'Modelling the PEM fuel cell cathode', *J.Appl. Electrochem. Soc.* (27), 281–289.

Brusca, S., Famoso, F., Lanzafame, R., Mauro, S., Marino, A. & Monforte, P. (2016), Theoretical and experimental study of gaussian plume model in small scale system, *in* '71^{st} Conference of the Italian Thermal Machines Engineering Association, ATI2016'.

Cutlip, M. B. & Shacham, M. (1999), *Problem Solving in Chemical Engineering with Numerical Methods*, Prentice-Hall.

Downs, J. J. & Vogel, E. F. (1993), 'A plant-wide industrial process control problem', *Computers and Chemical Engineering* **17**(3), 245–255.

Fuel Cell Handbook (2004), U.S. Department of Energy. Free download: https://www.netl.doe.gov.

Ilachinski, A. (2001), *Cellular Automata: A Discrete Universe*, World Scientific, Singapore.

Karnopp, D., Margolis, D. & Rosenberg, R. (1990), *System Dynamics: A Unified Approach*, John Wiley & Sons. Second edition.

Larminie, J. & Dicks, A. (2000), *Fuel cell systems explained*, John Wiley and Sons.

Martin, D. O. (1976), 'The change of concentration standard deviations with distance', *J. Air Pollution Control Assoc.* **26**(2), 145–147.

Martin-Villalba, C., Manzur, M. & Urquia, A. (2018), 'Virtual lab in Modelica for air pollution control', *Computer Tools in Education* (1), 5–15.

O'Hayre, R., Colella, W., Prinz, F. B. & Cha, S. W. (2006), *Fuel cell fundamentals*, John Wiley and Sons.

Ricker, N. L. (1993), 'Model predictive control of a continuous, nonlinear, two-phase reactor.', *Journal of Process Control* **3**, 109–123.

Rubio, M. A., Urquia, A. & Dormido, S. (2010), 'Dynamic modelling of PEM fuel cells using the FuelCellLib Modelica library', *Mathematical and Computer Modelling of Dynamical Systems* (16), 165–194.

Schiff, J. L. (2008), *Cellular Automata: A Discrete View of the World*, Wiley-Interscience, New York, USA.

Springer, T. E. & Zawodzinsky, T. A. (1991), 'Polymer electrolyte fuel cell model', *J. Electrochem. Soc.* (138), 2334–2342.

Urquia, A. (2000), *Modelado Orientado a Objetos y Simulación de Sistemas Híbridos en el Ámbito del Control de Procesos Químicos*, PhD Diss., UNED, Madrid, Spain.

Urquia, A. & Martin-Villaba, C. (2018), *Modeling and simulation in Engineering using Modelica*, Editorial UNED.

von Neumann, J. (1966), *Theory of self-reproducing automata*, Univ. of Illinois Press, Urbana and London.

Wolfram, S. (2002), *A New Kind of Science*, Wolfram Media Inc., Champain, IL, USA.

This activity book is aimed to provide an introduction to the simulation practice in Engineering using Modelica. To this end, we propose a series of thirteen independent hands-on assignments of increasing complexity. Each assignment contains the description of a system and a mathematical model of the system's behavior. The proposed task often consists in describing this mathematical model in the Modelica language and simulate it. In some assignments, the system's behavior is described as an atomic model, without internal structure. Some other assignments ask to design and implement a model library, and to compose the system model by instantiating and connecting components from this model library.

Before start working with this activity book, it is advisable to read its companion theory book: a free e-book entitled *Modeling and simulation in Engineering using Modelica*, written by Alfonso Urquía and Carla Martín, and published by Editorial UNED in 2018 (ISBN: 9788436273090). The theory book provides all the previous knowledge on the Modelica language required to complete the assignments.

Alfonso Urquía, **Carla Martín**, **Miguel Ángel Rubio** and **Victorino Sanz** are professors in the Departamento de Informática y Automática, at the Universidad Nacional de Educación a Distancia (UNED) in Madrid, Spain; and members of the research group on Modelling & Simulation in Control Engineering of UNED. Further information is available at www.euclides.dia.uned.es

This book has been written during the course of the Erasmus+ project "InMotion - Innovative teaching and learning strategies in open modelling and simulation environment for student-centered engineering education", Project No. 573751-EPP-1-2016-1-DE-EPPKA2-CBHE-JP, co-funded by the Erasmus+ Programme of the European Union. The European Commission support for the production of this book does not constitute an endorsement of the contents, which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Co-funded by the Erasmus+ Programme of the European Union

Innovative teaching and learning strategies in open modelling and simulation environment for student-centered engineering education
573751-EPP-1-2016-1-DE-EPPKA2-CBHE-JP

InMotion

UNED Editorial

Juan del Rosal, 14
28040 MADRID
Tel. Dirección Editorial: 913 987 521