User's guide for the Open Hydropower Library (OpenHPL)

Liubomyr Vytvytskyi

Porsgrunn, 8th August 2019

Contents

1 Introduction

OpenHPL is an open-source hydropower library that consists of hydropower unit models and is encoded in Modelica. Modelica is a multi-domain as well as a component-oriented modelling language that is suitable for complex system modelling. In order to develop the library, OpenModelica has been used as an open-source Modelica-based modelling and simulation environment.

A hydropower library: *OpenHPL* provides the capability for the modelling of hydropower systems of different complexity. The library includes the following units:

1. Various waterway units are modelled based on the mass and momentum balances, i.e., reservoirs, conduits, surge tank, fittings. A modern method for solving more detailed models (PDEs) is implemented in the library, and enables the modelling of the waterway with elastic walls and compressible water as well as open channel.
2. A hydrology model has been implemented and makes it possible to simulate the water inflow to the reservoirs.
3. Mechanistic models, as well as simple look-up table turbine models are implemented for the Francis and Pelton turbine types. The Francis turbine model also includes a turbine design algorithm that gives all of the needed parameters for the model, based on the turbine's nominal operating values.
4. The capability for multiphysics connections and work with other libraries is ensured, e.g., connecting with the Open-Instance Power System Library *OpenIPSL* makes it possible to model the electrical part for the hydropower system.

A detailed description of each hydropower unit and their uses are presented below in this user guide.

2 Installation

OpenHPL can be opened either in open-source OpenModelica[1] or commercial Dymola[2] modelling and simulation environments, which are based on the Modelica language. Here, OpenModelica is emphasized due to free availability. To install OpenModelica, follow the instructions at `https://openmodelica.org/download/download-windows` for Windows users, or find the installation instruction for other operating systems at `https://openmodelica.org` in "Download" tab. Some tutorials exist for Modelica at `http://book.xogeny.com`, and for OpenModelica at `https://goo.gl/76274H`.

OpenHPL can be found at Github[3]. To install this library, follow the instructions at the Github page.

In addition, Modelica models in OpenModelica can be simulated within a scripting language (Python[4] via the OMPython API[5], Julia[6] via the OMJulia API[7]) and further analysed using the analysis tools in the scripting language. The installation instructions for both these APIs can be found in the links provided in the footnotes for each API.

[1] `https://openmodelica.org`
[2] `https://www.3ds.com/products-services/catia/products/dymola`
[3] `https://github.com/usn-modelling/HydroPowerUSN`
[4] `https://www.python.org`
[5] `https://www.openmodelica.org/doc/OpenModelicaUsersGuide/latest/ompython.html`
[6] `https://julialang.org`
[7] `https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omjulia.html`

3 OpenHPL elements

An overview of each element of the hydropower library *OpenHPL* is provided in this section. A screenshot of *OpenHPL* in OpenModelica is shown in Figure 3.1.

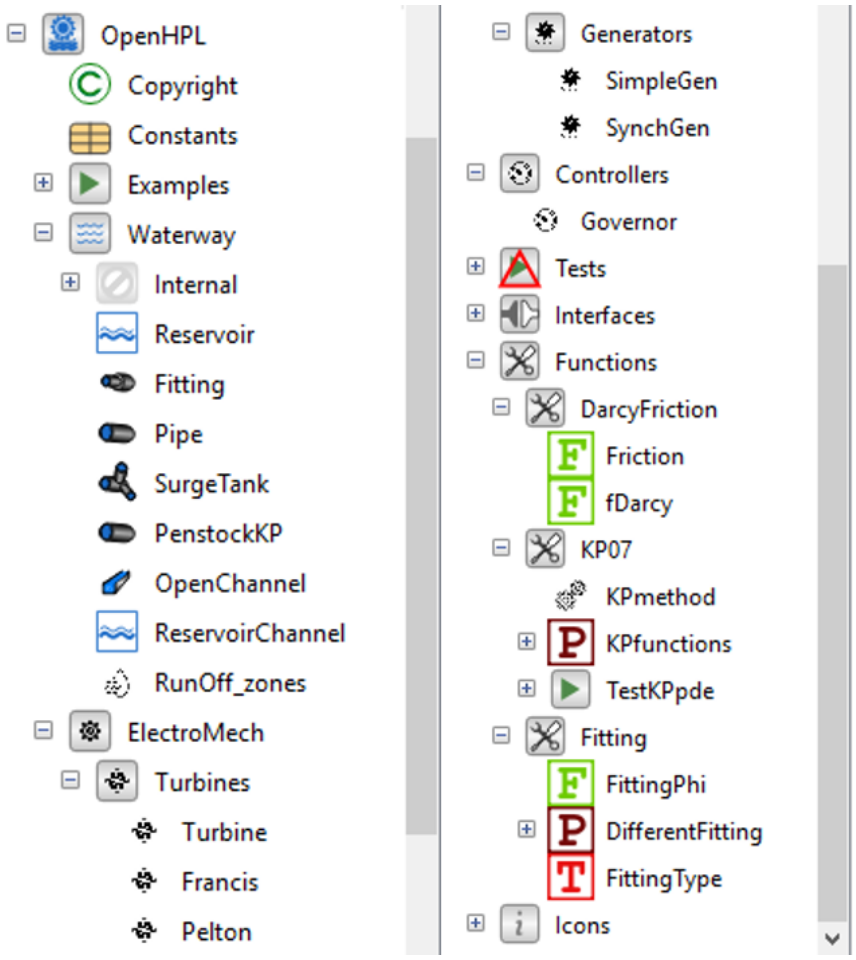

Figure 3.1: Screen shot of OpenModelica with the hydropower library.

It is visible from Figure 3.1 that the library is divided into various components and classes. Each of these components and classes is listed below with a short description:

- First, the *Copyright* element provides a reference to the license for this library.

- Next, the *Constants* element is a record's model that determines the common parameters for this library. It is possible to insert this class to models and use the common parameters for the whole library.

- Then, the *Examples* class provides various examples of using the library for hydropower system as well as examples of using *OpenHPL* together with power system library — *OpenIPSL*.

- The *Waterway* class consists of various unit models for the waterway of the hydropower system, such as reservoirs, conduits, surge tank, pipe fittings, etc.

- The *ElectroMech* class provides the electro-mechanical components of the hydropower system and consists of two main sub-classes: *Turbines* with various turbine unit models and *Generators* with models for a synchronise generator.

- Then, the *Controllers* class holds a simple model for a governor of the hydropower system.

- The *Tests* class provides various testing models for all library components.

- Next, the *Interfaces* class gives connector models for the library components.

- The *Functions class* consists of three sub-classes that define functions for the calculation of a friction term in the pipe — *DarcyFriction*, for solving PDEs using Kurganov-Petrova (KP) scheme — *KP07*, and for pressure drop calculation in various pipe fitting — *Fitting*.

- Finally, the *Icons* class holds icons for all library components.

Below, a detailed description of each unit model of the *OpenHPL* is provided.

3.1 Interfaces

First, a detailed description of the interface connectors is provided here. In the *OpenHPL*, two types of connectors are typically used.The first type is the standard Modelica real input/output connector, the other type is a set of connectors that represent the water flow and are modelled similar to the connection in an electrical circuit with voltage and current, or similar to the idea of potential and flow in Bond Graph models. The water flow connector which is called *Contact* in the library, contains information about the pressure in the connector and mass flow rate that flows through the connector. An example of a Modelica code for defining the *Contact* connector looks as follows:

```
connector Contact "Water flow connector"
  Modelica.SIunits.Pressure p "Contact pressure";
  flow Modelica.SIunits.MassFlowRate m_dot "Mass flow rate
     through the contact";
  // Creating an icon for connector
  annotation (Icon(graphics={  Ellipse(extent
     ={{-100,-100},{100,100}}, lineColor = {28, 108, 200},
     fillColor = {0, 128, 255}, fillPattern =
     FillPattern.Solid)}));
end Contact;
```

In addition, some extensions of this water flow connector are developed for the better use in the library. These extensions are listed hereby.

- *TwoContact* is an extension from the *Contact* model which provides a model of two connectors of inlet and outlet contacts. A Modelica code for this model looks as follows:

```
partial model TwoContact "Model of two connectors"
  // Specifying connectors and their placement in
     diagram
  Contact p "Inlet contact" annotation(Placement(
     transformation(extent={{-110,-10},{-90,10}})));
  Contact n "Outlet contact" annotation(Placement(
     transformation(extent={{90,-10},{110,10}})));
end TwoContact;
```

- *ContactPort* is an extension from the *TwoContact* model which also provides information about a mass flow rate between these two connectors. The mass flow rate that flows through the inlet connector is equal to the mass flow through the outlet connector. This model is used for the pipe modelling. A Modelica code for this *ContactPort* model looks as follows:

```
partial model ContactPort "Model of two connectors
   with mass flow rate"
  Modelica.SIunits.MassFlowRate m_dot "Mass flow rate
     ";
  extends TwoContact;
equation
  0 = p.m_dot + n.m_dot;
  m_dot = p.m_dot;
end ContactPort;
```

- *ContactNode* is an extension from the *TwoContact* model and provides a node pressure that is equal to the pressures from these two connectors. This model also defines the mass flow rate that is the sum of the mass flow rates through the inlet and outlet connectors. This model is used for the surge tank modelling. A Modelica code for this *ContactNode* model looks as follows:

```
partial model ContactNode "Model of two connectors
   and node pressure"
  Modelica.SIunits.Pressure p_n "Node pressure";
  Modelica.SIunits.MassFlowRate m_dot "Mass flow rate
     ";
  extends TwoContact;
equation
  p_n = p.p;
  p.p = n.p;
  m_dot = p.m_dot + n.m_dot;
end ContactNode;
```

- *TurbineContacts* is an extension from *ContactPort* model and provides the real input and output connectors, additionally. This model is used for turbine modelling. A Modelica code for this *TurbineContacts* model looks as follows:

```
partial model TurbineContacts "Model of turbine
   connectors"
  extends ContactPort;
  // Specifying additional connectors and their
     placement in diagram
  input Modelica.Blocks.Interfaces.RealInput u_t "[
     Guide vane|nozzle] opening of the turbine"
     annotation (Placement(transformation(extent =
     {{-20, -20}, {20, 20}}, rotation = -90, origin
     ={0,120})));
  Modelica.Blocks.Interfaces.RealOutput P_out "
     Mechanical Output power" annotation (Placement(
     transformation(origin={0,-110}, extent
     ={{-10,-10},{10,10}}, rotation = 270)));
end TurbineContacts;
```

3.2 Functions

Here, a detailed description of the functions and their used algorithms in the library, are presented.

3.2.1 Friction term

First, the functions for defining the friction force in the waterway are described. More details can be found in Bernt Lie's Lecture notes, [1].

The friction force F_f is directed in the opposite direction of the velocity v (the linear velocity average across the cross-section of the pipe) of the fluid, [1]. A common expression for friction force in the filled pipes is the following:

$$F_f = -\frac{1}{8}\pi\rho L D f_D v|v| \tag{3.1}$$

Here, L and D are related to the pipe width and diameter, respectively. f_D is a Darcy friction factor that is a function of Reynolds' number N_{Re}, with the roughness ratio $\frac{\varepsilon}{D}$ as a parameter, see Figure 3.2.

In Figure 3.2, the turbulent region ($N_{Re} > 2.3 \cdot 10^3$) is a flow regime where the velocity across the pipe has a stochastic nature, and where the velocity v is relatively uniform across the pipe when we average the velocity over some short period of time. The laminar region ($N_{Re} < 2.1 \cdot 10^3$) is a flow regime with a regular velocity v which varies as a parabola with the radius of the pipe, with zero velocity at the pipe wall and maximal velocity at the centre of the pipe.

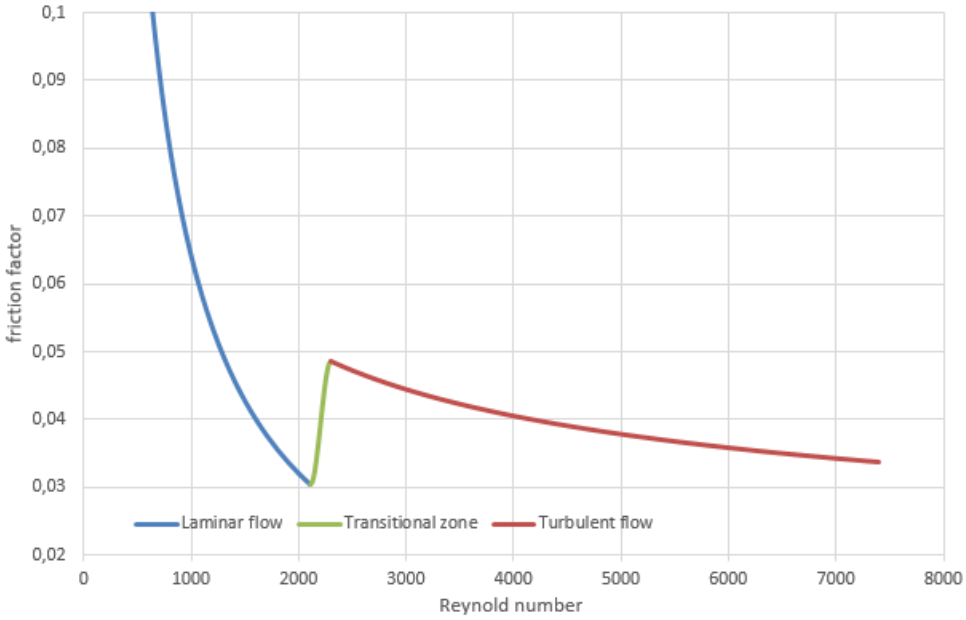

Figure 3.2: Darcy friction factor as a function of Reynolds' number.

Darcy friction factor varies with the roughness of the pipe surface, specified by roughness height ε. For laminar flow in a cylindrical pipe ($N_{Re} < 2.1 \cdot 10^3$), the Darcy friction factor

f_{D} can be found using the following expression:

$$f_{\mathrm{D}} = \frac{64}{N_{\mathrm{Re}}} \tag{3.2}$$

Here, Reynolds' number is found as follows: $N_{\mathrm{Re}} = \frac{\rho|v|D}{\mu}$, where μ is the the fluid viscosity.

For turbulent flow ($N_{\mathrm{Re}} > 2.3 \cdot 10^3$), it is common to rewrite the expression for the Darcy friction factor as

$$f_{\mathrm{D}} = \frac{1}{\left(2\log_{10}\left(\frac{\varepsilon}{3.7D} + \frac{5.74}{N_{\mathrm{Re}}^{0.9}}\right)\right)^2} \tag{3.3}$$

In order to define the Darcy friction factor in a region between laminar and turbulent flow regimes, a possibility is to use some interpolation expressions between the laminar value at $N_{\mathrm{Re}} = 2100$ and the turbulent value at $N_{\mathrm{Re}} = 2300$, e.g., a cubic polynomial fitting with the same slope as laminar friction at $N_{\mathrm{Re}} = 2100$ and turbulent friction at $N_{\mathrm{Re}} = 2300$, [1]. To achieve the global differentiability, with $p(N_{\mathrm{Re}}) = aN_{\mathrm{Re}}^3 + bN_{\mathrm{Re}}^2 + cN_{\mathrm{Re}} + d$, thus:

$$\begin{aligned}
p(N_{\mathrm{Re}} = 2100) &= f_{\mathrm{D}}^{\mathrm{l}}(N_{\mathrm{Re}} = 2100) \\
p(N_{\mathrm{Re}} = 2300) &= f_{\mathrm{D}}^{\mathrm{t}}(N_{\mathrm{Re}} = 2300) \\
\frac{dp}{dN_{\mathrm{Re}}}\Big|_{N_{\mathrm{Re}}=2100} &= \frac{df_{\mathrm{D}}^{\mathrm{l}}}{dN_{\mathrm{Re}}}\Big|_{N_{\mathrm{Re}}=2100} \\
\frac{dp}{dN_{\mathrm{Re}}}\Big|_{N_{\mathrm{Re}}=2300} &= \frac{\partial f_{\mathrm{D}}^{\mathrm{t}}}{\partial N_{\mathrm{Re}}}\Big|_{\frac{\varepsilon}{D}, N_{\mathrm{Re}}=2300}
\end{aligned} \tag{3.4}$$

Hence, the constants a, b, c and d can be found as follows:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2100^3 & 2100^2 & 2100 & 1 \\ 2300^3 & 2300^2 & 2300 & 1 \\ 3\cdot 2100^2 & 2\cdot 2100 & 1 & 0 \\ 3\cdot 2300^2 & 2\cdot 2300 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} \frac{64}{2100} \\ \frac{1}{\left(2\log_{10}\left(\frac{\varepsilon}{3.7D} + \frac{5.74}{2300^{0.9}}\right)\right)^2} \\ -\frac{64}{2100^2} \\ -0.25\frac{0.316}{2300^{1.25}} \end{bmatrix} \tag{3.5}$$

Based on the presented equation for calculation of the friction force in the waterway, two functions are encoded in this class *DarcyFriction*. The first function is for defining the Darcy friction factor and called *fDarcy*. This function has the following inputs: the Reynolds' number N_{Re}, the pipe diameter D, and the pipe roughness height ε. Then, based on Eq. 3.2 for the laminar flow (Reynold number < 2100), Eq. 3.3 for turbulent flow (Reynold number > 2300), and Eq. 3.5 for transitional zone ($2100 <$ Reynold number < 2300); the *fDarcy* function provides value for the Darcy friction factor f_{D}.

Another function, *Friction* is for defining the actual friction force and is based on a response from the *fDarcy* function. This function has the following inputs: the linear velocity v, the pipe length and diameter L and D, the liquid density and viscosity ρ and

μ, and the pipe roughness height ε. As an output, this function provides a value for the friction force F_f based on Eq. 3.1. An example of a Modelica code for defining the *Friction* function looks as follows:

```
function Friction "Friction force with Darcy friction factor"
  import Modelica.Constants.pi;
  input Modelica.SIunits.Velocity v "Flow velocity";
  input Modelica.SIunits.Diameter D "Pipe diameter";
  input Modelica.SIunits.Length L "Pipe length";
  input Modelica.SIunits.Density rho "Density";
  input Modelica.SIunits.DynamicViscosity mu "Dynamic
     viscosity of water";
  input Modelica.SIunits.Height eps "Pipe roughness height";
  // Function output (response) value
  output Modelica.SIunits.Force F_f "Friction force";
// Local (protected) quantities
protected
  Modelica.SIunits.ReynoldsNumber N_Re "Reynold number";
  Real f "friction factor";
algorithm
  N_Re := rho * abs(v) * D / mu;
  f := fDarcy(N_Re, D, eps);
  F_f := 0.5 * pi * f * rho * L * v * abs(v) * D / 4;
end Friction;
```

3.2.2 KP scheme

Here, functions for solving PDEs in Modelica are described. First, the overview of the KP scheme is presented. More details about this scheme can be found in Roshan Sharma work, [2], and other works, [3], [4].

This is a well-balanced second-order scheme, which is a Reimann problem solver free scheme (central scheme) while at the same time, it takes advantage of the upwind scheme by utilizing the local, one side speed of propagation (given by the eigenvalues of the Jacobian matrix) during the calculation of the flux at the cell interfaces, [2].

The central-upwind numerical scheme is presented for the one-dimensional case.

$$\frac{\partial U(x,t)}{\partial t} + \frac{\partial F(x,t,U)}{\partial x} = S(x,t,U) \tag{3.6}$$

Here, $U(x,t)$ is the state vector, where states are the functions of position x and time t. $F(x,t,U)$ is the vector of fluxes and $S(x,t,U)$ is the source terms.

In order to solve this PDE, it should be first discretized by finite-volume methods. With the finite volume method, we divide the grid into small control volumes/cells and then apply the conservation laws. This control volume/cell with notations are shown in Figure 3.3.

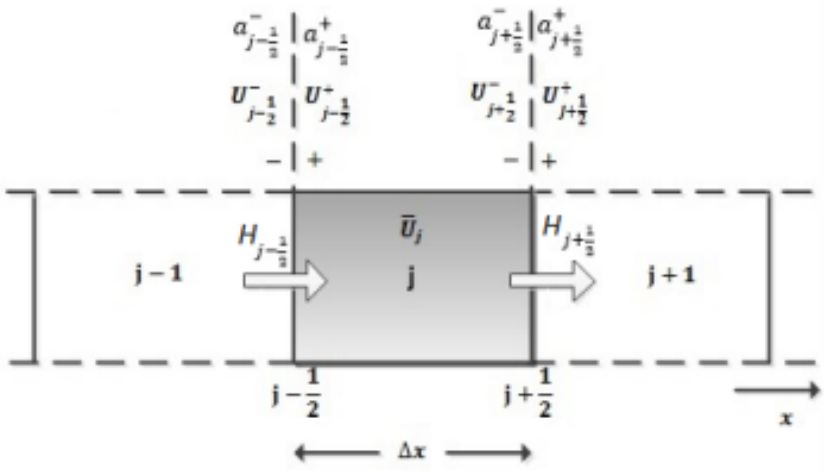

Figure 3.3: Control volume/cell, [2].

Hence, the semi-discrete (time-dependent ODEs) central-upwind scheme can be then written in the following form:

$$\frac{d}{dt}\bar{U}_j(t) = -\frac{H_{j+\frac{1}{2}}(t) - H_{j-\frac{1}{2}}(t)}{\Delta x} + \bar{S}_j(t) \tag{3.7}$$

Here, $\bar{U}_j$ are the cell centre average values, while $H_{j\pm\frac{1}{2}}(t)$ are the central upwind numerical fluxes at the cell interfaces and are given by:

$$\begin{aligned} H_{j+\frac{1}{2}}(t) &= \frac{a^+_{j+\frac{1}{2}} F\left(U^-_{j+\frac{1}{2}}\right) - a^-_{j+\frac{1}{2}} F\left(U^+_{j+\frac{1}{2}}\right)}{a^+_{j+\frac{1}{2}} - a^-_{j+\frac{1}{2}}} + \frac{a^+_{j+\frac{1}{2}} a^-_{j+\frac{1}{2}}}{a^+_{j+\frac{1}{2}} - a^-_{j+\frac{1}{2}}}\left[U^+_{j+\frac{1}{2}} - U^-_{j+\frac{1}{2}}\right] \\ H_{j-\frac{1}{2}}(t) &= \frac{a^+_{j-\frac{1}{2}} F\left(U^-_{j-\frac{1}{2}}\right) - a^-_{j-\frac{1}{2}} F\left(U^+_{j-\frac{1}{2}}\right)}{a^+_{j-\frac{1}{2}} - a^-_{j-\frac{1}{2}}} + \frac{a^+_{j-\frac{1}{2}} a^-_{j-\frac{1}{2}}}{a^+_{j-\frac{1}{2}} - a^-_{j-\frac{1}{2}}}\left[U^+_{j-\frac{1}{2}} - U^-_{j-\frac{1}{2}}\right] \end{aligned} \tag{3.8}$$

Here, $a^\pm_{j\pm\frac{1}{2}}$ are the one-sided local speeds of propagation.

For calculating the numerical fluxes $H_{j\pm\frac{1}{2}}(t)$, the values of states at the cell interfaces $U^\pm_{j\pm\frac{1}{2}}$ are needed. These values can be calculated as the endpoints of a piecewise linearly

reconstructed function:

$$\begin{aligned} U^{-}_{j+\frac{1}{2}} &= \bar{U}_j + \tfrac{\Delta x}{2} s_j \\ U^{+}_{j+\frac{1}{2}} &= \bar{U}_{j+1} - \tfrac{\Delta x}{2} s_{j+1} \\ U^{-}_{j-\frac{1}{2}} &= \bar{U}_{j-1} + \tfrac{\Delta x}{2} s_{j-1} \\ U^{+}_{j-\frac{1}{2}} &= \bar{U}_j - \tfrac{\Delta x}{2} s_j \end{aligned} \tag{3.9}$$

The slope s_j of the reconstructed function in each cell is computed using a limiter function to obtain a non-oscillatory nature of the reconstruction. The KP scheme utilizes the generalized *minmod* limiter as:

$$s_j^- = \theta \tfrac{\bar{U}_j - \bar{U}_{j-1}}{\Delta x}, s_j^c = \tfrac{\bar{U}_{j+1} - \bar{U}_{j-1}}{2\Delta x}, s_j^+ = \theta \tfrac{\bar{U}_{j+1} - \bar{U}_j}{\Delta x}$$

$$s_j = minmod\left(s_j^-, s_j^c, s_j^+\right) = \begin{cases} \min\left(s_j^-, s_j^c, s_j^+\right), & \text{if } s_j^- > 0 \;\&\; s_j^c > 0 \;\&\; s_j^+ > 0 \\ \max\left(s_j^-, s_j^c, s_j^+\right), & \text{if } s_j^- < 0 \;\&\; s_j^c < 0 \;\&\; s_j^+ < 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.10}$$

The parameter $\theta \in [1,2]$ is used to control or tune the amount of numerical dissipation or numerical viscosity present in the resulting scheme. The value of $\theta = 1.3$ is an acceptable starting point in general.

It can be observed that for a given j^{th} cell, the information from the neighbouring cells $j-1$ and $j-2$ (to the left) and $j+1$ and $j+2$ (to the right) are required for calculating the flux integrals. This will pose difficulties at the cells on the left and right boundaries. While evaluating the flux integrals near the left boundary cells ($j = 1$ and $j = 2$) and near the right boundary cells ($j = N-1$ and $j = N$; N is the number of cells in the grid), imaginary cells that lie outside the physical boundary should also be taken into consideration, see Figure 3.4.

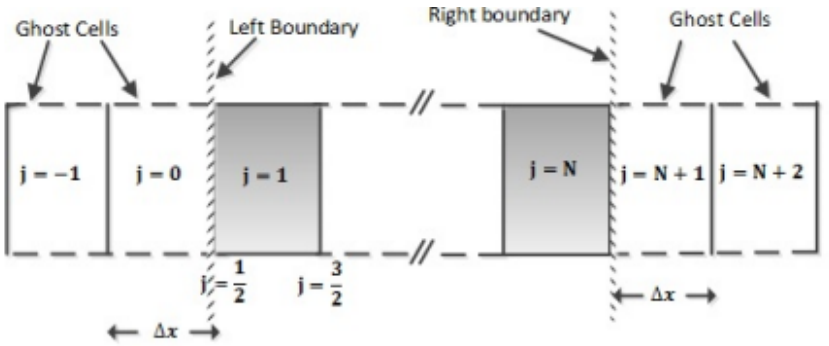

Figure 3.4: Ghost cells at the grid boundaries, [2].

These imaginary cells denoted by $j = 0$ and $j = -1$ on the left, and $j = N+1$ and $j = N+2$ on the right are called the ghost cells. The average value of the conserved variables at

the centre of these ghost cells depends on the nature of the physical boundary taken into account. These ghosts cells can be defined in the following way:

$$\begin{aligned} \bar{U}_{j=0} &= 2\bar{U}_{j=1} - \bar{U}_{j=2} \\ \bar{U}_{j=-1} &= 2\bar{U}_{j=0} - \bar{U}_{j=1} \\ \bar{U}_{j=N+1} &= 2\bar{U}_{j=N} - \bar{U}_{j=N-1} \\ \bar{U}_{j=N+2} &= 2\bar{U}_{j=N+1} - \bar{U}_{j=N} \end{aligned} \tag{3.11}$$

The one-sided local speeds of propagation can be estimated as the largest and the smallest eigenvalues $\lambda_{1,2}$ of the Jacobian $\frac{\partial F}{\partial U}$ of the system as:

$$\begin{aligned} a^{+}_{j\pm\frac{1}{2}} &= \max\left(\lambda^{+}_{1,j\pm\frac{1}{2}}, \lambda^{-}_{1,j\pm\frac{1}{2}}, 0\right) \\ a^{-}_{j\pm\frac{1}{2}} &= \max\left(\lambda^{+}_{2,j\pm\frac{1}{2}}, \lambda^{-}_{2,j\pm\frac{1}{2}}, 0\right) \end{aligned} \tag{3.12}$$

Lastly, the source term $\bar{S}_j(t)$ has to be appropriately discretized to ensure the well-balanced method. This can be written as:

$$\bar{S}_j(t) = S\left(\bar{U}_j\right) \tag{3.13}$$

Hence, the separate functions for each of these elements defining in Eqs. 3.8-3.12 are modelled and encoded in *OpenHPL*. These functions are assembled in the *KPfunctions* folder in class *KP07*, and look as follows:

- *GhostsCell* function provides values of the conserved variables at the centre of the ghost cells, using Eq. 3.11. As an input piece of information this function receives the number of cells N, and the state vector with the cell centre average values $\bar{U}_{j=1..N}$. Then, the *GhostsCell* function returns a state vector with the cell centre average values for all (including ghost) cells $\bar{U}_{j=-1..N+2}$.
- *SlopeVectoreS* function returns the slope vector $s_{j=0..N+1}$ of the reconstructed function for each cell, using Eq. 3.10. This function has the following inputs: the number of cells N, parameters θ and Δx, and the state vector with the cell centre average values $\bar{U}_{j=-1..N+2}$.
- Then, function *WiseU* is created to define the values of states $U^{\pm}_{j\pm\frac{1}{2}}$ as the endpoints of a piecewise linearly reconstructed function from Eq. 3.9 using the two previous functions. This function has the following inputs: the number of cells N, parameters θ and Δx, condition and values for the boundaries, and the state vector with the cell centre average values $\bar{U}_{j=1..N}$.
- Another function as the *SpeedPropagationApipe* provides the one-sided local speeds of propagation $a^{\pm}_{j\pm\frac{1}{2}}$, using Eq. 3.12. As an input information this function receives the number of cells N and vectors of eigenvalues $\lambda^{\pm}_{1,j\pm\frac{1}{2}}$ and $\lambda^{\pm}_{2,j\pm\frac{1}{2}}$ of the Jacobian of the system.

- The last function *FluxesH* in the *KPfunctions* folder, defines the central upwind numerical fluxes at the cell interfaces $H_{j\pm\frac{1}{2}}$, using Eq. 3.8. This function has the following inputs: the number of cells N, the values of states at the cell interfaces $U^{\pm}_{j\pm\frac{1}{2}}$, the one-sided local speeds of propagation $a^{\pm}_{j\pm\frac{1}{2}}$, and the vector of fluxes $F\left(U^{\pm}_{j\pm\frac{1}{2}}\right)$.

Then, the primary function for the KP scheme *KPmethod* is created which uses the last three presented functions to define the right-hand side of Eq. 3.7 (discretization solution of PDE). As an input piece of information, this function receives the number of cells N, parameters θ and Δx, the state vector with the cell centre average values $\bar{U}_{j=1..N}$, vectors of eigenvalues $\lambda^{\pm}_{1,j\pm\frac{1}{2}}$ and $\lambda^{\pm}_{2,j\pm\frac{1}{2}}$ of the Jacobian of the system, the vector of fluxes $F\left(U^{\pm}_{j\pm\frac{1}{2}}\right)$ and source terms $\bar{S}_j$, and condition and values for the boundaries. It should be noted that the *KPmethod* function is encoded for the cases of systems with two states (state vector $\bar{U}_{j=1..N}$ consists of two states) that is common for the detailed model of the pipe or open channel model (see those unit models below). The boundaries are specified with the inlet and outlet state values: either inlet (or: outlet) values for both states, or inlet and outlet values for one of the states.

In the case with the use of the KP scheme for the open channel model [2], [3], one of the states should be processed through the scheme with some additional vector that is ensured in this *KPmethod* function ($B_{j=-1..N+2}$ vector is also input to the functions *KPmethod* and *WiseU*).

It should be noted that due to the issues of the simulation speed, all of the presented functions in class *KP07* are implemented as the *model* type in OpenModelica instead of the *function* type. An example of a Modelica code for defining the *KPmethod* function looks as follows:

```
model KPmethod
  extends Icons.Method;
  parameter Integer N "number of segments";
  input Real U[2 * N] "state vector",
            dx "length step",
            theta = 1.3 "parameter for slope limiter",
            S_[2 * N] "source term vector S",
            F_[2 * N, 4] "vector F",
            lam1[N, 4] "matrix of eigenvalues '+'",
            lam2[N, 4] "matrix of eigenvalues '-'",
            B[N + 4] = zeros(N + 4) "additional for open
               channel",
            boundary[2, 2] "values for boundary conditions";
```

```
  input Boolean boundaryCon[2, 2] "boundary conditions
    consideration";
  output Real diff_eq[2 * N] "right hand side for KP solution
    ";
  Real U_[8, N] "matrix with boundary state values. Can be
    extracted";
protected
  Real H_[2 * N, 2] "matrix of fluxes",
       A_speed[N, 4] "matrix of one-side local speeds
         propagation";
public
  KPfunctions.WiseU wiseU(N = N, theta = theta, U = U, B = B,
      dx = dx, boun = boundary, bounCon = boundaryCon) "use
    function for defing the piece wise linear reconstruction
    of vector U";
  KPfunctions.SpeedPropagationApipe speedA(N = N, lamda1 =
    lam1, lamda2 = lam2) "use function for defing the
    one-side local speeds propagation";
  KPfunctions.FluxesH fluxesH(N = N, U_ = U_, A_ = A_speed,
    F_ = F_) "use function for defing the central upwind
    numerical fluxes";
equation
  ///// piece wise linear reconstruction of vector U
  U_ = wiseU.U_;
  ///// one-side local speeds propagation
  A_speed = speedA.A;
  ///// central upwind numerical fluxes
  H_ = fluxesH.H;
  //// right hand side of diff. equation
  diff_eq = (-(H_[:, 1] - H_[:, 2]) / dx) + S_;
end KPmethod;
```

Examples of using the KP scheme for solving PDEs are also provided in the class *KP07* in the *TestKPpde* folder. More information about using the *KPmethod* function is presented below in the waterway modelling section for the *PenstockKP* and *OpenChannel* units.

3.2.3 Fitting

The functions for defining the pressure drop in various pipe fittings are described here. More details can be found in Bernt Lie's Lecture notes, [1].

Due to different constrictions in the pipes, it is of interest to define losses in these fittings. This can be done based on friction pressure drop which can be calculated as:

$$\Delta p_{\mathrm{f}} = \frac{1}{2}\phi\rho v|v| \tag{3.14}$$

Here, the dimensionless factor ϕ is $\phi = f_{\mathrm{D}}\frac{L}{D}$ for a long, straight pipe. Here, ϕ will be the generalized friction factor. In this case, it is possible to write pressure drop for different constrictions. Some cases of various fittings are shown in Figures 3.5-3.8. Equations for the dimensionless factor ϕ are also demonstrated in these figures for the presented fittings.

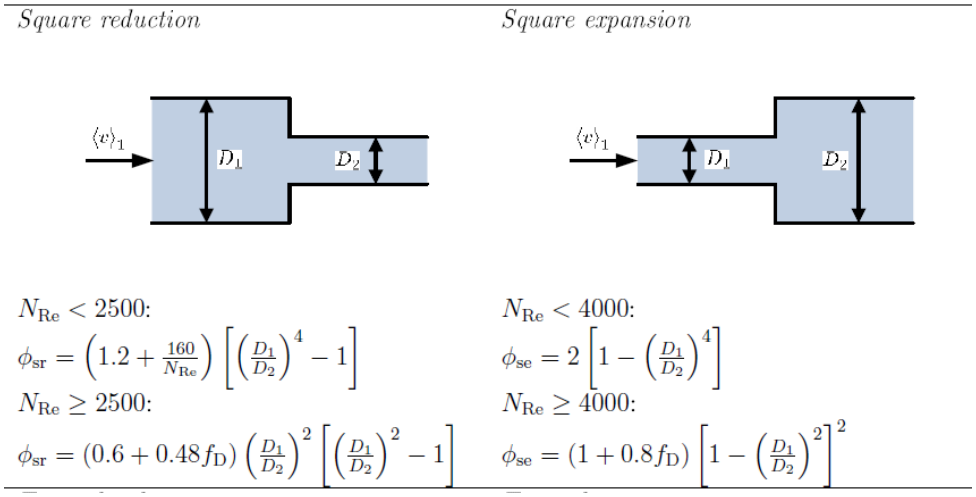

Figure 3.5: Square reduction/expansion fittings, [1].

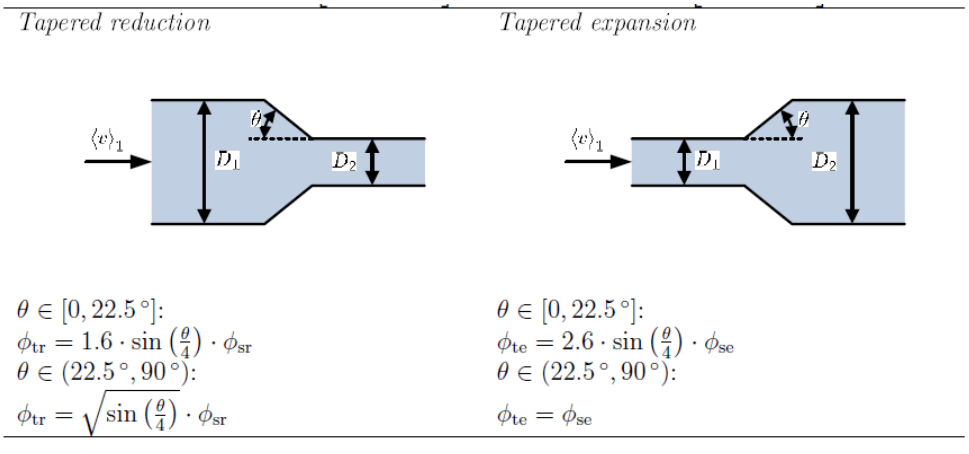

Figure 3.6: Tapered reduction/expansion fittings, [1].

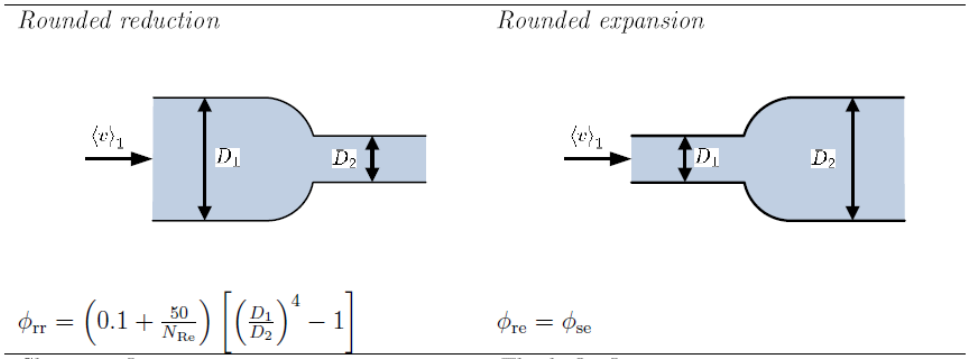

Figure 3.7: Rounded reduction/expansion fittings, [1].

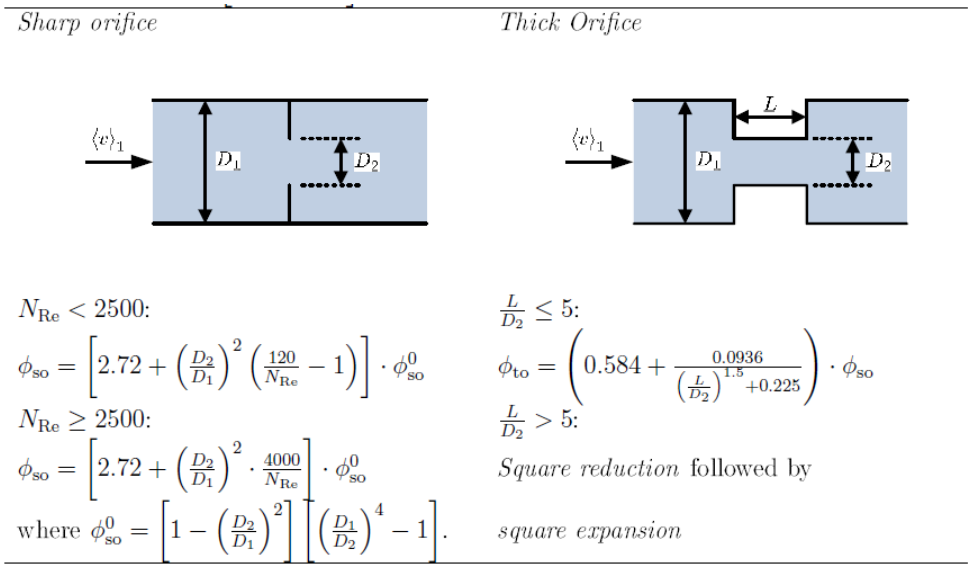

Figure 3.8: Sharp/Thick orifice fittings, [1].

Based on the presented equations and figures for the calculation of the dimensionless factor ϕ in the various fitting, a set of functions is encoded regarding each specific type of fittings, such as *SquareReduction*, *SquareExpansion*, *TaperedReduction*, *TaperedExpansion*, *RoundedReduction*, *SharpOrifice*, and *ThickOrifice*. All these functions receive the Reynolds' number N_{Re}, diameters of first and second pipes D_1 and D_2, and the pipe roughness height ε. Then, based on the equations from Figures 3.5-3.8, these functions provide value for the dimensionless factor ϕ. As an example, a Modelica code for defining the *SquareReduction* function looks as follows:

```
function SquareReduction
  input Modelica.SIunits.ReynoldsNumber N_Re "Reynold number"
    ;
  input Modelica.SIunits.Height eps "Pipe roughness height";
```

```
  input Modelica.SIunits.Diameter D_1, D_2; //Pipe diameters
  output Real phi;
protected
  Real f_D "friction factor";
algorithm
  f_D := Functions.DarcyFriction.fDarcy(N_Re, D_1, eps);
  if N_Re < 2500 then
    phi := (1.2 + 160 / N_Re) * ((D_1 / D_2) ^ 4 - 1);
  else
    phi := (0.6 + 0.48 * f_D) * (D_1 / D_2) ^ 2 * ((D_1 / D_2
       ) ^ 2 - 1);
  end if;
end SquareReduction;
```

Another function, *FittingPhi* also provides the dimensionless factor ϕ as an output. This function calls the presented above functions with a specific type of the fitting in order to get value for the factor ϕ. This function has the following inputs: the linear velocity v, the pipe length L, diameters of first and second pipes D_1 and D_2, liquid density and viscosity ρ and μ, the pipe roughness height ε. The last input for this function is a variable with the specific type *FittingType* that holds information about the fitting type.

3.3 Waterway

A typical structure of the waterway of the hydropower system is shown in Figure 3.9.

3.3.1 Reservoir

Figure 3.9 shows that the water level in the reservoir H_r is a key quantity, [5]. Similarly to the water tank, a reservoir model can be described by mass and momentum balances as following, [6]:

$$\begin{aligned} H_\mathrm{r}\tfrac{d\dot{m}_\mathrm{r}}{dt} &= \tfrac{\rho}{A_\mathrm{r}}\dot{V}_\mathrm{r}^2 + A_\mathrm{r}\left(p_\mathrm{atm} - p_\mathrm{r}\right) + \rho g H_\mathrm{r} A_\mathrm{r} - F_\mathrm{f,r} \\ \tfrac{dm_\mathrm{r}}{dt} &= \dot{m}_\mathrm{r} \end{aligned} \tag{3.15}$$

Here, $\dot{m}_\mathrm{r}$ is the reservoir mass flow rate that can be found from the reservoir volumetric flow rate $\dot{V}_\mathrm{r}$. A_r is a square area of the reservoir. p_atm and p_r are the atmospheric and the reservoir outlet pressures, respectively. $F_\mathrm{f,r}$ is a friction term that can be found using Darcy friction factor.

In a simple case, it can be assumed that the level of the reservoir is constant, the reservoir inlet flow equal the outlet flow, and the area of the reservoir is closed to infinity. Then

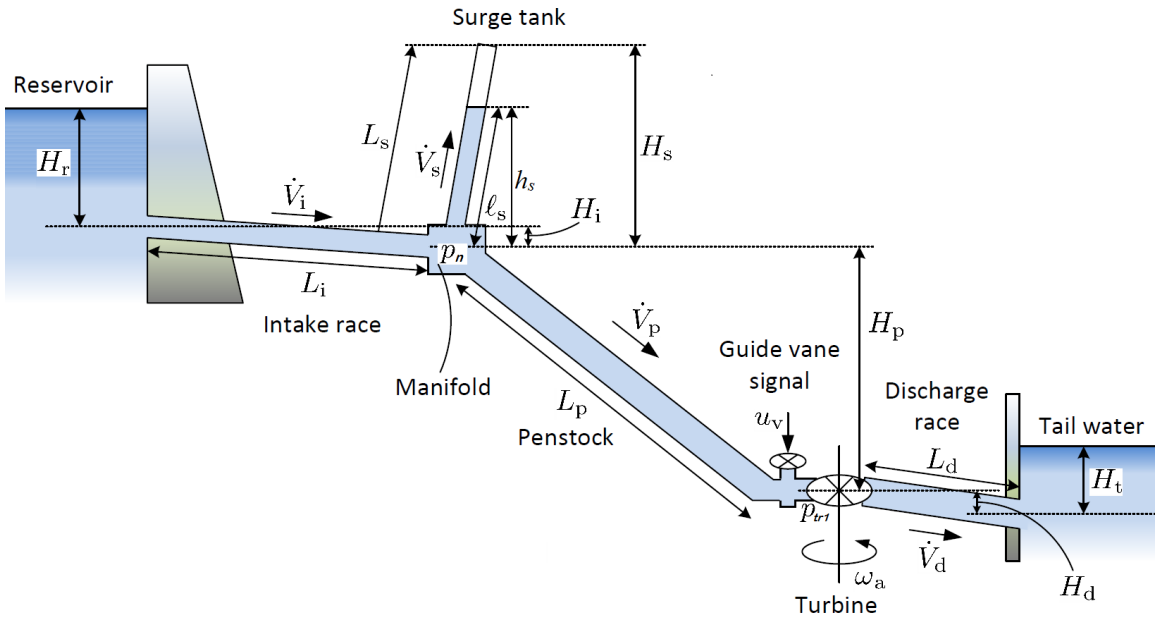

Figure 3.9: Reservoir structure, [1].

the reservoir can be presented just as an equation for pressure in the inlet/outlet of the reservoir, [5], [6].

$$p_r = p_{atm} + \rho g H_r \tag{3.16}$$

Hence, both of these cases are modelled in the *Reservoir* unit in the library. This unit uses the *Contact* connector and can be connected to other waterway units. The *Reservoir* unit can be specified with the following options:

- The user can choose a simple model of the reservoir, and calculate the outlet pressure depending on the depth of the outlet from the reservoir.
- The user can also choose a more complicated model, add the inflow to the reservoir and specify the reservoir geometry.
- Also, it is possible to connect an input signal with the varying water level in the reservoir.

3.3.2 Fitting

There are various possibilities of the fittings for the pipes with different diameters as well as the existence of orifices in the pipe. In this unit *Fitting*, the pressure drop due to these constrictions is defined using Eq. 3.14 and function *FittingPhi*. The *Fitting* unit uses the *ContactPort* connector model in order to have inlet and outlet connectors and

the possibility to define pressure drop between those connectors. Then, this unit can be connected to the other waterway units.

When the *Fitting* unit is in use, the user can postulate the specific type of fitting that is of interest and required based on the geometry parameters for this fitting.

3.3.3 Pipe

The simple model of the pipe unit *Pipe* gives possibilities for easy modelling of different conduit: intake race, penstock, tailrace, etc. In these waterway units, there are only small pressure variations due to the small slope angle (height difference between inlet and outlet of the component). That is why the model for these units can be simplified by considering incompressibility of the water and the inelasticity of the walls, [4]–[6]. A sketch of the pipe with all needed terms for modelling is shown in Figure 3.10.

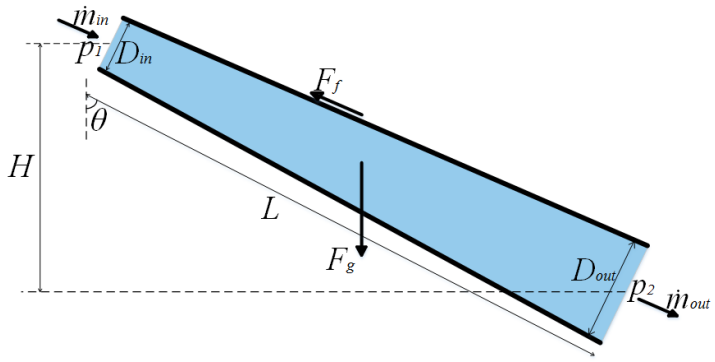

Figure 3.10: Model for flow through a pipe.

In the case of incompressible water, the mass in the filled pipe is constant, and:

$$\frac{dm_c}{dt} = \dot{m}_{c,in} - \dot{m}_{c,out} = 0 \tag{3.17}$$

Here, the mass of the water in the pipe (conduit) is $m_c = \rho V_c = \rho L_c \overline{A}_c$, where ρ is the water density, V_c – the volume of the water in the pipe, L_c – the length of the pipe (conduit) and $\overline{A}_c$ – the averaged cross-section area of the pipe that are defined from averaged pipe diameter $\overline{D}_c$. The inlet and outlet mass flow rates are equal with $\dot{m}_{c,in} = \rho \dot{V}_{c,in}$ and $\dot{m}_{c,out} = \rho \dot{V}_{c,out}$ respectively, where $\dot{V}_{c,in} = \dot{V}_{c,out}$ – the inlet and outlet volumetric flow rates in the pipe.

The momentum balance for this simplified model can be expressed as:

$$\frac{dM_c}{dt} = \dot{M}_{c,in} - \dot{M}_{c,out} + F_{p,c} + F_{g,c} + F_{f,c} \tag{3.18}$$

Here, the momentum of the water in the pipe is $M_c = m_c v_c$, where v_c is the average water velocity and can be defined as $v_c = \dot{V}_c / \overline{A}_c$. The inlet and outlet momentum flow rates are $\dot{M}_{c,in} = \dot{m}_{c,in} v_{c,in}$ and $\dot{M}_{c,out} = \dot{m}_{c,out} v_{c,out}$ respectively, where $v_{c,in} = \dot{V}_{c,in} / A_{c,in}$ and $v_{c,out} = \dot{V}_{c,out} / A_{c,out}$ are the velocities in the inlet and outlet of the pipe, respectively; and are equal in a case with constant diameter of the pipe ($A_{c,in} = A_{c,out}$). $F_{p,c}$ – the pressure force, due to the difference between the inlet and outlet pressures $p_{c,1}$ and $p_{c,2}$ can be calculated as follows: $F_{p,c} = A_{c,in} p_{c,1} - A_{c,out} p_{c,2}$. There is also gravity force that is defined as $F_{g,c} = m_c g \cos \theta_c$, where g – the gravitational acceleration and θ_c – the angle of the pipe slope that can be defined from the ratio of height difference H_c and the length L_c of the pipe. The last term in the momentum balance is friction force which can be calculated as $F_{f,c} = -\frac{1}{8} L_c f_{D,c} \pi \rho \overline{D}_c v_c |v_c|$ using the Darcy friction factor $f_{D,c}$ for the conduit.

The main defined variable is the volumetric flow rate. In this *Pipe* unit, the flow rate changes simultaneously in the whole pipe (information about the speed of wave propagation is not included here). Water pressures can be shown just in the boundaries of pipe (inlet and outlet pressure from connectors). This unit uses the *ContactPort* connector model and can be connected to other waterway units.

When the *Pipe* unit is in use, the user can specify the required geometry parameters for this pipe: length L_c, height difference H_c, inlet and outlet diameters $D_{c,1}$ and $D_{c,2}$, and pipe roughness height ε_c. In order to define the friction force $F_{f,c}$ the *Friction* function is used here. It should be noted that this unit provides possibilities for the modelling of pipes with both positive and negative slopes (positive or negative height difference). This unit can be initialized by the initial value of the flow rate $\dot{V}_{c,0}$. Otherwise, user can choose to an option when the simulation starts from steady-state and the OpenModelica handles automatically initial steady-state values (does not work properly in OpenModelica).

3.3.4 Surge Tank

The surge shaft/tank will be presented here as a vertical open pipe with constant diameter together with manifold, which connecting conduit, surge volume and penstock, [5], [6]. Surge volume (vertical open pipe) is shown in Figure 3.11.

The model for the surge volume can be described by mass and momentum balances as follows:

$$\begin{aligned} \frac{dm_s}{dt} &= \dot{m}_{s,in} = \rho \dot{V}_s \\ \frac{dm_s v_s}{dt} &= \dot{m}_{s,in} v_{s,in} + F_{p,s} + F_{g,s} + F_{f,s} \end{aligned} \tag{3.19}$$

Here, the mass of the water in the surge tank is $m_s = \rho V_s = \rho l_s A_s = \rho A_s \frac{h_s}{\cos \theta_s}$, where ρ is the water density, V_s is the volume of the water in the surge tank, h_s and l_s are the height and length of the surge tank filled with water and A_s is the cross-section area of the surge tank that defined from the vertical pipe diameter D_s. The water velocity v_s can be defined as $v_s = \dot{V}_s / A_s$. The inlet water velocity $v_{s,in} = \dot{V}_s / A_s$. $F_{p,s}$ is the pressure

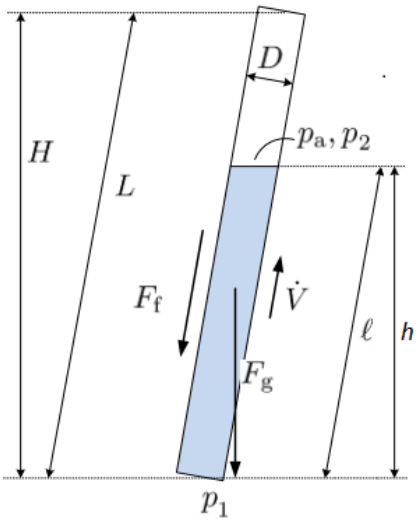

Figure 3.11: Model for a vertical open pipe.

force, due to the difference between the inlet and outlet pressures $p_{s,1}$ and p_{atm} and can be calculated as follows: $F_{p,s} = A_s\,(p_{s,1} - p_{atm})$. There is also gravity force that is defined as $F_{g,s} = m_s g \cos\theta_s$, where g – the gravitational acceleration and θ_s – the angle of the slope of the surge tank and can be defined from the ratio of height difference H_s and length L_s. The last term in the momentum balance is friction force, which can be calculated as $F_{f,s} = -\frac{1}{8} l_s f_{D,s} \pi \rho D_s v_s |v_s|$ using Darcy friction factor $f_{D,s}$ for the surge tank.

The manifold is described by the preservation of mass in steady-state; the volumetric flow rate in the intake race $\dot{V}_i$ equals to the sum of volumetric flow rates from surge volume $\dot{V}_s$ and penstock $\dot{V}_p$: $\dot{V}_i = \dot{V}_p + \dot{V}_s$. In addition, the manifold pressure is equal for all three connections. This manifold is already implemented in the *ContactNode* connectors model that is used in this *SurgeTank* unit. Then, this unit can be connected to other waterway units.

In the *SurgeTank* unit, the user can specify the required geometry parameters for the surge tank (vertical pipe): length L_s, height difference H_s, diameters D_s, pipe roughness height ε_s, and value for the atmospheric pressure p_{atm}. In order to define the friction force $F_{f,s}$ the *Friction* function is used here. This unit can be initialized by the initial values of the flow rate $\dot{V}_{s,0}$ and water height $h_{s,0}$. Otherwise, the user can decide on an option when the simulation starts from the steady-state and the OpenModelica automatically handles the initial steady-state values (does not work properly in OpenModelica).

3.3.5 Pipe with compressible water and elastic walls

Unlike the conduit, the penstock has considerable pressure variation due to a considerable height drop. Thus, to make the model for the penstock more realistic, the compressible water and the elastic walls of the penstock should be taken into account. To express the compressibility/elasticity, some compressibility coefficients which show the relationship between pressure, water density and pipe inner radius, are used, [4], [6].

The isothermal compressibility β_T is defined as follows:

$$\beta_T = \frac{1}{\rho}\frac{d\rho}{dp} \tag{3.20}$$

Here, ρ and p denote density and pressure, respectively. Assuming that the isothermal compressibility is independent of the pressure, this equation can be rewritten in a way that is convenient to calculate the fluid density at different pressures:

$$\rho = \rho^{\mathrm{atm}} e^{\beta_T (p - p^{\mathrm{atm}})} \tag{3.21}$$

Here p^{atm} is the atmospheric pressure and ρ^{atm} is the water density at atmospheric pressure. The relation between density and pressure from this equation is a fairly linear dependency for the pressure in the range which is normal in hydropower plants. That is why the previous equation can be simplified as follows:

$$\rho \approx \rho^{\mathrm{atm}} (1 + \beta_T (p - p^{\mathrm{atm}})) \tag{3.22}$$

In the same way, the relation between pressure and pipe cross-section area can be defined using equivalent compressibility coefficient β^{eq} due to the pipe shell elasticity; after simplification the relation looks as follows:

$$A \approx A^{\mathrm{atm}} (1 + \beta^{eq} (p - p^{\mathrm{atm}})) \tag{3.23}$$

Here, A^{atm} is the pipe cross-section area at atmospheric pressure.

It is also possible to define a linear relationship for the product of density and cross-sectional area that change with the pressure.

$$A \cdot \rho \approx A^{\mathrm{atm}} \rho^{\mathrm{atm}} (1 + \beta^{\mathrm{tot}} (p - p^{\mathrm{atm}})) \tag{3.24}$$

Here, β^{tot} is the total compressibility due to water compressibility and pipe shell elasticity ($\beta^{\mathrm{tot}} = \beta_T + \beta^{eq}$), and is related to the speed of sound in water inside the pipe.

Hence, using the previous equations for the relationship between the density of the water, cross-sectional area of the pipe, and pressure in the pipe, ODEs (3.17) and (3.18) for mass and momentum balances can be further developed into the PDEs, [4]:

$$\begin{gathered} A_{\mathrm{p}}^{\mathrm{atm}} \rho^{\mathrm{atm}} \beta^{\mathrm{tot}} \frac{\partial m_{\mathrm{p}}}{\partial t} = -\frac{\partial \dot{m}_{\mathrm{p}}}{\partial x} \\ \frac{\partial \dot{m}_{\mathrm{p}}}{\partial t} = -\frac{\partial}{\partial x}\left(\dot{m}_{\mathrm{p}} v_{\mathrm{p}} + A_{\mathrm{p}} p_{\mathrm{p}}\right) + \rho A_{\mathrm{p}} g \cos\theta - \tfrac{1}{8} f_{\mathrm{D,p}} \pi \rho D_{\mathrm{p}} v_{\mathrm{p}} |v_{\mathrm{p}}| \end{gathered} \tag{3.25}$$

The KP scheme is chosen for the discretization of the model for the elastic penstock with compressible water. Firstly, PDEs (3.25) for the elastic penstock model should be presented in vector form as a standard formulation for KP scheme, [2]:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = S \tag{3.26}$$

Here, $U = \begin{bmatrix} p_\mathrm{p} & \dot{m}_\mathrm{p} \end{bmatrix}^T$ is a vector of conserved variables, $F = \begin{bmatrix} \frac{\dot{m}_\mathrm{p}}{A_\mathrm{p}^\mathrm{atm}\rho^\mathrm{atm}\beta^\mathrm{tot}} & \dot{m}_\mathrm{p} v_\mathrm{p} + A_\mathrm{p} p_\mathrm{p} \end{bmatrix}^T$ is a vector of fluxes, and $S = \begin{bmatrix} 0 & \rho A_\mathrm{p} g \cos\theta_\mathrm{p} - \frac{1}{8} f_\mathrm{D,p} \pi \rho D_\mathrm{p} v_\mathrm{p} |v_\mathrm{p}| \end{bmatrix}^T$ is a source terms vector.

As shown above in the description of the KP scheme, the eigenvalues $\lambda_{1,2}$ of the Jacobian $\frac{\partial F}{\partial U}$ of the system are needed and can be found as follows, [4]:

$$\lambda_{1,2} = \frac{v_\mathrm{p} \pm \sqrt{v_\mathrm{p}^2 + \frac{4A_\mathrm{p}}{A_\mathrm{p}^\mathrm{atm}\rho^\mathrm{atm}\beta^\mathrm{tot}}}}{2} \tag{3.27}$$

From these eigenvalues, it can be deduced that the speed of sound is given as $c = \sqrt{\frac{A_\mathrm{p}}{A_\mathrm{p}^\mathrm{atm}\rho^\mathrm{atm}\beta^\mathrm{tot}}}$, thus confirming that the total compressibility factor β^tot is related to the speed of sound.

Hence, the function for the KP scheme *KPmethod* from function class *KP07* is then used in unit *PenstockKP* in order to discretize the presented PDEs into ODEs. The *KPmethod* function provides the right hand side of Eq. 3.7 (discretization solution of PDE) that is then used for ODE in the *PenstockKP*. Moreover, the values of states at the cell interfaces $U^\pm_{j\pm\frac{1}{2}}$ are taken from function *KPmethod* in the *PenstockKP* unit in order to define the vectors of eigenvalues $\lambda^\pm_{1,j\pm\frac{1}{2}}$ and $\lambda^\pm_{2,j\pm\frac{1}{2}}$, and the vector of fluxes $F\left(U^\pm_{j\pm\frac{1}{2}}\right)$. Then, these vectors together with the state vector with the cell centre average values $\bar{U}_{j=1..N}$, and source terms vector $\bar{S}_j$ are used in the function *KPmethod*. The boundaries conditions are also specified for the *KPmethod* function in the *PenstockKP* unit and are the values for the inlet and outlet pressures $p_\mathrm{p,1}$ and $p_\mathrm{p,2}$.

The *PenstockKP* unit uses the *TwoContact* connector model that provides information about inlet and outlet pressure and the mass flow rate of two connectors which can be connected to other waterway units. In this *PenstockKP* unit, the user can specify the required geometry parameters for the: length L_p, height difference H_p, inlet and outlet diameters $D_\mathrm{p,1}$ and $D_\mathrm{p,2}$, pipe roughness height ε_p and the number of cells N for the discretization. In order to define the friction force $F_\mathrm{f,p}$ in the cell of the pipe, the *Friction* function is used here. This unit can be initialized by the initial value of the flow rate $\dot{V}_\mathrm{p,0}$ and pressure $p_\mathrm{p,0}$ for each cell of the pipe. In order to simplify the pressure initialization,

the user can simply specify the initial value for the surge tank water height $h_{s,0}$ (then an encoded formula for the pressure initialization is used). Otherwise, the user can choose an option when the simulation starts from steady-state and the OpenModelica automatically handles the initial steady-state values (does not work properly in OpenModelica).

3.3.6 Open Channel

Similarly to the detailed model of the pipe, the model of the open channel is also encoded in the library. The open channel model looks as follows, [2], [4]:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = S \tag{3.28}$$

where:

$$U = \begin{bmatrix} q & z \end{bmatrix}^T ,$$

$$F = \begin{bmatrix} q & \frac{q^2}{z-B} + \frac{g}{2}(z-B)^2 \end{bmatrix}^T ,$$

$$S = \begin{bmatrix} 0 & -g(z-B)\frac{\partial B}{\partial x} - \frac{g f_n^2 q|q|(w+2(z-B))^{\frac{4}{3}}}{w^{\frac{4}{3}}} \frac{1}{(z-B)^{\frac{7}{3}}} \end{bmatrix}^T ,$$

with: $z = h + B$, and $q = \frac{\dot{V}}{w}$. Here, h is water depth in the channel, B is the channel bed elevation, q is the discharge per unit width w of the open channel. f_n is the Manning's roughness coefficient. The KP scheme is described earlier, but some additional specific details for open channels should be added here. Firstly, the eigenvalues for this model are defined as follows, [2]:

$$\lambda_{1,2} = u \pm \sqrt{gh} \tag{3.29}$$

where, u is the cross-section average water velocity. In the channel areas which are dry or almost dry (if the computational domain contains a dry bed, islands or coastal areas), the values of $h^{\pm}_{i\pm\frac{1}{2}}$ could be very small or even zero. In such cases when $h^{\pm}_{i\pm\frac{1}{2}} < \varepsilon$, with ε being an a-priori chosen small positive number (e.g. $\varepsilon = 1e^{-5}$), the velocity at the cell centres in the entire domain is recomputed by the ted by the desingularization formula, [2]:

$$\bar{u}_j = \frac{2\bar{h}_j \bar{q}_j}{\bar{h}_j^2 + \max\left(\bar{h}_j^2, \varepsilon^2\right)} \tag{3.30}$$

Then, the point values of the velocity $u^{\pm}_{i\pm\frac{1}{2}}$ at the left/right cell interfaces, i.e., at $x_j = x_{j\pm\frac{1}{2}}$ are computed as, [2]

$$\begin{aligned} u^{-}_{j+\frac{1}{2}} &= \bar{u}_j + \tfrac{\Delta x}{2} s_{u_j} \\ u^{+}_{j+\frac{1}{2}} &= \bar{u}_{j+1} - \tfrac{\Delta x}{2} s_{u_{j+1}} \\ u^{-}_{j-\frac{1}{2}} &= \bar{u}_{j-1} + \tfrac{\Delta x}{2} s_{u_{j-1}} \\ u^{+}_{j-\frac{1}{2}} &= \bar{u}_j - \tfrac{\Delta x}{2} s_{u_j} \end{aligned} \tag{3.31}$$

The slope or the numerical derivative of the velocity s_{u_j} are calculated using the same limiter function as in equation 3.10, however, in this case replacing U by u (it has not been rewritten here for the sake of brevity), [2].

Hence, similar to the *PenstockKP* unit the function for the KP scheme *KPmethod* from function class *KP07* is then used in unit *OpenCannel* in order to discretize the presented PDEs into ODEs. The values of states at the cell interfaces $U^{\pm}_{j\pm\frac{1}{2}}$ are taken from function *KPmethod* in the *OpenCannel* unit in order to define the vectors of eigenvalues $\lambda^{\pm}_{1,j\pm\frac{1}{2}}$ and $\lambda^{\pm}_{2,j\pm\frac{1}{2}}$, the point values of the velocity $u^{\pm}_{i\pm\frac{1}{2}}$, and the vector of fluxes $F\left(U^{\pm}_{j\pm\frac{1}{2}}\right)$. Then, these vectors together with the state vector with the cell centre average values $\bar{U}_{j=1..N}$ and source terms vector $\bar{S}_j$ are used in the function *KPmethod*. The boundaries conditions are also specified for the *KPmethod* function in the *OpenCannel* unit and are the values for the inlet and outlet flows per unit width q_1 and q_2.

The *OpenCannel* unit uses the *TwoContact* connector model that gives information about inlet and outlet pressure (water depth in the channel) and the flow rate of two connectors which can be connected to other waterway units. In this *OpenCannel* unit, the user can specify the required geometry parameters for the: length L and width w of the channel, height vector H of the channel bed with a height from the left and right sides, the Manning's roughness coefficient f_n, and the number of cells N for the discretization. This unit can be initialized by the initial value of the flow rate $\dot{V}_0$ and water depth h_0 for each cell of the channel. User can also change the boundary condition for the KP scheme.

3.3.7 Reservoir Channel

In order to make a more detailed model of the reservoir, the open channel model is used, where the channel bed is assumed to be flat (no slope). Here, the user also specifies the geometry parameters of the channel (reservoir) such as length L and width w of the channel (reservoir), height vector H of the reservoir bed with height from the left and right sides (should be same number in order to have flatbed), and the number of cells N for the discretization. This unit can be initialized by the initial value of the water depth h_0 in the reservoir.

The *ReservoirChannel* unit uses the *Contact* connector that provides information about the outlet pressure and the flow rate from/to the reservoir which can be connected to other waterway units.

3.3.8 Runoff

Similar to many other hydrological models, the HBV model is based on the land phase of the hydrological (water) cycle, see Figure 3.12. The figure shows that the HBV model consists of four main water storage components connected in a cascade form. Using a variety of weather information, such as air temperature, precipitation and potential evapotranspiration, the dynamics and the balances of the water in the presented water storages are calculated. Hence, the runoff/inflow from some of the defined catchment areas can be found, [7].

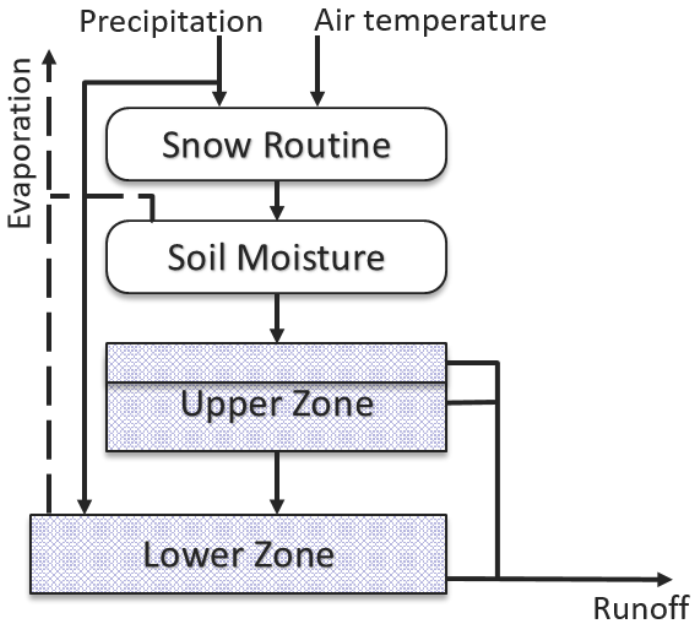

Figure 3.12: Structure of the HBV model.

The model is developed for each water storage component to define the dynamics and balances of the water. In addition, the catchment area is divided into elevation zones (usually not more than ten) where each zone has the same area. The air temperature and the precipitation are provided for each elevation zone. Hence, all calculations within each water storage component are performed for each elevation zone.

Snow routine

In the snow routine segment, the snow storage, as well as snowmelt are computed. This computation is performed for each elevation zone. Using the mass balance, the change in the dry snow storage volume $V_{\mathrm{s,d}}$, is found as follows:

$$\frac{dV_{\mathrm{s,d}}}{dt} = \dot{V}_{\mathrm{p,s}} - \dot{V}_{\mathrm{d2w}} \tag{3.32}$$

Here, the flow of the precipitation in the form of snow is denoted as $\dot{V}_{\mathrm{p,s}}$. This precipitation in the form of snow is defined from the input precipitation flow, $\dot{V}_{\mathrm{p}}$, based on the information about the air temperature, T, a threshold temperature for snowmelt, T_{T}, and for the area that is not covered by lakes (the fractional area covered by the lakes, a_{L}, is used):

$$\dot{V}_{\mathrm{p,s}} = \begin{cases} \dot{V}_{\mathrm{p}} K_{\mathrm{CR}} K_{\mathrm{CS}} (1 - a_{\mathrm{L}}), & \text{if } T \leq T_{\mathrm{T}} \\ 0, & \text{if } T > T_{\mathrm{T}} \end{cases} \tag{3.33}$$

Precipitation correction coefficients K_{CR} and K_{CS} are also used here, for the rainfall and snowfall precipitations, respectively. Then, the flow of precipitation in the form of rain is defined as follows:

$$\dot{V}_{\mathrm{p,r}} = \begin{cases} \dot{V}_{\mathrm{p}} K_{\mathrm{CR}} (1 - a_{\mathrm{L}}), & \text{if } T > T_{\mathrm{T}} \\ 0, & \text{if } T \leq T_{\mathrm{T}} \end{cases} \tag{3.34}$$

The flow of the melting snow (melting of snow from dry form to water form), $\dot{V}_{\mathrm{d2w}}$, can be found using the following expression based on the degree-day factor K_{dd} and the area of the elevation zone A_{e}:

$$\dot{V}_{\mathrm{d2w}} = \begin{cases} A_{\mathrm{e}} K_{\mathrm{dd}} (T - T_{\mathrm{T}}) (1 - a_{\mathrm{L}}), & \text{if } T > T_{\mathrm{T}} \text{ and } V_{\mathrm{s,d}} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.35}$$

Finally, the flow out of the snow routine to the next soil moisture segment, $\dot{V}_{\mathrm{s2s}}$, is found as a sum of flows of precipitation in the form of rain, and the melted snow:

$$\dot{V}_{\mathrm{s2s}} = \dot{V}_{\mathrm{p,r}} + \dot{V}_{\mathrm{d2w}} \tag{3.36}$$

It should be noted that a simplification related to the threshold temperature, T_{T}, is assumed here. This threshold temperature describes both the snow melt and the rainfall to snowfall transition temperatures in the presented model. In reality, this threshold temperature might differ for each of these processes. In addition, the storage of snow in water form is not considered here, mostly due to the simplification with the threshold temperature.

Soil moisture routine

In the soil moisture segment, the water storage in the ground (soil) is found together with actual evapotranspiration from the snow-free areas. The net runoff to the next segment (upper zone) is also defined here. Using the mass balance, the volume of the soil moisture storage, $V_{\mathrm{s,m}}$, is found as follows:

$$\frac{dV_{\mathrm{s,m}}}{dt} = \dot{V}_{\mathrm{s2s}} - \dot{V}_{\mathrm{s2u}} - \alpha_{\mathrm{e}}\dot{V}_{\mathrm{s,e}} \tag{3.37}$$

Here, $\dot{V}_{\mathrm{s2u}}$ is the net runoff to the next segment (the upper zone). $\dot{V}_{\mathrm{s,e}}$ is the actual evapotranspiration from the soil, that is taken into account only for the snow-free areas (zones). To define these snow-free zones, coefficient α_{e} is used and equals one for snow-free areas and zero for covered-by-snow areas. The actual evapotranspiration can be found from the potential evapotranspiration, $\dot{V}_{\mathrm{e}}$, the volume of the soil moisture storage, $V_{\mathrm{s,m}}$, the area of the elevation zone A_{e}, and the field capacity — threshold soil (ground) moisture storage, g_{T}:

$$\dot{V}_{\mathrm{s,e}} = \begin{cases} \frac{V_{\mathrm{s,m}}}{A_{\mathrm{e}} g_{\mathrm{T}}}\dot{V}_{\mathrm{e}}, & \text{if } V_{\mathrm{s,m}} < A_{\mathrm{e}} g_{\mathrm{T}} \\ \dot{V}_{\mathrm{e}}, & \text{if } V_{\mathrm{s,m}} \geq A_{\mathrm{e}} g_{\mathrm{T}} \end{cases} \tag{3.38}$$

The potential evapotranspiration, $\dot{V}_{\mathrm{e}}$, is defined as the input to the hydrology model, similarly to the air temperature and precipitations.

The output of the soil moisture segment — the net runoff to the next segment, $\dot{V}_{\mathrm{s2u}}$, can be found based on the field capacity, g_{T}, as follows:

$$\dot{V}_{\mathrm{s2u}} = \begin{cases} \left(\frac{V_{\mathrm{s,m}}}{A_{\mathrm{e}} g_{\mathrm{T}}}\right)^{\beta}\dot{V}_{\mathrm{s2s}}, & \text{if } 0 \leq V_{\mathrm{s,m}} < A_{\mathrm{e}} g_{\mathrm{T}} \\ \dot{V}_{\mathrm{s2s}}, & \text{if } V_{\mathrm{s,m}} \geq A_{\mathrm{e}} g_{\mathrm{T}} \end{cases} \tag{3.39}$$

Here, β is an empirical parameter for specifying the relationship between the flow out of the snow routine, the soil moisture storage, and the net runoff from the soil moisture. Typically, $\beta \in [2,3]$, which leads to nonlinearity in Eq. 3.39.

Runoff routine

The upper and lower zones from Figure 3.12 are combined into one segment — the runoff routine. In this segment, the runoff from the catchment area is found based on the outflow from the soil moisture. The effects of the precipitation to, and evapotranspiration from the lakes in the catchment area are also taken into account here.

The upper zone characterises components with quick runoff. The following mass balance is used for the upper zone description:

$$\frac{dV_{\mathrm{u,w}}}{dt} = \dot{V}_{\mathrm{s2u}} - \dot{V}_{\mathrm{u2l}} - \dot{V}_{\mathrm{u2s}} - \dot{V}_{\mathrm{u2q}} \tag{3.40}$$

Here, $V_{\mathrm{u,w}}$ is the water volume in the upper zone that depends on the saturation threshold, s_{T}, which defines the surface (fast) runoff, $\dot{V}_{\mathrm{u2s}}$, and the fast runoff, $\dot{V}_{\mathrm{u2q}}$. $\dot{V}_{\mathrm{u2b}}$ is the runoff to the lower zone and is defined by the percolation capacity, K_{PC}, for the area that is not covered by lakes:

$$\dot{V}_{\mathrm{u2l}} = A_{\mathrm{e}}(1 - a_{\mathrm{L}})K_{\mathrm{PC}} \tag{3.41}$$

The surface runoff, $\dot{V}_{\mathrm{u2s}}$, can be found using the saturation threshold, s_{T}, and the water volume in the upper zone, $V_{\mathrm{u,w}}$:

$$\dot{V}_{\mathrm{u2s}} = \begin{cases} a_1(V_{\mathrm{u,w}} - A_{\mathrm{e}}s_{\mathrm{T}}), & \text{if } V_{\mathrm{u,w}} > A_{\mathrm{e}}s_{\mathrm{T}} \\ 0, & \text{if } V_{\mathrm{u,w}} \leq A_{\mathrm{e}}s_{\mathrm{T}} \end{cases} \tag{3.42}$$

Here, a_1 is a parameter that represents the recession constant for the surface runoff. A similar recession constant, a_2, is used for the fast runoff, $\dot{V}_{\mathrm{u2q}}$, calculations:

$$\dot{V}_{\mathrm{u2q}} = a_2 \min(V_{\mathrm{u,w}}, A_{\mathrm{e}}s_{\mathrm{T}}) \tag{3.43}$$

The lower zone characterises the lake and the groundwater storages and defines the base runoff from the catchment area. The following mass balance equation is used for the lower zone description:

$$\frac{dV_{\mathrm{l,w}}}{dt} = \dot{V}_{\mathrm{u2l}} + a_{\mathrm{L}}\dot{V}_{\mathrm{p}} - \dot{V}_{\mathrm{l2b}} - a_{\mathrm{L}}\dot{V}_{\mathrm{e}} \tag{3.44}$$

The water volume in the lower zone is denoted as $V_{\mathrm{l,w}}$. As mentioned previously, $\dot{V}_{\mathrm{p}}$ and $\dot{V}_{\mathrm{e}}$ are the precipitation and the potential evapotranspiration flows, respectively. a_{L} is the fractional area covered by lakes. $\dot{V}_{\mathrm{l2b}}$ is the base runoff from the lower zone that can be found as follows:

$$\dot{V}_{\mathrm{l2b}} = a_3 V_{\mathrm{l,w}} \tag{3.45}$$

Here, a_3 is the recession constant similar to a_1 and a_2.

The total runoff from the catchment, $\dot{V}_{\mathrm{tot}}$, is a sum of the base, quick, surface runoffs for each elevation zones, and is defined as follows:

$$\dot{V}_{\mathrm{tot}} = \sum_{i=1}^{n} (\dot{V}_{\mathrm{l2b},i} + \dot{V}_{\mathrm{u2s},i} + \dot{V}_{\mathrm{u2q},i}) \tag{3.46}$$

Here, the base $\dot{V}_{\mathrm{l2b},i}$, quick $\dot{V}_{\mathrm{u2q},i}$, and surface $\dot{V}_{\mathrm{u2s},i}$ runoffs are first summed up for each of the n elevation zones and then these sums of the base, quick and surface runoffs are added together.

Hence, this hydrology model is encoded in the *OpenHPL* library as the *RunOff_zones* unit where the main defined variable is the total runoff from the catchment. This unit uses the standard Modelica connector *RealOutput* connector as an output from the model that can be connected to, for example, simple reservoir model *Reservoir* unit.

In order to get historic information about the air temperature, precipitation, and potential evapotranspiration for each of the elevation zones, the standard Modelica *CombiTimeTable* source models are used in order to read this data from the text files.

When the *RunOff_zones* unit is in use, the user can specify the required geometry parameters for the catchment: the number of elevation zones, all hydrology parameters such as threshold temperatures, degree-day factor, precipitation correction coefficients, field capacity and β parameter in soil moisture routine, threshold level for quick runoff in upper zone, percolation from upper zone to lower zone, recession constants for the surface and quick runoffs in upper zone, and recession constant for the base runoff in lower zone. Finally, the user can also specify the info about the text files where the data for the *CombiTimeTable* models are stored.

3.4 Electro-Mechanical

3.4.1 Turbine

The turbine unit can be expressed with a simple turbine model based on a look-up table (turbine efficiency vs. guide vane opening). This simple turbine model is described by Eq. 3.47, [1], [8], where the mechanical turbine shaft power $\dot{W}_{\mathrm{tr}}$ is defined as:

$$\dot{W}_{\mathrm{tr}} = \eta_{\mathrm{h}} \Delta p_{\mathrm{tr}} \dot{V}_{\mathrm{tr}} \tag{3.47}$$

Here, η_{h} gives the turbine hydraulic efficiency that is found from a standard turbine look-up table and depends on the turbine control signal, u_v. Δp_{tr} is the pressure drop through the turbine that is defined as the difference between inlet and outlet turbine pressures, i.e., $\Delta p_{\mathrm{tr}} = p_{\mathrm{tr1}} - p_{\mathrm{tr2}}$. The relationship between the turbine volumetric flow rate $\dot{V}_{\mathrm{tr}}$ and the pressure drop Δp_{tr} is described through a simple valve-like expression as follows:

$$\dot{V}_{\mathrm{tr}} = C_{\mathrm{v}} u_{\mathrm{v}} \sqrt{\frac{\Delta p_{\mathrm{tr}}}{p^{\mathrm{a}}}} \tag{3.48}$$

Here, C_{v} in Eq. 3.48 is some guide vane "valve capacity" that can be tuned by using the nominal turbine net head (nominal pressure drop) and the nominal turbine flow rate. p^{a} is the atmospheric pressure.

Based on Eqs. 3.47 and 3.48, the simple turbine model is implemented in *OpenHPL* as the *Turbine* element. In this *Turbine* unit, the multi-physic connections are used in order to stay connected to waterway units as well as to the other electro-mechanical units. Those connections are already implemented in the *TurbineContacts* connectors model that is

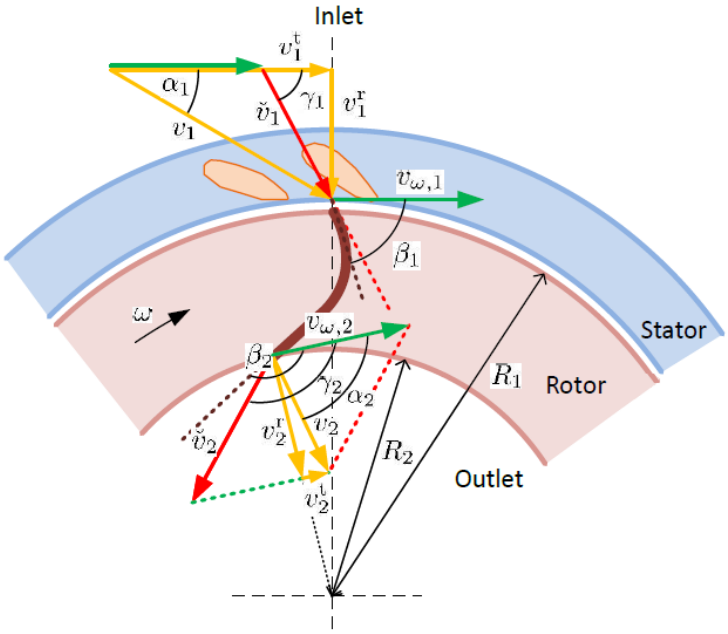

Figure 3.13: Key quantities in the Francis turbine model, with blade angles β_1 and β_2. The water effluent comes out from the paper plane, [1].

used in this *Turbine* unit. Then, this unit can be connected to other waterway and electro-mechanical units.

In the *Turbine* unit, the user can specify the required parameters for the simple turbine model: guide vane "valve capacity" C_v, the nominal turbine net head (nominal pressure drop) and the nominal turbine flow rate, turbine guide vane nominal opening signal $u_{v,n}$ in per unit value from 0 to 1. The user can also choose either to use the constant turbine efficiency and specify it, or to use the look-up table for the turbine efficiency and also specify this table.

3.4.2 Francis

Our library also includes a mechanistic Francis turbine model based on the Euler turbine equations. The key quantities of the model are shown in Fig. 3.13, and the shaft power $\dot{W}_s$ produced in the Francis turbine is defined as follows, [1], [9]:

$$\dot{W}_s = \dot{m}\omega\left(R_1\frac{\dot{V}}{A_1}\cot\alpha_1 - R_2\left(\omega R_2 + \frac{\dot{V}}{A_2}\cot\beta_2\right)\right). \tag{3.49}$$

Here, $\dot{m}$ and $\dot{V}$ are the mass and volumetric flow rate through the turbine, respectively, and ω is the angular velocity of the runner. R_1 and R_2 are the inlet and outlet radius of the

runner, respectively. A_1 and A_2 are the inlet and outlet cross-sectional areas, respectively, and can be defined by using the runner dimensions: R_1, R_2, and w_1 which is the inlet width/height of the runner/blades. α_1 is the inlet guide vane angle that is given by a control signal. β_2 is the outlet blade angle.

The total work rate $\dot{W}_t$ removed through the turbine is:

$$\dot{W}_t = \dot{W}_s + \dot{W}_{ft} + \Delta p_v \dot{V}. \tag{3.50}$$

Here, Δp_v is the pressure loss across the guide vane due to friction and is often neglected. The total work rate might also be formulated based on Bernoulli's law: $\dot{W}_t = \Delta p_{tr}\dot{V} + \frac{1}{2}\dot{m}\dot{V}^2(\frac{1}{A_0^2} - \frac{1}{A_2^2})$, from where the total pressure loss across the turbine Δp_{tr} can be defined; A_0 is the inlet cross section area to the spiral case. $\dot{W}_{ft}$ – the friction term that represents various friction losses within the turbine is calculated as follows:

$$\begin{aligned} \dot{W}_{ft} &= k_{ft,1}\dot{V}(\cot\gamma_1 - \cot\beta_1)^2 \\ &+ k_{ft,2}\dot{V}\cot^2\alpha_2 + k_{ft,3}\dot{V}^2. \end{aligned} \tag{3.51}$$

Here, $k_{ft,1}$, $k_{ft,2}$ and $k_{ft,3}$ are friction coefficients that represent shock, whirl, and pipe friction losses, respectively. These coefficients are tuning parameters for the mechanistic Francis turbine model. β_1 is the inlet blade angle which in the nominal operating condition should be equal to the angle of the relative velocity γ_1 in order to achieve an influent no-shock condition (the angle of the relative velocity is defined from: $\cot\gamma_1 = \cot\alpha_1 - \frac{\omega R_1}{\dot{V}}A_1$). To satisfy the no-whirl effluent condition, angle α_2 should be equal to 0. This angle is defined as $\cot\alpha_2 = \cot\beta_2 + \frac{\omega R_2}{\dot{V}/A_2}$.

We propose the following expressions for the turbine loss coefficients, [8]:

$$\begin{aligned} k_{ft,1} &= 11.6 \cdot 10^3 e^{8.9\cdot 10^{-3} H_\mathrm{n}} \\ k_{ft,2} &= 0 \\ k_{ft,3} &= 720 e^{6.7\cdot 10^{-3} H_\mathrm{n}} \end{aligned} \tag{3.52}$$

The efficiency of the turbine can be defined as follows:

$$\eta = \frac{\dot{W}_s}{\dot{W}_t} \tag{3.53}$$

Turbine design algorithm. Geometry parameters for the Francis turbine must be found in order to use the mechanistic turbine model as presented above. These parameters, such as blade angles or runner dimensions, can be found from design data. Typically, for real (in use) turbines, these data are unavailable due to trade confidentiality. Thus, it is of interest to develop a design algorithm that can be used to define all the geometry

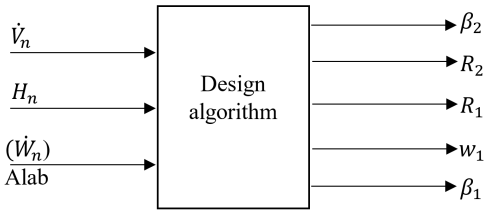

Figure 3.14: Block diagram that describes the turbine design algorithm (inputs and outputs).

parameters. The structure of this algorithm is shown in Fig. 3.14, where the input and output values for the design algorithm are presented.

As input data for the calculation, nominal net head H_n and volumetric flow rate $\dot{V}_n$ are used. A possible turbine design algorithm is as follows, ref. Fig. 3.13, [10]:

1. Choose the outlet blade angle β_2 and reference velocity $v_{\omega,2}$. These values are usually in the interval:
$$\begin{aligned} 158^\circ \leq \beta_2 \leq 165^\circ \\ 35\text{m/s} \leq v_{\omega,2} \leq 42\text{m/s} \end{aligned} \tag{3.54}$$
Here, the outlet angle and reference velocity take higher values for higher heads. Brekke suggests that these values may be chosen as $\beta_2 = 162.5^\circ$ and $v_{\omega,2} = 41\text{m/s}$.

2. Define the outlet runner cross-section area A_2 (radius R_2) and adjust it together with reference velocity $v_{\omega,2}$ to the normal synchronous rotational speed.
First, the meridional velocity is defined as:
$$v_2^r = -\frac{v_{\omega,2}}{\cot\beta_2}, \tag{3.55}$$
then outlet radius can be defined from the outlet cross-sectional area ($A_2 = \pi R_2^2$):
$$v_2^r = \frac{\dot{V}}{A_2} \Rightarrow R_2 = \sqrt{\frac{\dot{V}}{\pi v_2^r}} \tag{3.56}$$
Then, the turbine rotational speed n [RPM] can be calculated from the angular velocity ($\omega = \frac{\pi n}{30}$):
$$v_{\omega,2} = \omega R_2 \Rightarrow n = \frac{30 v_{\omega,2}}{\pi R_2} \tag{3.57}$$
After this the turbine speed should be reduced to the nearest synchronous speed (depends on number of pole pairs p in the generator: $n = \frac{60f}{p}$, where frequency f is constant 50 Hz) and then the outlet radius with the reference velocity should be recalculated in reverse order, using (3.57), (3.56) and (3.55).

Normally, the information about the turbine rotational speed is available, so the outlet runner radius and the reference velocity can be found directly from (3.55), (3.56) and (3.57).

3. Choosing the inlet runner dimension, inlet cross-section area A_1 (radius R_1 and width w_1).
The inlet radius can be defined from the reference velocity $v_{\omega,1}$ as follows:

$$R_1 = \frac{v_{\omega,1}}{\omega} = \frac{30 v_{\omega,1}}{\pi n} \tag{3.58}$$

Here, the reference velocity can be chosen from the range of reduced value $\overline{v}_{\omega,1} \in [0.7, 0.75]$, which is dimensionless and expressed as:

$$\overline{v}_{\omega,1} = \frac{v_{\omega,1}}{\sqrt{2gH}} \tag{3.59}$$

It is common to use $\overline{v}_{\omega,1} = 0.725$.
Regularly, in order to avoid backflow in the runner, an acceleration of the flow through the runner is desirable. That is why the outlet meridional velocity can be chosen approximately ten per cent higher than the inlet.

$$v_2^r = 1.1 v_1^r \tag{3.60}$$

Then the inlet runner width w_1 can be calculated from the inlet cross-sectional area ($A_1 = 2\pi R_1 w_1$):

$$v_1^r = \frac{\dot{V}}{A_1} \Rightarrow w_1 = \frac{\dot{V}}{2\pi R_1 v_1^r} \tag{3.61}$$

Here, it should be noted that the blade thickness could be included for improving the calculation of the inlet cross-section area, e.g., 10% of the perimeter.

4. The inlet blade angle β_1 can be found as follows:

$$\tan(180^\circ - \beta_1) = \frac{v_1^r}{v_{\omega,1} - v_1^t} \tag{3.62}$$

Here, v_1^t is the tangential velocity and can be defined from dimensionless value $\overline{v}_1^t = 0.48/\overline{v}_{\omega,1}$, using (3.59) to convert from dimensionless value.

Guide vane actuation. In addition, a model for the guide vane opening is also included in order to define the inlet guide vane angle α_1, [1]. The guide vane geometry is depicted in Figure 3.15.

From Figure 3.15 (a), assuming that the actuator cylinder is "vertical" in position "0", it can be found that

$$\begin{gathered} R_Y^2 = r_Y^2 + Y_0^2 \\ \cos\theta_0 = \frac{r_Y}{R_Y} \end{gathered} \tag{3.63}$$

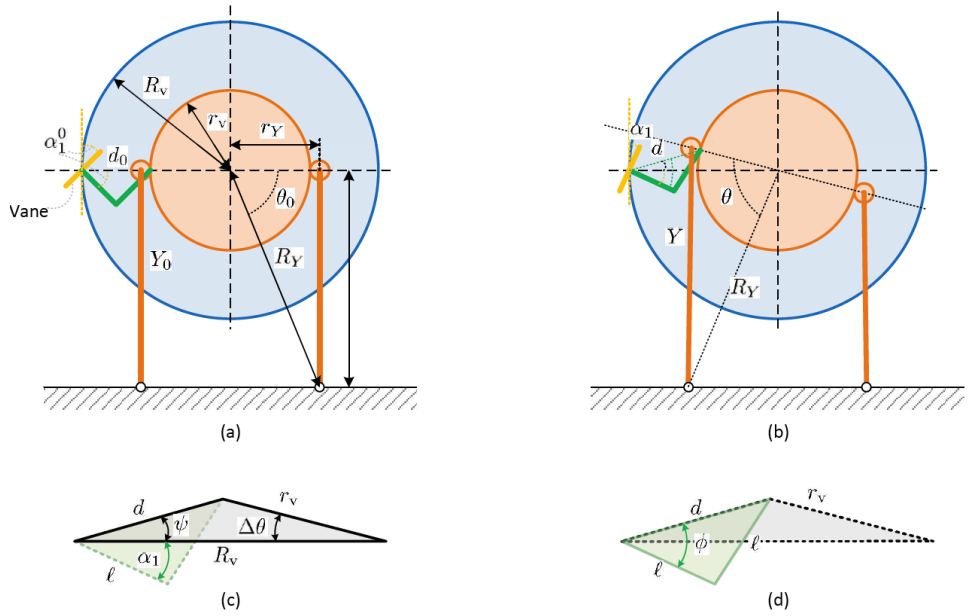

Figure 3.15: Guide vane geometry relating actuator position Y to guide vane angel α_1, [1].

Clearly, $d_0 = R_v - r_v$. Next, moving the actuator to position Y, Figure 3.15 (b) with the cosine law gives

$$Y^2 = r_Y^2 + R_Y^2 - 2r_Y R_Y \cos\theta \tag{3.64}$$

thus specifying angle θ. The change in angel θ is introduced in (Figure 3.14 (b), (c)) as

$$\Delta\theta \equiv \theta - \theta_0 \tag{3.65}$$

Then, applying the cosine law to Figure 3.15 (c) gives length d ($d \in [d_0, 2l]$) from

$$d^2 = r_v^2 + R_v^2 - 2r_v R_v \cos\Delta\theta \tag{3.66}$$

and then angle ψ from

$$r_v^2 = d^2 + R_v^2 - 2dR_v \cos\psi \tag{3.67}$$

Here, it is necessary to ensure that the sign of ψ equals to the sign of $\Delta\theta$.

From Figure 3.15 (d) and applying the cosine law, we find

$$l^2 = l^2 + d^2 - 2ld\cos\phi \Rightarrow \cos\phi = \frac{d}{2l} \tag{3.68}$$

Finally, the guide vane angle can be found as

$$\alpha_1 = \phi - \psi \tag{3.69}$$

In the above model, it has been assumed that the guide vane is perpendicular to the attached "arm" of length l, and that in position "0", a guide vane is at position "9 o'clock", Figure 3.15 (a), [1].

Hence, together the Francis turbine model, the turbine design algorithm and the guide vane actuation (servo position) are realized in the *Francis* turbine element in our library. In this *Francis* unit, the multi-physic *TurbineContacts* connectors model is also used and ensures connection to other waterway and electro-mechanical units. In addition, this *Francis* unit has also the standard Modelica *RealInput* connector that describes the angular velocity as an input to the Francis turbine model. Typically, this angular velocity connector is based on the derived info from (connected to) the generator units.

In the *Francis* unit, the user can specify the required nominal parameters for the Francis turbine: nominal turbine net head (nominal pressure drop), nominal turbine flow rate, nominal power, and nominal rotational speed. Then, the user can either choose to use the design algorithm that automatically defines the turbine geometry parameters (radius of the turbine blade inlet and outlet, the width of the turbine/blades inlet, the turbine inlet and outlet blade angles), or specify these turbine geometries manually. Similarly, the user has the same options for the losses coefficients and parameters for the guide vane actuation (servo position) model.

3.4.3 Pelton

Similar to the Francis turbine model, the mechanistic Pelton turbine model is developed and used. The key quantities of the model are shown in Fig. 3.16, and the shaft power $\dot{W}_s$ produced in the Pelton turbine is defined as follows, [1]:

$$\dot{W}_s = \dot{m} v_R \left[\delta(u_\delta) \cdot v_1 - v_R\right](1 - k\cos\beta) \tag{3.70}$$

Here, $\dot{m}$ is the mass flow rate through the turbine. The reference velocity is equal to $v_R = \omega R$: here, R is a is the radius of the rotor where the mass hits the bucket and ω is the angular velocity that is normally constrained by the grid frequency. The water velocity at position "1" (Figure 3.16) is equal to $v_1 = \frac{\dot{V}}{A_1}$, where $\dot{V}$ is the volumetric flow rate through the turbine and A_1 is a cross-sectional area at position "1" (the end of the nuzzle). β is the reflection angle with typical value of $\beta = 165°$, and $k < 1$ is some friction factor, typically $k \in [0.8, 0.9]$, [1]. In practical installations, there is a deflector mechanism to reduce the velocity $v_1\delta(u_\delta)$ to avoid over-speed.

The total work rate $\dot{W}_t$ removed through the turbine is:

$$\dot{W}_t = \dot{W}_s + \dot{W}_{ft} \tag{3.71}$$

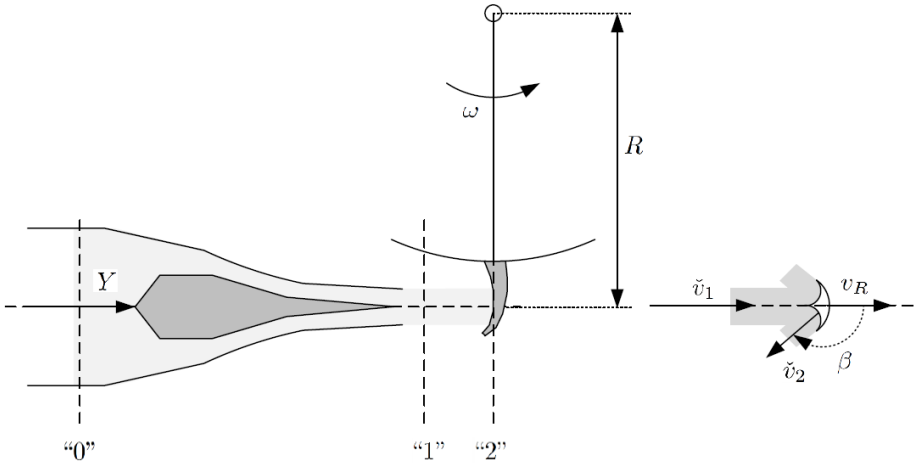

Figure 3.16: Some key concepts of the Pelton turbine, [1].

Here, $\dot{W}_{ft}$ is a friction of losses that can be found as follows:

$$\dot{W}_{ft} = K\,(1 - k\cos\beta)\,\dot{m}v_R^2 \tag{3.72}$$

Here, friction coefficient K equals 0.25, [1].

In addition, the pressure drop across the nozzle (positions "0" and "1'') Δp_n can be found as follows, [1]:

$$\Delta p_n = \frac{1}{2}\rho\dot{V}\left[\dot{V}\left(\frac{1}{A_1^2(Y)} - \frac{1}{A_0^2}\right) + k_f\right] \tag{3.73}$$

Here, A_0 is a cross sectional area at position "0" (the beginning of the nuzzle). $A_1(Y)$ means that the cross-sectional area at position "1" is a function of the needle position Y. k_f is a coefficient of friction loss in the nuzzle.

Hence, this Pelton turbine model is realized in the *Pelton* turbine element in our library. In this *Pelton* unit, the multi-physic *TurbineContacts* connectors model is also used and ensures connection to other waterway and electro-mechanical units. In addition, this *Pelton* unit also has the standard Modelica *RealInput* connector that describes the angular velocity as an input to the Francis turbine model. Typically, this angular velocity connector is based on the derived info from (connected to) the generator units.

In the *Pelton* unit, the user can specify the required geometry for the Pelton turbine: radius of the turbine runner, input diameter of the nuzzle, runner bucket angle, friction factors and coefficients, and deflector mechanism coefficient.

3.4.4 Simple Generator

Here, a simple model of an ideal generator with friction is considered. This model has inputs as electric power available on the grid and the turbine shaft power. This model is based on the angular momentum balance which depends on the turbine shaft power, the friction loss in the aggregate rotation, and the power taken up by the generator. The rotor angular velocity mainly depends on its inertia, internal friction and available power. The kinetic energy stored in the rotating generator is $K_a = \frac{1}{2} J_a \omega_a^2$, where ω_a is the angular velocity of the rotor and J_a is its moment of inertia. The kinetic energy K_a is changed by the power terms operating on the generator axis, e.g., the turbine shaft power $\dot{W}_s$ produced by the turbine, friction power $\dot{W}_{f,a}$, and the power taken up by the generator, $\dot{W}_g$, [1], and from energy the balance can be expressed as follows:

$$\frac{dK_a}{dt} = \dot{W}_s - \dot{W}_{f,a} - \dot{W}_g \tag{3.74}$$

$\dot{W}_{f,a}$ is the frictional power loss in the rotor. This frictional power loss is mainly due to losses in the shaft supporting bearings, losses in the transmission gearboxes and losses in the windage (air gap). For simplicity, it is assumed that the bearing term is dominating, and express $\dot{W}_{f,a}$ as

$$\dot{W}_{f,a} = \frac{1}{2} k_{f,b} \omega_a^2 \tag{3.75}$$

Here, $k_{f,b}$ is the bearing friction factor. The power taken up by the generator is transmitted to the grid with electric efficiency η_e. Thus the electric power available on the grid is $\dot{W}_e = \eta_e \dot{W}_g$.

Hence, this simple generator model is encoded in the *OpenHPL* as a *SimpleGen* unit. This unit has inputs as electric power available on the grid and the turbine shaft power which both are implemented with the standard Modelica *RealInput* connector. This *SimpleGen* unit also uses the standard Modelica *RealOutput* connectors in order to provide output information about the angular velocity and frequency of the generator. All these connectors can be connected to turbines units and other standard Modelica blocks.

In the *SimpleGen* unit, the user can specify the required parameters for the generator: moment of inertia of the generator, generator's electrical efficiency, friction factor in the rotor bearing box, the number of the generator poles. This unit can be initialised by the initial value of the angular velocity ω_0. Otherwise, the user can decide on an option when the simulation starts from a steady-state and the OpenModelica automatically handles the initial steady-state values (does not work properly in OpenModelica).

3.4.5 Synchronize Generator

Here, a more detailed model of the synchronous generator is presented. More details in the Behzad Sharefi master thesis, [6]. This model is based on the d-q decomposition and

assumed that the generator is connected to the grid, [6]. The voltage-current relation is given as:

$$\begin{bmatrix} R_a + R_e & x'_q + x_e \\ -x'_d - x_e & R_a + R_e \end{bmatrix} \begin{bmatrix} I_d \\ I_q \end{bmatrix} = \begin{bmatrix} E'_d + V_s \sin\delta_e \\ E'_q - V_s \cos\delta_e \end{bmatrix} \tag{3.76}$$

Here, R_a and R_e are the phase winding and equivalent network resistances, x_d, x_q, x'_d, and x'_q are d-/q-axis normal, and transient reactances. x_e is the equivalent network reactance. I_d and I_q are the d-/q-axis currents. E'_d and E'_q are the d-/q-axis transient voltages. V_s is the network RMS (Root-Mean-Squared) voltage. δ_e is the phase shift angle that is described as follows:

$$\frac{d\delta_e}{dt} = (\omega - \omega_s)\frac{n_p}{2} \tag{3.77}$$

Here, n_p is the number of poles in the generator, where ω and ω_s are the generator and grid angular velocities, respectively. The Swing equation is used to describe the angular velocity dynamics and looks as follows:

$$\frac{d\omega}{dt} = \frac{\dot{W}_s - P_e}{J\omega} \tag{3.78}$$

The dynamic equations for the transient operation are as follows:

$$\begin{aligned} T'_{qo}\frac{dE'_d}{dt} &= -E'_d + (x'_q - x_q)I_q \\ T'_{do}\frac{dE'_q}{dt} &= -E'_q + (x_d - x'_d)I_d + E_f \end{aligned} \tag{3.79}$$

Here, T'_{do} and T'_{qo} are the d-/q-axis transient open-circuit time constants. E_f is the voltage across the field winding with the following dynamic equation:

$$\frac{dE_f}{dt} = \frac{-E_f + K_E\left(V_{tr} - V_t - V_{stab}\right)}{T_E} \tag{3.80}$$

Here, K_E is the excitation system gain and T_E — excitation system time constant. V_{tr} is the voltage reference set point for the exciter. V_t is the terminal voltage and can be found as $V_t = \sqrt{\left(E'_d - R_a I_d - x'_q I_q\right)^2 + \left(E'_q - R_a I_q + x'_d I_d\right)^2}$. V_{stab} is the stabilisation voltage with the following dynamic equation:

$$\frac{dV_{stab}}{dt} = \frac{-V_{stab} + K_F\frac{dE_f}{dt}}{T_{FE}} \tag{3.81}$$

Here, K_F is the stabiliser gain, and T_{FE} — the stabiliser time constant.

The output active and reactive power of the generator can be found as follows:

$$\begin{aligned} P_e &= 3\left(E'_d I_d + E'_q I_q\right) \\ Q_e &= \sqrt{9V_t^2 I_t^2 - P_e^2} \end{aligned} \tag{3.82}$$

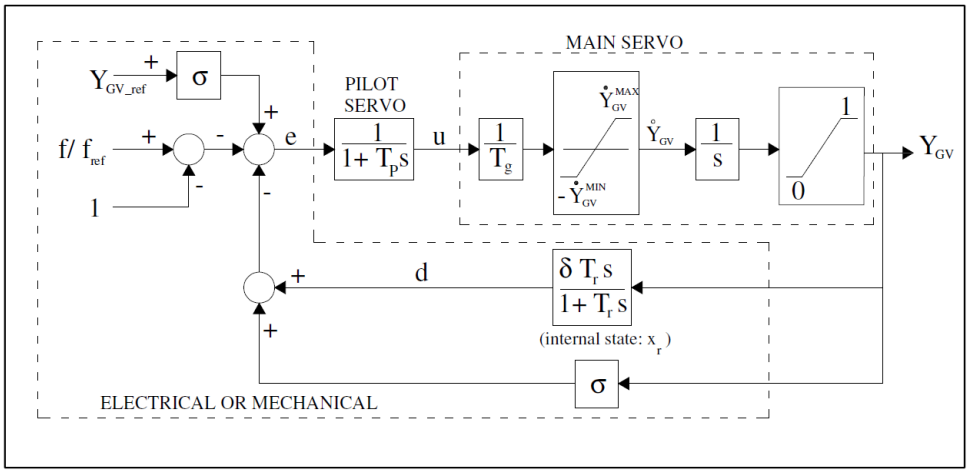

Figure 3.17: Block Diagram of the governor, [6].

Here, the terminate current is given as $I_t = \sqrt{I_d^2 + I_q^2}$.

Hence, this synchronise generator model is encoded in the *OpenHPL* as a*SynchGen* unit. This unit has inputs as the turbine shaft power, that is implemented with the standard Modelica *RealInput* connector. This *SynchGen* unit also uses the standard Modelica *RealOutput* connectors in order to provide output information about the angular velocity and frequency of the generator. All these connectors can be connected to turbines units and other standard Modelica blocks.

In the *SynchGen* unit, the user can specify the required nominal parameters for the generator: active and reactive powers drawn from the generator at Steady-State operating condition, phase winding resistance, and the number of poles. The following network parameters should be also specified by the user: equivalent network resistance and reactance, network RMS voltage, grid angular velocity. The user also specifies the d-/q-axis normal and transient reactances, d-/q-axis transient open-circuit time constants, minimum and maximum field voltages, excitation system, stabilizer gains, time constants, moment of inertia of the generator, and the friction factor in the rotor bearing box. This unit can be initialized, or the user can decide on an option for the self initialisation.

3.5 Governor

Here, a simple model of the governor that controls the guide vane opening in the turbine based on the reference power production is described. More details in the Behzad Sharefi master thesis, [6]. The block diagram of this governor model is shown in Figure 3.17.

Using the model in Figure 3.17 and the standard Modelica blocks, the governor model is encoded in our library as the *Governor* unit. This unit has inputs as the reference power production and generator frequency that are implemented with the standard Modelica *RealInput* connector. This *Governor* unit also uses the standard Modelica *RealOutput* connectors in order to provide output information about the turbine guide vane opening.

In the *SynchGen* unit, the user can specify the various time constants of this model (see Figure 3.17): pilot servomotor time constant T_p, primary servomotor integration time T_g, and transient droop time constant T_r. The user should also provide the following parameters: droop value σ, transient droop δ, and nominal values for the frequency and power generation. The information about the maximum, minimum, and initial guide vane opening should also be specified.

3.6 Examples

Here, various models that have been assembled in the *Examples* class are described.

3.6.1 HPSimple

In this model of the hydropower system, the simplified models are used for conduits and turbine modelling. The generator is not included in the model. The simple *Pipe* unit is used to represent the penstock, intake and discharge races. The simple *Turbine* unit is used to represent the turbine. The *Reservoir* unit is used to represent the reservoir and the tailwater (here, this unit uses a simple model of the reservoir that only depends on the water depth in the reservoir). Data from the Sundsbarm hydropower plant is used for this example model.

3.6.2 HPSimple_generator

In this model of the hydropower system, the simplified models are used for conduits, turbine, and generator modelling. The simple *Pipe* unit is used to represent the penstock, intake and discharge races. The simple *Turbine* unit is used to represent the turbine. The *SimpleGen* unit is used to represent the generator. The *Reservoir* unit is used to represent the reservoir and tailwater (here, this unit uses a simple model of the reservoir that only depends on the water depth in the reservoir). Data from the Sundsbarm hydropower plant is used for this example model.

3.6.3 HPSimple_Francis

In this model of the hydropower system, the simplified model is used for conduits modelling. The turbine and the generator are modelled with more detailed *Francis* and *SynchGen* units, respectively. The simple *Pipe* unit is used to represent the penstock, intake and discharge races. The *Reservoir* unit is used to represent the reservoir and the tailwater (here, this unit uses a simple model of the reservoir that only depends on the water depth in the reservoir). Data from the Sundsbarm hydropower plant is used for this example model.

3.6.4 HPDetailed

In this model of the hydropower system, the simplified models are used for conduits and turbine modelling, except for the penstock that is modelled with the more detailed *PenstockKP* unit. The generator is not included in the model. The simple *Pipe* unit is used to represent the intake and discharge races. The simple *Turbine* unit is used to represent the turbine. The *Reservoir* unit is used to represent the reservoir and the tailwater (here, this unit uses a simple model of the reservoir that only depends on the water depth in the reservoir). Data from the Sundsbarm hydropower plant is used for this example model.

3.6.5 HPDetailed_generator

In this model of the hydropower system, the simplified models are used for conduits, turbine, and generator modelling, except for the penstock that is modelled with the more detailed *PenstockKP* unit. The simple *Pipe* unit is used to represent the intake and discharge races. The simple *Turbine* unit is used to represent the turbine. The *SimpleGen* unit is used to represent the generator. The *Reservoir* unit is used to represent the reservoir and the tailwater (here, this unit uses a simple model of the reservoir that only depends on the water depth in the reservoir). Data from the Sundsbarm hydropower plant is used for this example model.

3.6.6 HPDetailed_Francis

In this model of the hydropower system, the simplified model is used for conduits modelling, except for the penstock that is modelled with the more detailed *PenstockKP* unit. The turbine and generator are modelled with more detailed *Francis* and *SynchGen* units, respectively. The simple *Pipe* unit is used to represent the intake and discharge races. The *Reservoir* unit is used to represent the reservoir and the tailwater (here, this unit uses

a simple model of the reservoir that only depends on the water depth in the reservoir). Data from the Sundsbarm hydropower plant is used for this example model.

3.6.7 HPSimple_Francis_IPSLGen

Here, the last example model (uses the *PenstockKP* and *Francis* units) is extended by synergy with the *OpenIPSL* for generator and power system modelling. The *Governor* unit from the *OpenHPL* is also used here. The penstock is modelled with the more detailed *PenstockKP* unit. The turbine is modelled with more detailed *Francis* unit. The simple *Pipe* unit is used to represent the intake and discharge races. The *Reservoir* unit is used to represent the reservoir and the tailwater (here, this unit uses a simple model of the reservoir that only depends on the water depth in the reservoir).

3.6.8 HPSimple_Francis_GridGen

Here, the *HPDetailed_Francis* example model (uses the *PenstockKP* and *Francis* units) is also extended by synergy with the *OpenIPSL* for only generator modelling. The *Governor* unit from the *OpenHPL* is also used here. The penstock is modelled with the more detailed *PenstockKP* unit. The turbine is modelled with the more detailed *Francis* unit. The simple *Pipe* unit is used to represent the intake and discharge races. The *Reservoir* unit is used to represent the reservoir and the tailwater (here, this unit uses a simple model of the reservoir that only depends on the water depth in the reservoir).

3.6.9 HPSimple_Francis_IPSLGenGov

Here, the *HPDetailed_Francis* example model (uses the *PenstockKP* and *Francis* units) is extended by synergy with the *OpenIPSL* for generator, governor and power system modelling. The penstock is modelled with the more detailed *PenstockKP* unit. The turbine is modelled with the more detailed *Francis* unit. The simple *Pipe* unit is used to represent the intake and discharge races. The *Reservoir* unit is used to represent the reservoir and the tailwater (here, this unit uses a simple model of the reservoir that only depends on the water depth in the reservoir).

3.6.10 HPSimple_Francis_IPSLGenInfBus

Here, the *HPDetailed_Francis* example model (uses the *PenstockKP* and *Francis* units) is extended by synergy with the *OpenIPSL* for generator and power system (including infinite bus) modelling. The *Governor* unit from the *OpenHPL* is also used here. The

penstock is modelled with the more detailed *PenstockKP* unit. The turbine is modelled with the more detailed *Francis* unit. The simple *Pipe* unit is used to represent the intake and discharge races. The *Reservoir* unit is used to represent the reservoir and the tailwater (here, this unit uses a simple model of the reservoir that only depends on the water depth in the reservoir).

3.6.11 HPSimple_OpenChannel

In this model of the hydropower system, the simplified models are used for conduits and turbine modelling. The generator is not included in the model. The simple *Pipe* unit is used to represent the penstock and intake race. The discharge race is an open channel here, and the *OpenChannel* unit is used for modelling. The simple *Turbine* unit is used to represent the turbine. The *Reservoir* unit is used to represent the reservoir and the tailwater (here, this unit uses a simple model of the reservoir that only depends on the water depth in the reservoir).

4 Basic example

Here, a basic (step-by-step) example is provided in order to show how to connect and specify elements from the *OpenHPL* in a flowsheet. Furthermore, an example of how to set up the OMPython API is also presented.

4.1 Flowsheet

In order to create a flowsheet model for the hydropower system in OpenModelica using the *OpenHPL*, the following steps should be performed:

1. Create a new Modelica class that is specified as "Model" and assign a name for this model. Then, open this model with the "Diagram view". See example in Figure 4.1.

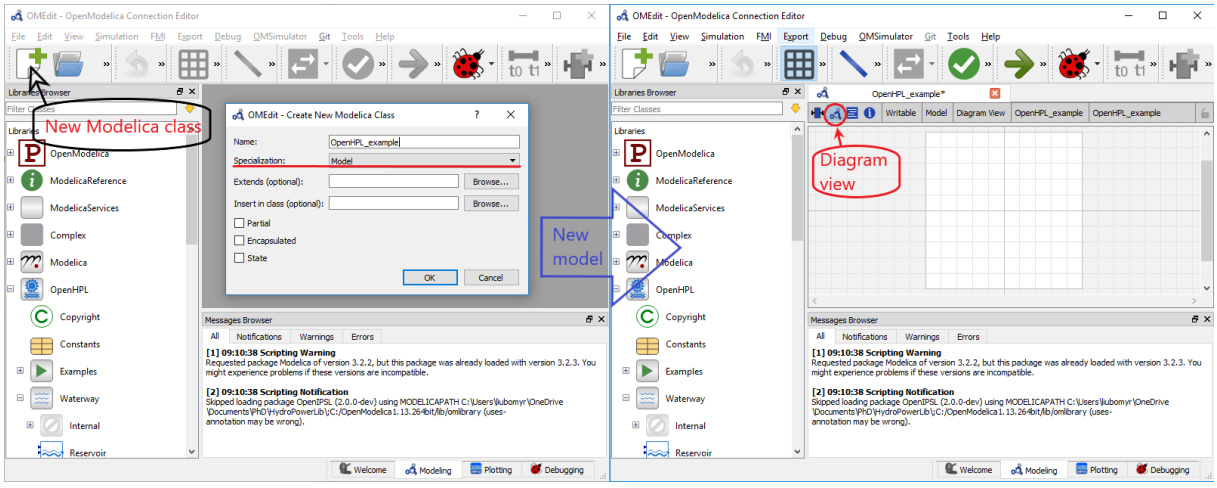

Figure 4.1: Creating a new model in OpenModelica.

2. Drag and drop all of the needed elements for the hydropower structure from the *OpenHPL* and provide a name for each element. Then, connect the connectors of these elements between each other. See example in Figure 4.2.

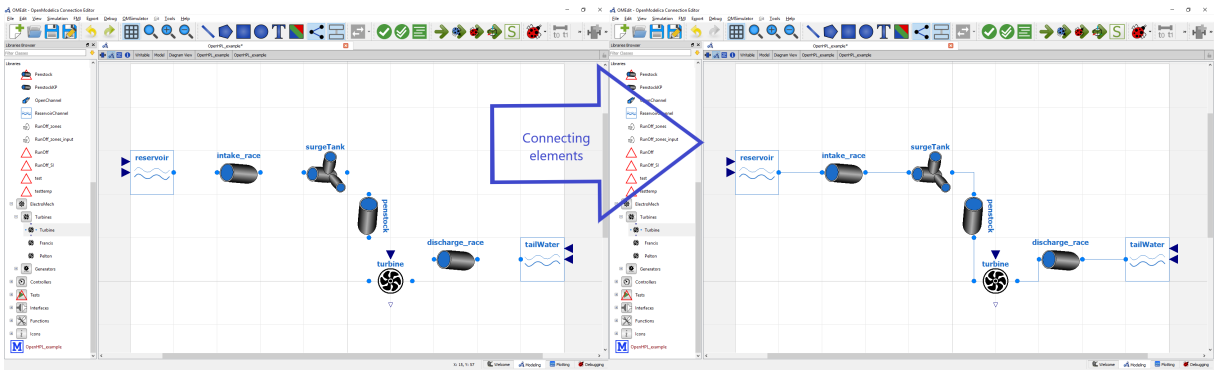

Figure 4.2: Connecting the elements of the hydropower system.

3. Also, insert the records model “Constants” from the *OpenHPL* with the name “Const” to the model in order to have control on some constants and properties that are common for all hydropower elements. As an example in case, typical initial value of the volumetric flow rate in the system for each “Pipe” unit can be specified.

4. Specify each of the elements with an appropriate geometry. See example for the specification of the intake race element in Figure 4.3.

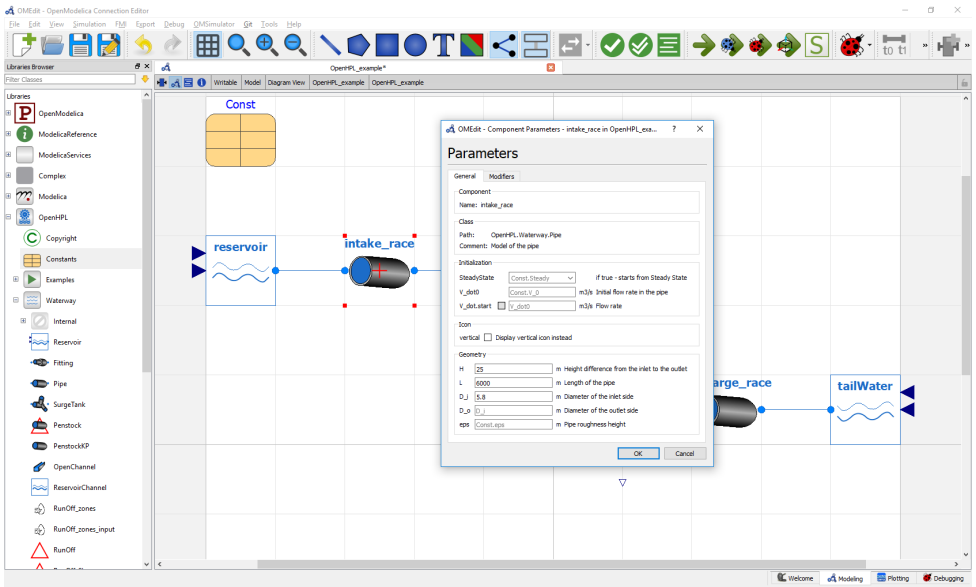

Figure 4.3: Specification of the elements.

5. Provide a control signal for the turbine. To make this, you can either add a source of the ramp signal from the standard Modelica library (“Modelica.Blocks.Sources. Ramp”), or create an input variable (or just a simple variable) for the example model

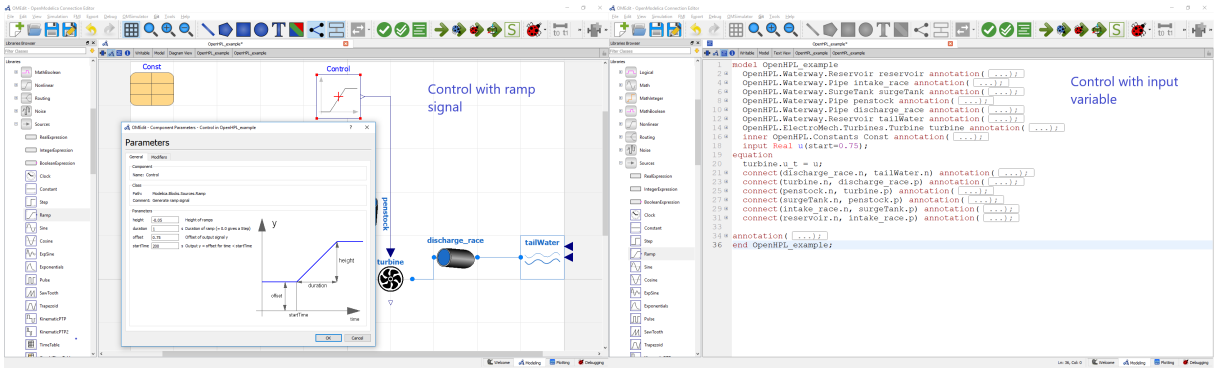

Figure 4.4: Creating a control signal for the turbine.

"OpenHPL example" and equate it to the turbine control input. Both possibilities are shown in Figure 4.4.

6. Specify the simulation setup values and save it in the model. Then, the simulation can be carried out. See example for the simulation specification and running in Figure 4.5.

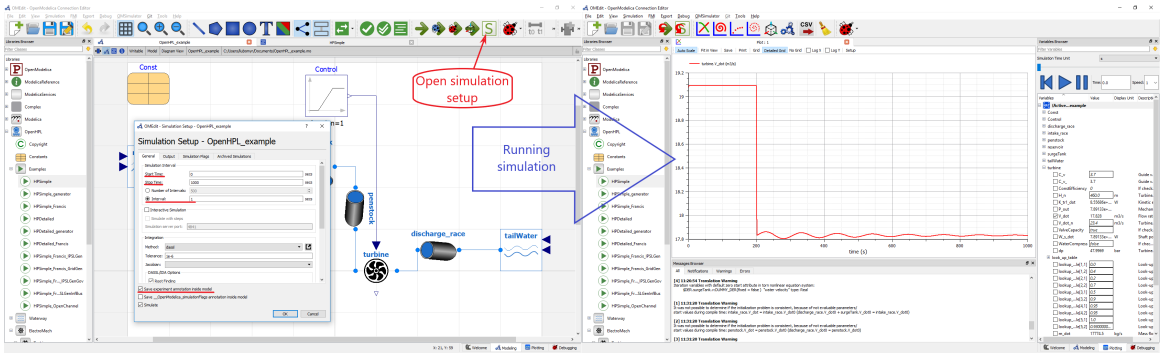

Figure 4.5: Specifying and running simulation.

4.2 OMPython API

In order to run the simulations of the developed example hydropower model from Python, the OMPython API for OpenModelica can be used. The following steps should be performed to set up the API:

1. Import the "Modelica system" environment form the OMPython package. Then, create an object in Python of the OpenModelica model "OpenHPL example". Here, the libraries that are used in the model should also be loaded to the object which

in this case, is the standard Modelica library and the *OpenHPL*. See an example of the code below:

```
from OMPython import ModelicaSystem
hps_s = ModelicaSystem("OpenHPL_example.mo", "
   OpenHPL_example", ["Modelica", "OpenHPL/package.mo
   "])
```

2. When the object is created, the simulation options, as well as the parameters and input variables can be specified. In order to check and specify the simulation options, the following commands can be used:

```
hps_s.setSimulationOptions(stepSize=0.1, stopTime
   =1000) # set simulation options
hps_s.getSimulationOptions() # get list of simulation
    options
```

   Similar commands for parameters and input variables look as follows:

```
hps_s.getParameters() # get list of model parameters
hps_s.setParameters(**{"turbine.H_n":460}) # set
   parameter value for the turbine nominal head
hps_s.getInputs() # get list of input variables
hps_s.setInputs(u=[(0,0.75),(100,0.75),(101,0.7)
   ,(1000,0.7)]) # set input value over time as a
   ramp signal
```

   It should be noted that here, I used the model with input variable for the control signal of the turbine (see item #5 and Figure 4.4 in the previous flowsheet section).

3. Run the simulation and get the results. An example of these commands are carried out as follows:

```
hps_s.simulate() # run simulation
hps_s.getSolutions() # get list of solution variables
time, Vdot, p_tr1, p_tr2 = hps_s.getSolutions("time",
    "turbine.V_dot", "turbine.p_tr1", "turbine.p_tr2"
   ) # get results of simulation time variable, and
   the turbine flow rate, inlet and outlet pressures.
```

   These simulation results can be then plotted using *matplotlib* package. See the plots of the turbine flow rate and the pressures in Figure 4.6.

4. It is also possible to linearize the model for the future analysis. The linearization can be done with one command. However more commands can also be used to

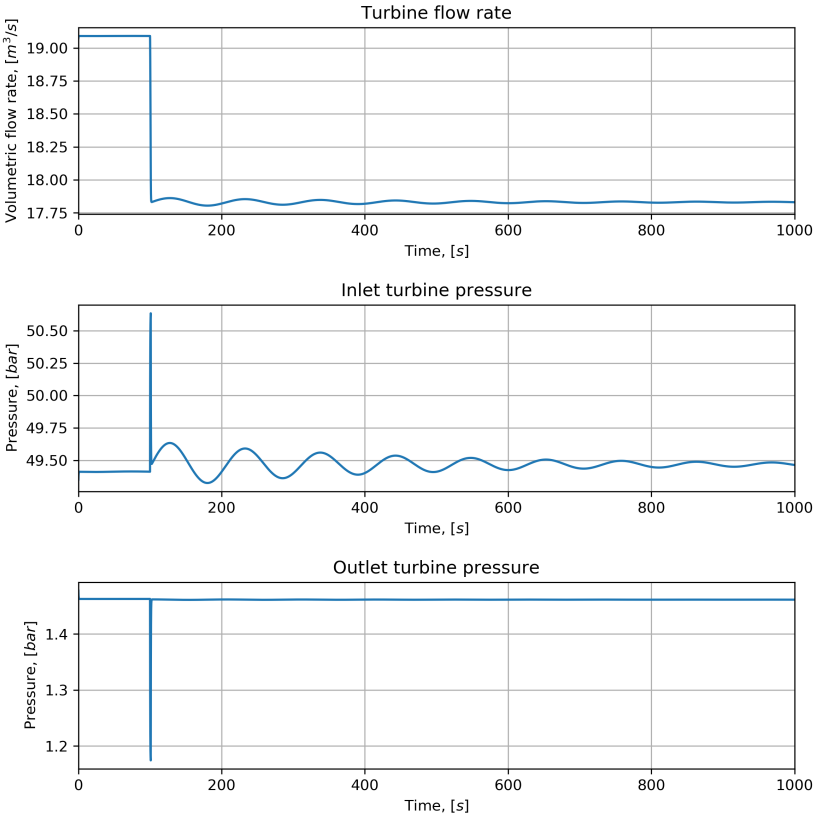

Figure 4.6: Plotting of simulation results.

check/specify the linearization options and define the states/inputs/outputs. See the example below:

```
hps_s.setLinearizationOptions(stopTime=0.1) # set a
   stop time for linearization (linearization is
   performed in this point)
hps_s.getLinearizationOptions() # get a list of the
   linearization options
As,Bs,Cs,Ds = hps_s.linearize() # actual
   linearization; defining standard A, B, C and D
   matrices.
hps_s.getLinearStates() # get list of states
hps_s.getLinearInputs() # get list of inputs
hps_s.getLinearOutputs() # get list of outputs
```

It should be noted that the linearized model should include the input variable which in this case is the input variable for the turbine control signal.

Similar to OMPython API, the running of OpenModelica models in Julia using OMJulia API can also be carried out. See the documentation of OMJulia for more information.

Bibliography

[1] B. Lie, 'Lecture notes in course FM1015 Modelling of Dynamic Systems', University of South-Eastern Norway, Porsgrunn, Norway, Tech. Rep., 2018.

[2] R. Sharma, 'Second order scheme for open channel flow', Porsgrunn: Telemark University College, Tech. Rep., 2015. [Online]. Available: http://hdl.handle.net/11250/2438453.

[3] L. Vytvytskyi, R. Sharma and B. Lie, 'Model based control for run-of-river system. Part 1: Model implementation and tuning', *Modeling, Identification and Control: A Norwegian Research Bulletin*, vol. 36, no. 4, pp. 237–249, 2015. DOI: 10.4173/mic.2015.4.4.

[4] L. Vytvytskyi and B. Lie, 'Comparison of elastic vs. inelastic penstock model using OpenModelica', in *Proceedings of the 58th Conference on Simulation and Modelling (SIMS 58) Reykjavik, Iceland, September 25th–27th, 2017*, Linköping University Electronic Press, Linköpings Universitet, 2017, pp. 20–28. DOI: 10.3384/ecp1713820.

[5] V. Splavska, L. Vytvytskyi and B. Lie, 'Hydropower Systems: Comparison of Mechanistic and Table Look-up Turbine Models', in *Proceedings of the 58th Conference on Simulation and Modelling (SIMS 58) Reykjavik, Iceland, September 25th–27th, 2017*, Linköping University Electronic Press, Linköpings Universitet, 2017, pp. 368–373. DOI: 10.3384/ecp17138368.

[6] B. R. Sharefi, 'Modeling for Control of Hydropower Systems', Master's thesis, Telemark University College, Porsgrunn, Norway, 2011.

[7] S. Shafiee, 'Automatic updating of hydrological models for runoff/inflow forecasting to hydropower system', Master's thesis, Telemark University College, Porsgrunn, Norway, 2013.

[8] L. Vytvytskyi and B. Lie, 'OpenHPL for modelling Trollheim hydropower plant', *Submitted to Energies*, 2019, ISSN: 1996-1073.

[9] ——, 'Mechanistic model for Francis turbines in OpenModelica', *IFAC-PapersOnLine*, vol. 51, no. 2, pp. 103–108, 2018. DOI: 10.1016/j.ifacol.2018.03.018.

[10] H. Brekke, *Hydraulic Turbines. Design, Erection and Operation*. Trondheim, Norway: Norwegian University of Science and Technology, 2001.