Daniel Christen

Modular GUI for hydro power planning

Semester Thesis

EEH – Power Systems Laboratory
Swiss Federal Institute of Technology (ETH) Zurich

Expert: Prof. Dr. Göran Andersson
Supervisor: Hubert Abgottspon

Zurich, February 22, 2013
Abstract

The main goal of this semester project is the implementation of an already existing software tool, written in Matlab, which calculates optimal use of hydro power plants, into a Java application. A second goal is to offer the same functionality for any power plant instead of only one specific.

A modular functionality is implemented in the new software. The user can build his own power plant out of its elements (reservoirs, turbines, pumps) graphically.
Furthermore, a short-term optimization is included, which means that a time period in the range of a few hours is considered in which a known demanded power is produced with minimal production cost.

First, the power plant model and the mathematical representation of the optimization problem is discussed.
In a further section, the realization in Java is explained on an example power plant composition. The first part of this section has its focus on the data structure and the graphical representation to offer the modular functionality.
The second part deals with the processing of the entered power plant data in order to be able to formulate the short-term optimization problem.
Finally, the program is applied to a power plant to show its functionality.
# Contents

1 Motivation and Objective ........................................ 1

2 Software Environment ........................................... 3
   2.1 Program Language ........................................ 3
   2.2 Development Tool: Eclipse .............................. 3

3 Hydro Power Plant Modelling .................................. 5
   3.1 Mathematical Model ...................................... 5
      3.1.1 Turbines .......................................... 7
      3.1.2 Pumps ............................................ 9
      3.1.3 Optimization Function ........................... 9
   3.2 CPLEX in Java .......................................... 11

4 Program Structure ............................................. 13
   4.1 Overview ............................................... 13
   4.2 Classes & Functions For Power Plant Building .......... 14
      4.2.1 Java List Function ................................ 15
      4.2.2 Class Mainframe .................................. 16
      4.2.3 Class PowerPlant .................................. 16
      4.2.4 Class PPElement .................................. 18
      4.2.5 Classes Reservoir, Turbine, Pump, Pipe .......... 19
      4.2.6 Class PipeNode ................................... 19
      4.2.7 GUI Classes ....................................... 19
      4.2.8 Other Functions ................................... 20
      4.2.9 Class DrawPanel ................................... 20
      4.2.10 Class XYChart .................................... 21
   4.3 Optimization Class HydroSolver .......................... 21
      4.3.1 Detecting Flows ................................... 21
Parameters

\( \pi^t \) production cost at stage \( t \)

\( c^1_{t,r} \) price for spilled water in reservoir \( r \) at stage \( t \) in \( \text{EUR} \frac{m^3}{1000m^3} \)

\( c^2_t \) price for power difference in \( \text{EUR} \frac{MW}{1000m^3} \) at stage \( t \)

\( u^t_r \) outflow in reservoir \( r \) in \( \frac{m^3}{h} \) at stage \( t \)

\( s^t_r \) spilled water of reservoir \( r \) in \( \frac{m^3}{h} \) at stage \( t \)

\( \gamma^t_r \) water value for reservoir \( r \) in \( \text{EUR} \frac{1000m^3}{1000m^3} \) at stage \( t \)

\( P^t_{diff} \) difference produced-demanded power at stage \( t \) [MW]

\( i^t_r \) inflow in reservoir \( r \) in \( \frac{1000m^3}{h} \) at stage \( t \)

\( w^t_r \) water in reservoir \( r \) in \( 1000m^3 \) at stage \( t \)

\( h^t_r \) actual water level of reservoir \( r \) in meter above sea [mas] at time \( t \)

\( h^{max,r} \) maximum water level of reservoir \( r \) in meter above sea [mas]

\( h^{min,r} \) minimum water level of reservoir \( r \) in meter above sea [mas]

\( q^t_{i,l} \) turbined water of block \( l \) of turbine \( i \) at stage \( t \)

\( q^t_{p,i,l} \) pumped water of block \( l \) of pump \( i \) at stage \( t \)

\( q^{max,pipe,i} \) maximum throughflow of pipe \( i \)

\( Q^{max,p,i} \) maximum throughflow of pump \( i \)

\( Q^{max,t,i} \) maximum throughflow of turbine \( i \)

\( Q^{min,p,i} \) minimum throughflow of pump \( i \)

\( Q^{min,t,i} \) minimum throughflow of turbine \( i \)

\( P^t_{t,i} \) power produced by turbine \( i \) at time \( t \)

\( P^t_{p,i} \) power consumed by pump \( i \) at time \( t \)

\( \eta_{t,i} \) efficiency of turbine \( i \)

\( \eta_{p,i} \) efficiency of pump \( i \)

\( \eta_{pipe,i} \) efficiency of pipe \( i \)

\( h_{t,i} \) height of turbine \( i \) in meter above sea [mas]

\( h_{p,i} \) height of pump \( i \) in meter above sea [mas]

\( D^p_t \) Demanded Power at stage \( t \)

\( t^t_{i,l} \) binary variable if block \( l \) of turbine \( i \) runs or not (e.g. maintenance)

\( p^t_{l,i} \) binary variable if block \( l \) of pump \( i \) runs or not

\( a_{t,i,l} \) variable \( a \) for block \( l \) of turbine \( i \) for linearized function \( y = a + b \cdot x \)

\( a_{p,i,l} \) variable \( a \) for block \( l \) of pump \( i \) for linearized function \( y = a + b \cdot x \)

\( b_{t,i,l} \) variable \( b \) for block \( l \) of turbine \( i \) for linearized function \( y = a + b \cdot x \)

\( b_{p,i,l} \) variable \( b \) for block \( l \) of pump \( i \) for linearized function \( y = a + b \cdot x \)

Sets:

\( R \) number of reservoirs

\( T \) number of turbines

\( P \) number of pumps
Chapter 1

Motivation and Objective

Motivation
The power market is more and more liberalized and leads to an increasing competition for power plant operators with other power suppliers. To operate a power plant at maximal cost efficiency, many algorithms are developed. At the eeh institute at ETH Zürich, there is such an optimization function implemented in a Matlab environment. Power plant operators usually don’t use Matlab in their daily business because of its complexity, therefore an approach to offer the same function in a different free to use software, in this case java is done.

In this thesis, the short-term optimization is considered, which is about optimal production scheduling for the day ahead. The used software environment is explained in chapter 2 in chapter 3 the mathematical background is presented and its implementation in the software is described in chapter 4.

Objective
This project aims to offer the same functionality like the already available Matlab algorithm described in the motivation extended to:

- without Matlab
- for any possible hydro power plant

A key element is the ability to let the user build his own power plant graphically in the program to satisfy the functionality for every composition. This
data is analysed and a mixed integer problem is formulated out of it to calculate the optimal use of the turbines and pumps.
Chapter 2

Software Environment

2.1 Program Language

Java has been selected as the program language. The advantages of using Java are:

- very good multi platform compatibility
- cost-free
- object oriented language which allows to write modular code
- it’s simpler than C++ due to automatic memory allocation and garbage collection

2.2 Development Tool: Eclipse

As development tool, Eclipse\(^3\) was used. It is a free open-source tool which is itself written in java and provides many plugins to help the user writing code.

For building the GUI, the WindowBuilder\(^4\) plugin was very helpful. It is developed by Google Inc.\(^7\) and offers drag and drop functionality to build a GUI in eclipse. When adding e.g. a button, the needed code lines are automatically generated and the result is visualized. Of course, in many cases it is better to write or edit code by hand to achieve the desired look in the GUI.
In the Figure 2.1, the interface of eclipse with windowbuilder is shown. The grey vertical bar (area 1) includes the swing GUI elements which can be added by drag and drop to the designed frame (area 2).
Chapter 3

Hydro Power Plant Modelling

In this chapter, the used mathematical model of a power plant is presented. Furthermore, the equations used to calculate the produced power out of the potential energy stored in the water are explained.

3.1 Mathematical Model

Short-term planning considers only a short time period - normally a day or a few hours - for the optimization and is normally done day-ahead. This leads to the assumption that the power demand is known because of already closed contracts.

In this chapter, the mathematical model of a power plant which is used in the program is described, its implementation is explained later in the chapter 4.

Figure 3.1 gives an overview over all models required for the optimization.
Figure 3.1: Power plant system model.
3.1.1 Turbines

The potential energy stored in the reservoirs is transformed into kinetic energy by penstocks which itself is transformed into energy by turbines. Written as a formula, this is:

\[ E_{\text{turbine}} = m_{\text{water}} \cdot g \cdot h_{\text{diff,}t} \cdot \eta_{\text{pipe}} \cdot \eta_{t} \quad [J] \quad (3.1) \]

where \( m_{\text{water}} \) is the mass of the water in 1000 kg/m\(^2\), \( g \) is the earth acceleration \( g = 9.81 \text{ m/s}^2 \) and \( h_{\text{diff,}t} \) is the height difference between the water source and the turbine:

\[ h_{\text{diff,}t} = h_{\text{source}} - h_{\text{turbine}} \quad [m] \]

To obtain the power, we divide the energy by the time. With \( q = \) amount of water which flows through the turbine per second in m\(^3\)/s follows:

\[ P_{\text{turbine}} = q \cdot g \left[ \frac{9.81 \text{ m}}{\text{s}^2} \right] \cdot h_{\text{diff,}t} [m] \cdot \eta_{\text{pipe}} [%] \cdot \eta_{t} \left[ \frac{\%}{\text{m}^3/\text{s}} \right] \quad [W] \quad (3.2) \]

If the produced power per water throughflow is plotted, we obtain the unit performance curve for the turbine.

An example is shown in Fig 3.2 where \( h_{\text{diff}} = 300 \text{ m} \) and efficiency curve shown in Fig. 3.3 was used.

![Figure 3.2: Example of a unit performance curve.](image-url)
To form in a later step a short-term optimization problem, we need a linear function. To achieve this, we approximate the performance curve with a piecewise linearization. Here an example of 4 pieces:

Making a piecewise linearization means that the curve is cut into the desired amount of pieces and for each piece a linear regression is calculated. For the performance curve it is important to have a good linearization, because it is a main element in the calculation. A simple linearization would
lead to a much greater error and would end up probably in a wrong result.

3.1.2 Pumps

Pumps work in the other direction than turbines (shown in Fig. 3.1). So you know how much power they consume and you can calculate how much water they pump into the reservoir.

The only difference in comparison to a turbine is the calculation of the height. For pumps, you have to consider the difference between the upper and the lower reservoir, not between the upper reservoir and the pump:

\[ h_{\text{diff},p} = h_{\text{destination}} - h_{\text{source}} \quad [m] \]  

(3.3)

The consumed power of pumps is therefore:

\[ P_{\text{pump}} = q_p \cdot g \left( \frac{m}{s^2} \right) \cdot h_{\text{diff},p} \cdot m \cdot \frac{1}{\eta_{\text{pipe}}} \cdot \frac{1}{\eta_p} \left( \frac{\%}{m^3/s} \right) [W] \]  

(3.4)

Note that the efficiencies are inverted because the pipe uses more power if they are low.

3.1.3 Optimization Function

The goal of the optimization is to produce the demanded power with minimal production costs. This is achieved by minimizing the following function:

\[ \pi^t = \arg \min_{u_t} \sum_{r \in R} u_t^r \cdot \gamma_r^t + s_r^t \cdot c_{1,r}^t + P_{\text{diff}}^t \cdot c_2 \]  

(3.5)

Where \( \pi \) is the production cost, \( u_r(t) \) is the water outflow of reservoir \( r \) at time \( t \), \( \gamma_r \) is the watervalue for reservoir \( r \), \( c_{1,r} \) is the lost value from spilled water, \( P_{\text{diff}} \) is the difference from produced power to demanded power and \( c_2 \) is the price for \( P_{\text{diff}} \) per MW.

This minimization problem is subject to several constraints:

1. Given total power production

\[ \sum_{i \in T, k \in P} ((P_{t,i}^l - P_{p,k}^l) = DP_{t}^l - P_{\text{diff}}^l \]  

(3.6)
The sum of produced energy in all turbines and consumed energy in all pumps must be equal to the demanded power $DP$, otherwise a difference $P_{\text{diff}}$ occurs which must be purchased on the market.

2. Water outflow of reservoir $r$

$$u^t_r = \sum_{i \in T, p \in P, \text{i,p connected to } r} q^t_i - q^t_p + s^t_r - i^t_r \quad (3.7)$$

The water outflow is equal to the water consumed by the turbines plus the water pumped in by the pumps plus the spilled water and minus the water inflows.

3. At each station either pumping or turbining is possible:

$$\forall r \in R : (\max_{t \in r}(t^t_i) + \max_{p \in r}(p^t_i)) \leq 1 \quad (3.8)$$

4. Water flows in turbines and pumps lays between defined limits

$$Q_{\text{min}, t, i} \leq Q^t_{i, i} \leq Q_{\text{max}, t, i}$$

if $t^t_i = 1$

$$Q_{\text{min}, p, i} \leq Q^t_{p, i} \leq Q_{\text{max}, p, i}$$

if $p^t_i = 1$
3.2 CPLEX in Java

CPLEX\(^6\) is a mathematical optimizer from IBM to solve among others also mixed integer problems. It offers a java api to implement the functionality into a java program. One file(cplex.jar) has to be imported into the eclipse project and a link to the function library file must be given (Java code: \texttt{System.loadLibrary(...);}). Important is that the correct library for the system architecture (32- or 64 Bit) and for the right OS (Windows/Mac) is loaded. This is done automatically by a function in the HydroSolver class. The library files have to be stored in the program directory in the following folders:

<table>
<thead>
<tr>
<th>Library Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 32Bit</td>
<td>/CPLEXlib/win/32Bit/</td>
</tr>
<tr>
<td>Windows 64Bit</td>
<td>/CPLEXlib/win/64Bit/</td>
</tr>
<tr>
<td>Mac OS 32Bit</td>
<td>/CPLEXlib/mac/32Bit/</td>
</tr>
<tr>
<td>Mac OS 64Bit</td>
<td>/CPLEXlib/mac/64Bit/</td>
</tr>
</tbody>
</table>

A short example how to use cplex in java is given below.

**Small Example for use CPLEX in JAVA:**

For maximizing a function like

$$3x_1 - 5x_2 + 7x_3$$ \hspace{1cm} (3.9)

with constraints

$$x_1 + 2x_2 \leq 10$$ \hspace{1cm} (3.10)
$$x_1 - x_2 + 6x_3 \leq 50$$ \hspace{1cm} (3.11)

and bounds

$$5 \leq x_1 \leq 30$$ \hspace{1cm} (3.12)
$$x_2 \geq 0$$ \hspace{1cm} (3.13)
$$x_3 \geq 0$$ \hspace{1cm} (3.14)

you have to set it up as follows:
1. Create the solver object:
   IloCplex cplex = new IloCplex();

2. Set upper and lower bounds:
   double[] ub={30.0, DOUBLE.MAX_VALUE, DOUBLE.MAX_VALUE};
   double[] lb={5.0, 0.0, 0.0};

3. create variable array with bounds:
   IloNumVar[] var = cplex.numVarArray(3, lb, ub);

4. create array of multiplicators for the variables:
   double[] varMult={3.0, -5.0, 7.0};

5. Hand over the values to the solver:
   cplex.addMaximize(cplex.scalProd(var, varMult));

6. Add constraints:
   cplex.addLe(cplex.sum(cplex.prod(1.0, var[0]), cplex.prod(2.0, var[1])), 10.0);
   cplex.addLe(cplex.sum(cplex.prod(1.0, var[0]), cplex.prod(-1.0, var[1]), cplex.prod(6.0, var[2])), 50.0);

7. Solve the problem and get the data:
   if(cplex.solve()){
      double[] solutionValues = cplex.getValues(var);
   }

In number 7, the cplex.solve() flag is true if a solution is found, otherwise it is false and no results are returned.
The solution values are returned in an array where the first one corresponds to $x_1$ the second to $x_2$ and the third to $x_3$. 
Chapter 4

Program Structure

4.1 Overview

As already mentioned, Java is an object oriented programming language. This turns out to be useful to build up a modular program where each element or object in reality (like reservoirs, turbines, pipes etc.) has its corresponding Java object.

Such an object is represented by a class which contains variables and functions. An example how such a class or object looks like is shown in Figure 4.1.

Figure 4.1: Example of an object in Java.
Apart from the elements which represents a real object, there are also objects needed to handle the interaction between user and software, calculate the optimal use or manage the interaction between other objects.

An overview over the whole program structure is shown in figure 4.2. The references to and dependencies with other classes are not shown for better readability.

![Figure 4.2: Overview of program structure.](image)

### 4.2 Classes & Functions For Power Plant Building

As can be seen in the overview (Figure 4.2) the program contains in total 19 classes. They are described in this section to give a more detailed view into the software functions.

First, an explanation of the classes needed for building a power plant in the program is given. In a further section, the focus lies on the solver class.
4.2.1 Java List Function

Later in this section, it is described how the power plant elements are saved in the PowerPlant class. This class uses dynamic lists to store the data. There exist two variants of lists in java:

**ArrayList**

This list is like a normal array with the difference that it has a dynamic size. The advantage of this list is the fast access to every element over the index number, but the disadvantage is that the whole list has to be rearranged if an element at the beginning or in the middle of the list is deleted. An ArrayList in java is initialized as follows:

\[
\text{List<type> listname=new ArrayList<type>;}
\]

**LinkedList**

The linked list has a node at the beginning which points to the first element. The first element points to the second, the second to the third and so on. The advantage is here that if an element in the middle of the list is removed, only the pointer from the element before has to be changed to the element after the removed one. This is a very fast process. A disadvantage is the access to elements in the middle or at the end of the list. You have to iterate through all elements before the desired one until you are at the right position which needs much time. The code for initializing a LinkedList is almost the same like the code for an ArrayList:

\[
\text{List<type> listname=new LinkedList<type>;}\]

Which list used in this program?

In this program, array lists are used, because deleting and adding files happens only when the power plant is build (only a few times), but the solver needs to access the elements quiet often.
4.2.2 Class Mainframe

This class handles the interaction between a user and the program. It includes the following functions:

- Load and store a built power plant

- Add power plant elements, connect and delete them

- Visualize the power plant with the ability to position the elements by moving them around by mouse

- Call other functions from other objects needed for functionality

4.2.3 Class PowerPlant

Stores the power plant construction, mainly how the elements are connected. The elements are stored in array lists. For every element type there is a separate list which makes it easy to access an element. Each object has its own index number which represents the location in the list. This number is not visible to the user, it’s only needed for the data management. A short overview of the containing functions can be seen in Figure 4.3. Green dots represent public functions which can be used by any other functions and red dots stand for private functions (can only be used by other functions in the class). The blue triangular stands for a constructor.
### Constructor

- `PowerPlant()`  
- `PowerPlant(String)`  
- `getPowerPlantName(): String`  
- `setName()`  
- `addRes(): void`  
- `addTur(): void`  
- `addPump(): void`  
- `addPipe(): void`  
- `addElement(i)`: void  
- `getResList(i): Reservoir[]`  
- `getTurList(i): Turbine[]`  
- `getPumpList(i): Pump[]`  
- `getPipeList(i): Pipe[]`  
- `getElementList(i): String[]`  
- `getReservoir(int): Reservoir`  
- `getPump(int): Pump`  
- `getTurbin(int): Turbine`  
- `getPipe(int): Pipe`  
- `getPipeNode(int): PipeNode`  
- `getElement(int): Element`  
- `getElement(String): Element`  
- `getElementID(String): int`  
- `getElementType(String): String`  

### Figure 4.3: Example of the data structure in the power plant class.

In Figure 4.3, this structure is illustrated for a power plant with 3 reservoirs, 2 turbines, 1 pump, 1 pipe node and 6 pipes.
4.2.4 Class PPElement

Class PPElement is a "superclass" containing the properties all the other power plant elements have in common, e.g. name or index number. The other elements like reservoirs, turbine, pumps and pipes inherit the properties from this class and only contain specific characteristics.
4.2.5 Classes Reservoir, Turbine, Pump, Pipe

A power plant consists of reservoirs, turbines, pumps and pipes as shown in Fig. 3.1. Each of these objects has its own counterpart in the program. The classes store and provide access to the variables needed for the particular object. Like explained above, they all inherit the characteristics of PPElement and only specify special properties like e.g. the water content which is only present in the reservoir element.

The special properties for each class are shown below:

<table>
<thead>
<tr>
<th>Reservoir</th>
<th>Turbine/Pump</th>
<th>Pipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water content</td>
<td>max. throughflow</td>
<td>max. throughflow</td>
</tr>
<tr>
<td>per height</td>
<td></td>
<td></td>
</tr>
<tr>
<td>min. sea level</td>
<td>min. throughflow</td>
<td>efficiency</td>
</tr>
<tr>
<td>max. sea level</td>
<td>efficiency</td>
<td>connected elements</td>
</tr>
<tr>
<td>start sea level</td>
<td>height</td>
<td></td>
</tr>
<tr>
<td>inflow</td>
<td>availability</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Special properties for different classes.

4.2.6 Class PipeNode

PipeNode inherits its variables and function from the Pipe class, even if many functions are not needed. This class exists to provide the ability to connect multiple pipes together. The idea is that two or more elements like a pump and a turbine can share a pipe to a reservoir, which makes sense because normally there is only one pipe for such a combination.

4.2.7 GUI Classes

GUI classes are:

- ResEditFrame
- PumpEditFrame
- TurEditFrame
- EditPipeFrame
CHAPTER 4. PROGRAM STRUCTURE

- PipeNodeFrame

They handle the input of data and visualize the already entered data of the power plant objects explained in previous sections.

4.2.8 Other Functions

CSV-reader

A very important function in all of these classes is the csv reader function. It reads a .csv file line per line, isolates values which are separated by the characters "," or ";" and returns an array containing all variables in it. The save and load functions in the mainframe class is also based on this reader, but modified so that it writes values and words or searches key words to know where to place the read values correctly.

Table function

The table in the GUI frames bases on basic java swing tables but there is also implemented a ”tableChanged listener” to detect modifications in a field. This offers the ability to change values directly by editing it in the table instead of being obliged to change every time the csv file.

Charts

To visualize the data, charts are drawn in the GUI frame. For doing this, the class XYChart is used which is explained in more detail in section 4.2.10.

4.2.9 Class DrawPanel

A desired function of the program is to visualize the power plant built by the user. For doing this, the Mainframe class needs to draw the lines between connected elements. This is implemented in this class. DrawPanel uses the Line class which defines a line and it offers the function to connect two points with known coordinates on the visualization panel.
4.2.10 Class XYChart

Drawing plots of data sets is the task of this class. It uses the free available library JFreeChart [9] to draw curves. The input parameters are arrays with two or more columns, in the first column all x-values are stored, in the other columns the corresponding y values (each column is a line in the chart). Additionally it offers also the possibility to enter names in form of strings to name the axes and the plot.

The maximum and minimum x- and y-borders for the data plot are automatically chosen by the function.

4.3 Optimization Class HydroSolver

In this class, the whole optimization process is done. Because this is the most complex class in the program, a more detailed look than on the other classes is taken here.

In order to perform a short-term optimization, the following tasks have to be done:

- Detect all possible flows in the power plant.
- Piecewise linearisation of performance curves.
- Form a correct MIP which can be evaluated by CPLEX [6].
- Actualize all changing values for each timestep.

Those tasks are know discussed in detail.

4.3.1 Detecting Flows

Starting an optimization process in the program begins with the detection of possible flows. Figure [4.5] shows the six possible flows for the power plant structure already used in Fig. [4.3]
The operating principle of the flow detection function is as follows:

Three arrays are initialized, an array for upper elements, an array for lower elements and an array to store the pipe indexes which connects the two elements.

After this, the algorithm checks all pipes and determines the connected elements. Without used pipe nodes, this is a straight process. The height data of both elements are compared and the higher element is stored in the upper array and the lower one in the lower array.

When connected to a pipe node, this procedure is a bit more complex. The problem is when more than one pipe node is between two elements which must also be possible to handle.

The procedure to solve this problem is shown in the Figure 4.6. Each pipe node is replaced by the reservoir connected to it to receive a simple connection without pipe nodes in between. The removed pipe is stored temporarily and is then added to the array containing the connecting pipes.

If a second reservoir would be connected to the right pipe node in Fig. 4.6, the same procedure would be done for each reservoir, resulting in a total of 8 possible flows instead of the 4 in the example below.
After detecting all flows, the stored flows for the power plant in Figure 4.5 can be seen in Table 4.2:

<table>
<thead>
<tr>
<th>Upper element:</th>
<th>R0</th>
<th>R0</th>
<th>R2</th>
<th>T0</th>
<th>P0</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower element:</td>
<td>T0</td>
<td>P0</td>
<td>T1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
</tr>
<tr>
<td>Connecting pipes:</td>
<td>0:1</td>
<td>0:2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 4.2: Stored flows for power plant in Figure 4.5

### 4.3.2 Performance Curve

The performance curve shows how much power a turbine produces or a pump consumes per water throughflow. It is calculated similar as equations 3.2 and 3.4 without the height term. So we obtain a function in \( \frac{MW}{h \cdot m^3/s} \):
\[
\frac{P_{\text{turbine}}}{h} = q_t \cdot g [9.81 \frac{m}{s^2}] \cdot \eta_{\text{pipe}} [\%] \cdot \eta_t \left[ \frac{\%}{m^3/s} \right] \quad [W] \quad (4.1)
\]

\[
\frac{P_{\text{pump}}}{h} = q_p \cdot g [9.81 \frac{m}{s^2}] \cdot \frac{1}{\eta_{\text{pipe}} [\%]} \cdot \frac{1}{\eta_p} \left[ \% \right] \left[ m^3/s \right] \quad [W] \quad (4.2)
\]

The maximum through-flow is defined by the connecting pipes. Its value corresponds to the smallest maximum through-flow of all connecting pipes between the reservoir and the element.

To assure that no division by zero is done in the function \(4.2\), the minimal inserted value in the equation is 0.05 (\(\hat{\eta}_p = 5\%\)) efficiency for \(\eta_p\) (\(\eta_{\text{pipe}}\) is never 0, otherwise, the pump is not able to operate).

### 4.3.3 Piecewise Linearization

Piecewise linearization was explained in section [3.1]. In the HydroSolver class, the piecewise function calculates 3 linearized pieces of the given performance curve and stores the values into arrays in the object (e.g. turbine).

There is an array for the "a"-values and one for the "b"-values to calculate later \(y = a + b \cdot x\).

<table>
<thead>
<tr>
<th>Array a:</th>
<th>(a_0)</th>
<th>(a_1)</th>
<th>(a_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array b:</td>
<td>(b_0)</td>
<td>(b_1)</td>
<td>(b_2)</td>
</tr>
</tbody>
</table>

Table 4.3: Array to store linearization values

\[
y_0 = a_0 + b_0 \cdot x_0 \quad 0 \leq x_0 \leq \frac{1}{3}x_{\text{max}} \quad (4.3)
\]

\[
y_1 = a_1 + b_1 \cdot x_1 \quad \frac{1}{3}x_{\text{max}} \leq x_1 \leq \frac{2}{3}x_{\text{max}} \quad (4.4)
\]

\[
y_2 = a_2 + b_2 \cdot x_2 \quad \frac{2}{3}x_{\text{max}} \leq x_2 \leq x_{\text{max}} \quad (4.5)
\]

Equation (4.3) defines the linearization of the first third of the curve, equation (4.4) defines the second third and equation (4.5) defines the third piece.
4.3.4 Spill Cost

The spill cost defined in the code is 10% higher than respective watervalue. The penalty of 10% is arbitrarily chosen.

\[ c_{1,r} = 1.1 \cdot \gamma_r \]  \hspace{1cm} (4.6)

4.3.5 Generate MIP

In the mathematical description (Section 3.1) the optimization function and its constraints were explained. Until this point, the following data is known to the optimizer:

- All flows with upper- and lower element and the connecting pipes
- Piecewise linearization data in the objects

Our variables we want to know are:

- Outflow \( u_r \) and Spilling water \( s_r \) for each reservoir
- Throughflow \( q_t \) for each turbine
- Throughflow \( q_p \) for each pump

Define Variable Multiplicators

Turbines and pumps:

Each turbine and each pump have three through-flow variables (for each piece in the linearized function) and therefore three different values to multiply with them. The values are calculated by the height difference and the values from the linearization:

\[ \xi_{t,i,l} = (h_{\text{uperelement}} - h_{t,i}) \cdot b_{t,i,l} \]  \hspace{1cm} (4.7)
\[ \xi_{p,i,l} = (h_{\text{uperelement}} - h_{\text{lowerelement}}) \cdot b_{p,i,l} \]  \hspace{1cm} (4.8)

\[ l = 0, 1, 2 \]

The value ”a” from the linearization is only added if the corresponding block of the turbine or pump is running and is not automatically zero when the
throughflow is zero (like in (4.7)) is therefore multiplicated with the binary variable.

\[ \zeta_{t,i,l} = (h_{\text{uperelement}} - h_{t,i}) \cdot a_{t,i,l} \]  
\[ \zeta_{p,i,l} = (h_{\text{uperelement}} - h_{\text{lowerelement}}) \cdot a_{p,i,l} \]  
\[ l = 0, 1, 2 \]

**Reservoirs:**
For reservoirs, there is the watervalue and the costs for spilling water which have to be multiplied with the variables. The watervalue is calculated out of the water content which itself is at the beginning given by from the starting sea level. To do this, there is a simple function which searches the corresponding x value when a y value is given and vice versa.

**Definition of Bounds**

**Turbines and pumps:**
For turbines and pumps, there is a upper and a lower bound for each piece of the piecewise linearized function (see equations (4.3) - (4.5)). When a block of the element doesn’t run the flow must be able to be zero. Therefore, the lower bounds are initialized as zero while the minimum value for the through-flow when the block is running is set by constraints. The upper bounds are defined by the upper limits of the blocks received from the piecewise linearization.

**Reservoirs:**
For the reservoirs, there is no outflow constraint since bound, because the maximum outflow is already defined by the turbine bound. But the cplex function initializes the lower bound at 0 if no input is given. However a \(-\infty\) is wanted (to allow pumping = negative outflow), the lower bound for reservoir is initialized at -Double.MAX_VALUE (largest negative value possible for type double).

**Definition of Constraints**

**Turbines and pumps:**
For each of the three linearized parts, there is a binary variable initialized to
ensure that only one block runs. This is easily achieved by varying bounds (shown here for a turbine $i$):

\[ q_{t,i,0} \geq 0 \]  
\[ q_{t,i,1} \geq \frac{1}{3} x_{\text{max}} \cdot t_{i,1} \]  
\[ q_{t,i,2} \geq \frac{2}{3} x_{\text{max}} \cdot t_{i,2} \]

The produced power is calculated with:

\[ P_{t,i} = \sum_{l=0}^{2} (\xi_{t,i,l} \cdot q_{t,i,l} + \zeta_{t,i,l} \cdot t_{i}) \]  
\[ P_{p,i} = \sum_{l=0}^{2} (\xi_{p,i,l} \cdot q_{p,i,l} + \zeta_{p,i,l} \cdot p_{i}) \]

Another parameter which has to be considered is the availability of turbines and pumps.
The functionality is achieved by setting the throughflow to zero if the turbine is not available and allow it to be greater than zero if the turbine or pump is available

\[ q_{t(p),i} = \begin{cases} 0 & \text{if availability at time } t = 0 \\ 0 & \text{if availability at time } t = 1 \end{cases} \]

Reservoirs:
For reservoirs, the constraints are much more complex to calculate. There are multiple constraints:

1. The outflow is equal to the turbined water from the connected turbines minus the pumped water from the connected pumps minus the inflow.

2. Usage of water less than the actual watercontent plus the inflow into the reservoir

3. If more water is pumped into the reservoir than it is able to store, spilling water is $>0$. 
For the first constraint the connected pumps and turbines have to be known. Therefore, an matrix CE (Connected Elements) which stores the connections is generated, with (1) for every connected element which consumes water and a (-1) for every element which adds water, otherwise zero. For the example power plant used in Figure 4.5 the matrix looks as follows:

<table>
<thead>
<tr>
<th></th>
<th>Turbine0</th>
<th>Turbine1</th>
<th>Pump0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservoir0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>Reservoir1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Reservoir2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.4: Matrix CE which stores the connected elements

With this data it is possible to calculate the water consumed by the connected elements \( u_r \) in \([\frac{m^3}{h}]\), throughflows in \([\frac{m^3}{s}]\), inflows in \([\frac{1000m^3}{h}]\):

\[
u_r = 3600 \cdot \left( \sum_{n=0}^{T} CE(r, n) \cdot \sum_{l=0}^{2} q_{t,n,l} + \sum_{n=T}^{T+P} CE(r, n) \cdot \sum_{l=0}^{2} q_{p,n,l} \right) - i_r \cdot 1000 \quad (4.17)
\]

Limiting the consumed water to the actual water content plus inflows is done as following.

\[
u_r \leq w_r \cdot 1000 \quad (4.18)
\]

The inflow is already contained in the outflow variable. \( w_r \) is multiplied by 1000 because the outflow is in \([\frac{m^3}{h}]\) and the water content in \(1000m^3\).

To fulfill the 3rd constraint,

\[
s_r = \begin{cases} 
0 & \text{if } -u_r \leq (w_{r,max} - w_r) \cdot 1000 \\
-u_r - (w_{r,max} - w_r) \cdot 1000 & \text{else} 
\end{cases} \quad (4.19)
\]

is introduced.

If more water than the reservoir capacity \( (w_{r,max} - w_r) \) is pumped in (negative \( u_r \)), there is spilling water, otherwise zero spilling occurs.
Optimization Function

At this point, all needed values except the cost for the power difference $c_2$ and the power demand DP are known. This two values are requested to be entered by the user before starting the optimization. Finally it’s possible to create the equation (3.5).

4.3.6 Timestep Handling

For each timestep, the optimization problem is solved. To do this, the changed variables needs to be updated in every iteration, which is done by the timestep handling function. The procedure is as follows:

1. Update watercontent:
   \[ w^{t+1}_r = w^t_r - u^t_r / 1000 \quad (4.20) \]

2. Update watervalue:
   \[ \gamma^{t+1}_r = \text{value\_curve}(w^{t+1}_r) \quad (4.21) \]

3. Update sealevel:
   \[ h^{t+1}_r = \text{sealevel\_curve}(w^{t+1}_r) \quad (4.22) \]

The number of timesteps is defined by the number of x values in the power difference cost array, entered by the user.

4.3.7 Proper Use Of HydroSolver Class

Some notes about the program code:

The HydroSolver class has to be started by generating a new HydroSolver object and give the power plant object to optimize to it. Automatically the detection of possible flows and the piecewise linearization of all turbine and pump performance curves are provoked. Afterwards, the optimization process is started by calling the function ”calculateOptimum(pp, powerdemand)” (parameters: pp = powerplant, powerdemand = array with power demand data).
This function returns all results in an arraylist containing one array for each pump and each turbine, three for each reservoir and one which includes the power difference data. An example how this looks like is given below for the power plant used in Figure 4.5.

<table>
<thead>
<tr>
<th>Stored</th>
<th>hour 0</th>
<th>hour 1</th>
<th>...</th>
<th>hour n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughflow Turbine0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throughflow Turbine1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throughflow Pump0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Watercontent R0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Watercontent R1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Watercontent R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spill R0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spill R1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spill R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sealevel R0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sealevel R1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sealevel R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Powerdifference</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5: Return list resulting from power plant used in Figure 4.5.

### 4.4 Class OptimizationFrame

This class displays the data results and offers input fields for the power demand data and the power difference costs.

After starting the optimization process like described in the section before, the results are plotted and stored in a csv-file. This file is stored in the same directory as the csv-file of the power demand is located.

To prevent an override of previous stored results, the file name contains the actual date and time in its name.
Chapter 5

Conclusion and Outlook

5.1 Conclusion

An already existing algorithm for optimizing the use of hydro power plants considering short time periods which is implemented in a matlab environment has been implemented in a java program. It turned out that java was a good decision because the implementation of the cplex function was straightforward. In addition, the speed turned out to be very fast (less than a second for the power plant structure in Fig. 4.5 and 24 timesteps), but it has not yet been tested with very big system configurations.

Thanks to the modularity the software works for every possible power plant and offers therefore much more flexibility than its matlab counterpart. Because of the modularity of the code, the program offers a very good base for future work.

5.2 Outlook

Like already mentioned in the conclusion, the modularity of the program gives a possibility to easily implement new functions or to extend existing functions.
A few ideas for further work are listed below:
Minimum Pump Up Time

A pump needs a couple of minutes to start because the water pressure has to be established to bring the water flow orientation in upwards direction. Therefore many power plant operators want to run their pumps only if they are used for more than a minimum time, e.g. one hour to minimize the start-up to operating ratio.

The option to set such a minimum pump up time could be implemented.

Secondary Control Option

The system operator buys the option to activate energy from power plant operators to cover outages. If an operator decides to offer this energy, he has to be able to deliver the demanded energy within 15 minutes (secondary control [8]) and is therefore not able to produce at full power. In addition, at minimum one turbine which runs at the negotiated power must be active all the time the secondary control energy is offered to be able to provide negative energy.

An option to chose if the power plant offers secondary control and if yes, how much power is offered could be a further future work. The described additional constraints must then be fulfilled if the option is activated.

Inflow and Spot Price Predictions

The inflows and spot prices could be estimated based on statistical data and actual weather forecasts.
Bibliography


Appendix A

Results For A Typical Powerplant

The power plant used in Figure 4.5 is optimized in this section to show how the program and the results look like.

A.0.1 Power Plant Data

Figure A.1 shows the start situation.

Figure A.1: Power plant to optimize.
The data for Reservoir0 and Reservoir2 can be seen in Figure A.2 and A.3.

Figure A.2: Data for Reservoir0.
APPENDIX A. RESULTS FOR A TYPICAL POWERPLANT

Figure A.3: Data for Reservoir2.

The start heights are:

<table>
<thead>
<tr>
<th>Reservoir</th>
<th>Start height [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reservoir0</td>
<td>1000</td>
</tr>
<tr>
<td>Reservoir2</td>
<td>800</td>
</tr>
</tbody>
</table>

Table A.1: Starting heights for reservoirs

Reservoir1 is modeled as a very big Reservoir with no inflows and almost zero water value.

All turbines and pumps have the same efficiency curve:
APPENDIX A. RESULTS FOR A TYPICAL POWERPLANT

Figure A.4: Efficiency curve for turbines and pump.

<table>
<thead>
<tr>
<th></th>
<th>minimum troughflow [m^3/s]</th>
<th>maximum troughflow [m^3/s]</th>
<th>height [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turbine0</td>
<td>2.0</td>
<td>25.0</td>
<td>400</td>
</tr>
<tr>
<td>Turbine1</td>
<td>2.0</td>
<td>25.0</td>
<td>180</td>
</tr>
<tr>
<td>Pump0</td>
<td>1.0</td>
<td>25.0</td>
<td>400</td>
</tr>
</tbody>
</table>

Table A.2: Minimum and maximum throughput and heights for turbines and pump

The availability is set always to 1 for all elements and all pipes have large maximum throughflows and efficiency=1 so that they not influence the result.

Now the optimization frame is opened by clicking on "Optimierung->Start" and the power demand and power cost data is requested. The used data is shown in Figure A.5.
When all required data to start the optimization process is available and the process is started by clicking the "Start Optimierung" button, the results shown in Figure A.6 are displayed.
A.0.2 Results

As can be seen in Fig. A.6, in almost all hours the power is produced by Turbine1. This makes sense, because the height difference between Reservoir2 and Turbine1 is higher than between Turbine0 and Reservoir0 and both have the same efficiency curve. In addition, Reservoir2 has a lower water value at the beginning.

In the hours 8 to 17 Turbine0 is also working with $2 \text{ m}^3/\text{s}$. The reason is that Turbine1 can not produce the demanded power alone and it is cheaper to produce also with Turbine0 instead of buying power on the market.

In the end, Turbine0 is also active because the watervalue in reservoir2 is now at a higher value than at beginning and it is now cost effective to produce some power with water from Reservoir0.

The spilling water for Reservoir0 is due to the inflows. Because no negative power has to be produced, the pump stays at zero.

If the price for difference in power production is very low, it can be cost effective to buy all the energy on the market than to produce it with the turbines. Figure A.7 shows the result when the cost for power difference is set always to 1 EUR/MWh.

The black curve corresponds to the power difference.
Figure A.6: Optimization results.
Figure A.7: Optimization results for cost in power difference = 1 EUR/MW.