NTNU – Trondheim
Norwegian University of
Science and Technology

Optimization of Energy Storage in Buildings Based on Self-optimizing Control

Vegard Skogstad

Chemical Engineering and Biotechnology
Submission date: June 2014
Supervisor: Sigurd Skogestad, IKP
Co-supervisor: Vinicius de Oliveira, IKP

Norwegian University of Science and Technology
Department of Chemical Engineering

Summary

A steadily increasing fraction of Europe's electricity is generated by renewable and less reliable energy sources. It is therefore necessary to find smart ways to store excess energy until it is demanded. This could be achieved by using the energy storage potential in the hot water tank. The heating of hot water tanks can be improved by using a cost minimization strategy to make hot water tanks heat when electricity price is low and conserve energy when price is high. This would both provide economic benefits to the owner of the tank and, provided widespread use, a more stable energy market.

This thesis considers the optimal operation of energy storage in buildings with focus on the hot water tank. The objective has been to minimize operational cost while still meeting the hot water demands of the end user. To achieve this, a hot water tank system has been modeled using Simulink and Matlab and a feedback control structure implemented to stabilize the system. By using ideas from self-optimizing control the cost function has been simplified to a form that makes it solvable even with limited computation resources. The simplified problem has then been solved using an MPC-solver. The resulting optimized case (Case II) has been compared with a simple policy of heating for a set amount of hours at night (Case III) and holding a constant temperature in the tank (Case I).

Based on the findings in this thesis, using optimization to find optimal energy levels in the tank does not give any significant benefit over using a simple policy of heating the tank at night. Though both cases are economically better than using a constant temperature set point, the difference is not huge. Compared to holding a constant temperature in the tank, Case II and Case III gave savings of 8.61% and 9.12% respectively. Actual implementation of the system could still be beneficial in areas with more pronounced price variation. Further work should be focused on improving the solver and verifying the closeness to optimality of the simplifications that have been made.

Sammendrag

En stadig økende andel av Europas elektrisitet generes av fornybare energikilder som vind- og solkraft. Disse energikildene varier med værforhold og sesong i mye større grad enn fossile energikilder. Det er derfor nødvendig å finne smarte måter å lagre overskuddsenergi på. Dette kan oppnås ved å bruke lagringspotensialet i varmtvannstanken. Oppvarming av varmtvannstanker kan effektiviseres ved å minimere tankens driftskostnad. Da kan man oppnå at varmtvannstankene varmes når strømprisen er lav og sparer energi når prisen er høy. Dette ville gi både direkte økonomiske fordeler til eieren av tanken og, forutsatt utbredt bruk, stabilisering av energimarkedet.
Denne avhandlingen vurderer optimal drift av energilagring i bygninger med fokus på varmtvannstanken. Målet har vært å redusere driftskostnadene og samtidig møte sluttbrukerens forventninger. For å oppnå dette, har varmtvannstank-systemet blitt modellert i Simulink og Matlab. Det har også blitt implementert feedback-regulering for å stabilisere systemet. Ved å bruke ideer fra selv-optimalisere kontroll har kostnadsfunksjonen blitt forenklet til en form som gjør det mulig å løse minimeringsproblemet selv med begrenset beregningskapasitet. Det forenklede problemet har deretter blitt løst ved bruk av MPC. Løsningen (Case II) har blitt sammenliknet med en policy der tanken varmes i et fast tidsintervall hver natt (Case III) og med å holde konstant temperatur i tanken (Case I).
Funnene i denne avhandlingen viser at å finne det optimale energinivået i tanken ikke gir noen vesentlig forbedring sammenliknet med å varme tanken i et fast tidsintervall hver natt. Selv om begge tilfeller gir lavere driftskostnad enn å bruke konstant tanktemperatur, er forskjellen liten. Sammenlignet med å holde konstant tanktemperatur gir Case II og Case III besparelser på henholdsvis $8,61\%$ og $9,12\%$. Å implementere systemet i praksis kan fortsatt være gunstig i områder med større prisvariasjon. Videre arbeid bør fokusere på å forbedre løseren og verifisere at forenklingene som er gjort faktisk gir nesten-optimale resultater.

Preface

This thesis was written as the final part of my M.Sc. in Chemical Engineering at the Norwegian University of Science and Technology.

I would like to thank my supervisor, Professor Sigurd Skogestad for allowing me to work on this project and for his patience when my results were not forthcoming. I would also like to thank my co-supervisor Vinicius de Oliveira for supporting me during my work with this thesis. When my simulation did not yield reasonable results, he would always have a suggestion that pushed me in the right direction. This thesis would have been impossible without the work of Emma Johansson. Reading about her work on the same system gave me great insight into the problem, and using her controller tunings saved me a lot of time and work.

Finally, I would like to thank my fellow students at Chemical Engineering and Biotechnology for making these five years an unforgettable time!

Declaration of Compliance

I declare that this is an independent work according to the exam regulations of the Norwegian University of Science and Technology (NTNU).

Vegard Skogstad
Trondheim, June 23, 2014

Contents

List of Tables

List of Figures

Nomenclature

Symbol	Explanation	Unit
$\mathbf{A}$	State matrix	various
$\mathbf{B}$	Input matrix	various
c_p	Specific heat capacity	kJ/kg,℃
$\mathbf{C}$	Disturbance matrix	various
d	Disturbance	various
$\mathbf{D}$	Output matrix	various
E	Energy in the hot water tank	kJ
G	Transfer function	various
H	Enthalpy	kJ
J	Cost function	€
K_c	Controller gain	various
n	number of time steps in prediction horizon	-
N	Length of prediction horizon	sec
p	Electricity price	€/MWh
P	Pressure	Pascal
Q	Heater duty	kW
q_{cw}	Flow of cold water for mixing	l/min
Q_d	Energy demand	kW
q_{hw}	Flow of hot water to end user	l/min
q_{in}	Flow of cold water into the tank	l/min
Q_{loss}	Heat loss to surroundings	kW/h
q_{out}	Flow of water out of the tank	l/min
t	Time	sec
T	Tank temperature	℃
T_{cw}	Cold water temperature	℃
t_{end}	Simulation end point	sec
$T_{hw,s}$	Hot water temperature set point	℃
T_{hw}	Hot water temperature	℃
T_s	Temperature set point	℃
t_v	Time that $T \leq T_{hw,s}$	hours
u	Input	various
U	Internal energy	kJ
U_k	Kinetic energy	kJ
U_p	Potential anergy	kJ
V	Volume	l
V	Volume set point	l
W_b	Work from pressure difference	kJ
W_f	Work from expansion of the control volume	kJ
W_s	Shaft work	kJ
x	State	various

Symbol	Explanation	Unit
y	Output	various
y_e	Error to controller	various
y_m	Measured ouput	various
y_s	Output set point	various
Greek sybols	**Explanation**	**Unit**
Δt	Size of time step	-
ρ	Density	kg/l
τ_i	Integral time	sec

Abbreviation	Explanation
BRIC	Brazil, Russia, India & China
IMC	Internal model control
IPCC	Intergovernmental Panel on Climate Change
PID	Proportional-integral-derivative
SIMC	Simple IMC
O.S.	Optimal solution
NVE	Norwegian Water Resources & Energy Directorate

Chapter 1

Introduction

Global energy demand is ever increasing, most recently thanks to growth in emerging markets and BRIC-countries. At the same time there is global agreement that CO_2 emissions have to be decreased. IPCC (Edenhofer et al., 2014) predicts that global CO_2 emissions from the energy supply sector alone will at least double by 2050. To avoid these predictions there needs to be a massive shift from oil based energy to renewable resources. One of IPCC's key proposals is to decarbonize electricity generation. As of today 30 % of all electricity comes from nuclear and renewable resources while the rest comes from carbon based sources. To increase this to the targeted 80 % by 2050, large investments has to be made into renewable resources like solar and wind power. There are several additional challenges with these energy sources, the most important being their unpredictability. Neither windmills nor solar panels are able to produce electricity with the consistency that a coal factory or natural gas plant can. With an increased dependency on renewable resources the energy market will become more volatile with weather trends greatly impacting the total amount of electricity being produced. To facilitate investments in renewable energy sources, solar and wind power have been heavily subsidized by European governments, guaranteeing power companies a certain price regardless of demand. In France electricity generated from wind turbines sells for 83 €/MWh, demand or not, while nuclear energy sells for approximately 40 €/MWh (Lerouge and Atlan, 2013). This is not a sustainable long term solution, so it is necessary to find ways to store excess energy until it is demanded.
One way to store energy that is available in almost every household, is in the domestic hot water tank. Hot water tanks are essentially like large batteries, storing energy in the form of hot water. If the heating of hot water tanks could happen when the supply of electricity is high, and thus the price low, this would stabilize the price and renewable energy sources would become more competitive. In this thesis it is proposed to achieve this by using smart control systems and an economical optimization for the hot water tank. The tank should use energy when the prices are low and conserve energy while prices are high, all while still providing

hot water to the end user. The novelty of this approach is that it should provide direct economical benefits to the owner of the hot water tank, making distribution of the system a lot easier. It would also provide an alternative way of stabilizing a more volatile energy market; instead of focusing on shifting the supply of energy to meet the demand, the demand is shifted to meet the supply.
Another advantage of smart control is that demand would decrease during the hours that total energy usage is high, known as peak hours. Considering that increasing peak hour demand is what makes new investments in electricity grid infrastructure necessary, decreasing peak demand could help postpone investments in expensive infrastructure. This is important as total energy capacity in Norway has been largely unchanged since 1989 (Doman, 2013). Several projects are planned in the coming years that would severely strain the electricity grid, perhaps the biggest among them being the electrification of Utsirahøyden (Olje- og energidepartementet, 2014). Official estimates indicate that investments into electricity infrastructure alone will account for more than 1300 MNOK in 2013 (Olje- og energidepartementet, 2012), so the economical potential of postponing such investments is huge.

1.1 Previous work

Several other projects have looked into using hot water tanks to decrease the total energy demand during peak hours. Integral Energy have been running a successful program for domestic hot water control since 1995 (Charles River Associates, 2003). In the program customers give control of their hot water heater to Integral in return for cheaper electricity. The company uses their controlled hot water heaters to decrease energy demand during peak hours, by heating at time slots when energy is cheap. The general heating policy used by Integral was to heat during the night and also add a small heating period during the day. Limiting demand during peak hours have also been investigated by Ericson (2009), who tested the effects of disconnecting domestic hot water heaters during peak hours using a statistical model. Ericson found that this would cause a spike of energy usage once the heaters were reconnected to the electricity grid. This "payback effect" might lead to simply shifting the total peak demand to a different time slot instead of decreasing it, provided enough hot water heaters are disconnected at the same time. Cycling the disconnection times was suggested to avoid this. Other positive effects from disconnection was that the total energy consumption actually decreased marginally.
This thesis is a direct continuation of the model proposed by de Oliveira et al. (2013) and the results of Johansson (2013). Both have looked at the hot water heating system from an economical optimization perspective. Johansson simulated a hot water tank system and tested out several different combinations of heating policies and control structures. The goal was to compare different cases and find a simple structure that was close to optimal. The simulations were carried out for one day at a time with promising results. It must be noted that there was put no constraints on the temperature in the tank at the end of each day. The proposed optimal solution therefore had a full tank and maximum temperature at the start

of each day and an almost empty tank with the minimum allowed temperature at the end. For all but one of the cases, reheating the tank for the next day of simulation was not taken into consideration. Johansson found that the cost could be minimized by keeping the tank at maximum volume and changing the temperature depending on price. For this case an additional cost factor was added to account for reheating the tank. This gave savings in the range of 33.6-35.8 % compared to the worst case (holding a constant maximum temperature and volume in the tank). The mentioned case is however not practically implementable. This is because the optimal temperature set points were found from running multiple simulations of the same case with different set points and the same demand, then picking the best one. In reality, future demand is not predictable, so being able to find optimal set points for a specific demand profile in this way is not realistic. The controller tunings found by Johansson was reused in this thesis, these are shown in Section 3.4.

1.2 Thesis scope

This thesis considers the optimal operation of energy storage in buildings with focus on the hot water tank, and is a continuation of the work of Johansson (2013). The objective is to minimize the cost of operation while still meeting the hot water demands of the end user. This was to be achieved using smart simplifications that makes the optimization problem solvable even with limited computation resources. The objective has been solved by modeling a hot water tank system in Simulink and Matlab and implementing a feedback control structure to stabilize the system. By using ideas from self-optimizing control the minimization problem have been simplified to be possible to solve even with limited computation resources. The resulting optimized case (Case II) has been compared with a simple policy of heating for a set amount of hours at night (Case III) and always heating as much as possible (Case I). The effects of chosen back-off and tank size has also been investigated.

1.3 Thesis structure

Theory and introduction of the concepts used in the rest of the thesis is shown in Chapter 2, then follows a description of the system and model development in Chapter 3. The different cases that have been simulated are described in Chapter 4 while Chapter 5 explains how these cases were implemented into Matlab and Simulink. The results are presented in Chapter 6 and discussed in Chapter 7. At the end, conclusions and recommendations for further work are presented in Chapter 8.

Chapter 2

Theory

2.1 Control theory

Control theory describes the behavior of dynamic systems and how to control them. As an interdisciplinary field the terminology and use of symbols tends to vary with the background of the author, so understanding the meaning behind each abbreviation and symbol can be a daunting task for the uninitiated. For this reason the amount of theory in this section is kept as small as possible. For a more thorough look into control theory, see Seborg et al. (2003) or Foss et al. (2003).

2.1.1 Modeling a dynamic system

To understand control theory a general understanding of state space representation is required. A state space representation is a mathematical model of an actual system. Here the behavior of the actual system is recreated using first order differential equations and state variables. Any variable that has been determined to significantly affect the system, and thus included in the model, is a state variable. There are four types of state variables: System inputs, $\mathbf{u}$, states, $\mathbf{x}$, outputs, $\mathbf{y}$, and disturbances, $\mathbf{d}$. All system described by a linear continuous time system, can be reduced to the state space representations shown in Eq. 2.1 (Ruscio, 2009).

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{C}\mathbf{d} \tag{2.1}$$

$$\mathbf{y} = \mathbf{D}\mathbf{x} \tag{2.2}$$

Where $\dot{\mathbf{x}}$ is a vector containing the time derivatives of each state and $\mathbf{A}$ and $\mathbf{B}$ are matrices whose composition are determined by the represented differential equation. $\mathbf{C}$ and $\mathbf{D}$ are selection matrices which describes how the input and states affect the output. If the initial conditions are known the new state can be calculated using

$$\mathbf{x_{i+1}} = \mathbf{x_i} + \dot{\mathbf{x}}_\mathbf{i}\Delta t \tag{2.3}$$

From Eq. 2.3 it follows that the outputs can be calculated at any point if:

- The matrices **A**, **B**, **C** and **D** are known.
- The disturbances, inputs and initial conditions are available.

This allows for new system states, to be calculated as long as the input is known, which leads to the general representation of a dynamic system shown in Fig. 2.1. In the figure it has been assumed that all states are measured directly ($\mathbf{C} = \mathbf{I}$). This gives $\mathbf{x} = \mathbf{y}$.

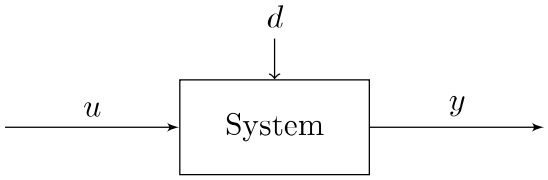

Figure 2.1: A dynamic system with inputs u, disturbances d and outputs y.

2.1.2 Controlling a dynamic system

Eq. 2.1 implies that changes in the state **y** can be controlled by manipulating the input **u**. It does however not take into account that disturbances also will change the state. Which tends to destabilize the system if the change in inputs does not counteract the change in disturbances. This is where a controller comes into play. By manipulating the inputs in a smart way, a controller counteracts disturbances and stabilizes the system.
Control theory distinguishes between two different control arrangements, closed-loop (feedback) and open-loop (feed-forward) control. Open-loop control measures the input and from this makes a control decision. Closed-loop control instead measures the output from the system and makes adjustment depending on how much it differs from the wanted output. In this thesis closed-loop control is used. A graphical representation of closed-loop control is shown in Fig. 2.2. Here y_e is the

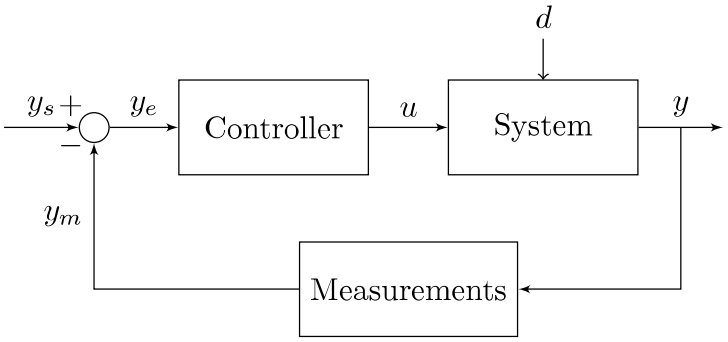

Figure 2.2: Closed-loop control of a dynamic system.

difference between the measured output y_m, and its set point y_s. Based on the error y_e that enters the Controller, the input u to the system changes. This is how

the controller tries to keep the output at it's set point($y = y_s$). If the output is exactly equal to its set point the input from the controller would be left unchanged, for all other states the controller would change the input depending on the size and sign of the error. The difference between y_m and y is just measurement error, for computer models the measurement error is 0.

2.1.3 PID controllers

PID controller is short for proportional-integral-derivative controller. The name corresponds to the different parts of the controller, where each attempts to counteract different errors. The proportional part corrects present error, the integral part corrects past error, while the derivative part corrects for future errors. Some controllers do without one or several of these corrective parts, the most common is the PI controller, which does not correct for future errors. The controllers used in this thesis are PI controllers, a PI controller works as shown in Eq. 2.4.

$$u(t) = K_c \left(y_e + \frac{1}{\tau_i} \int_0^t y_e(\tau) \mathrm{d}\tau \right) \tag{2.4}$$

In control theory the Laplace transformation (Foss et al., 2003) of the above equation is commonly used instead,

$$G = K_c \left(1 + \frac{1}{\tau_i s} \right). \tag{2.5}$$

The PI controller depends on two parameters, K_c and τ_i. The controllers can be tuned by manipulating these parameters. Typical methods of obtaining good K_c and τ_i values are: From an open loop step response, from closed loop using P-control (Shamsuzzoha, 2013) or from obtaining first or second order models using the half-rule and then applying SIMC tuning rules (Skogestad, 2003). For this thesis, controller tunings found by Johansson (2013) were used.

2.1.4 Optimal control

The previous sections have dealt with disturbance rejection. In optimal control, the goal is not only to limit the disturbance, but also to ensure effective input usage. Typical objectives of optimal control would be to minimize costs while also ensuring smooth operation. An example would be minimization of the cost J such as

$$\min_u J(\mathbf{u}, \mathbf{d}), \tag{2.6}$$

while keeping within some constraints.The inputs $\mathbf{u}$ are typically manipulated indirectly by adjusting the set point of feedback and feed-forward controllers. By choosing clever set points the control goals can be achieved while also ensuring optimality. This is the basis of model predictive control(MPC) and the optimized case described in Section 4.2.

Model predictive control

MPC is a closed loop procedure based on repeatedly solving an optimization problem. Solving control problems using an optimization approach gives a set of optimal inputs. Finding optimal inputs for a period into the future is not new, and has been used well before MPC. Often these solutions tended to be poor for the later inputs in the set, as disturbances or inaccuracies in the model had built up. The state of the real system will at this point be radically different from what it was modeled to be, and proposed optimal solutions for the modeled system is no longer optimal in reality. This is where MPC excels. The defining feature of MPC is that it is closed loop, which implies that it receives continuous feedback from the system. The MPC uses this feedback to calculate a new set of optimal inputs for the system. It will therefore solve an optimization problem repeatedly, with only the first step of it's proposed solutions ever getting implemented. A basic MPC procedure as per Mayne et al. (2000) is shown in Algorithm 1. Intuitively finding a

Algorithm 1 MPC procedure.

1: **for** $t = 0, 1, 2, \ldots$ **do**
2: Find the current state x_t.
3: Solve a dynamic optimization problem on the prediction horizon from t to $t + N$ with x_t as the initial condition.
4: Apply the first control move u_t from the solution above.
5: **end for**

set of inputs far into the future and then just applying the first one might seem like a waste of computing power, but in practice it is very powerful. Finding optimal future inputs can be likened to giving the solver "vision". Even if the vision isn't perfect (due to inaccuracies in the model) it makes the solver take into account future disturbances and not just what happens at that instant. The beauty of MPC is that even though predictions towards the end of the horizon are not entirely correct, the only implemented step is very often close to optimal.
The biggest differences among MPC solvers are the size and the discretization of the prediction horizon. The prediction horizon is the set of time sections that the optimizer finds solutions for. The simplest MPC solvers have uniformly sized time sections, while more advanced MPC solvers tend to use small time sections at the beginning of the prediction horizon and then gradually larger sections towards the end. The motivation for this is to improve prediction that is actually going to be used (the first one) at the cost of some loss of detail where the model is least accurate. MPC solvers can have either a moving or shrinking horizon. A moving horizon has a constant length, the horizon is called moving as it moves along with the state, always a certain number of steps ahead of the current state. For continuous systems a moving horizon is typically preferred. A shrinking horizon has a constant end point. It is said to be shrinking as the size of the horizon shrinks as time passes. Shrinking horizon is optimal for systems that should run for a specific period before stopping.

Self-optimizing control

Optimal control can be hard to achieve, but understanding the general concept should not be difficult. Optimal control can be defined as

$$\min_u J(\mathbf{u}, \mathbf{d}) = \min_u J(\mathbf{u}_{opt}(\mathbf{d}), \mathbf{d}) = J_{opt}(\mathbf{d}). \tag{2.7}$$

Simply put, an optimal solution has the inputs that minimizes J for the given disturbances. These inputs are defined as optimal inputs $\mathbf{u_{opt}}(\mathbf{d})$, and the resulting cost function is defined as the optimal cost $J_{opt}(\mathbf{d})$. Finding the optimal inputs is far from trivial for most cases. Systems with hard to predict disturbances are particularly good candidates for self-optimizing control. Self-optimizing control is simply finding system variables that gives feasible, near optimal operation from being kept at a constant value. Good self-optimizing control is to a large extent about engineering knowhow and can be achieved by:

- Using inputs to control the active constraints (Mayne et al., 2000) at optimal operation.
- Remaining input should be used to keep a self-optimizing variable c constant. c should be relatively insensitive to disturbances.

Following these rules should give close to optimal operation. For complex systems with several possible self-optimizing variables, Skogestad (2004) suggests that different alternatives can be compared using the loss function

$$L(\mathbf{u}, \mathbf{d}) = J(\mathbf{u}, \mathbf{d}) - J_{opt}(\mathbf{d}). \tag{2.8}$$

Which gives an indication of how close to optimal each solution is.
One of the challenges for self-optimized control is that the set of active constraints might change with the disturbances. We call this entering a new "active constraint region" (Jacobsen, 2011). If the active constraints region changes, new variables would have to be controlled, and the best choice of self-optimizing variable will change as well. If this happens a new control structure or at least new set points should be used.

2.2 Pareto Optimization

For complex problems the goal is often to achieve several things at once. Defining if a solution is optimal can be difficult for such problems, as there might not exist one solution that gives optimal results for both objectives. Cases such as these have multiple Pareto optimal solutions. A Pareto optimal solution is defined as a solution that can not be improved in one objective without decreasing the other objective. All such solutions are said to be located along the Pareto optimal front (Holene, 2013). A random Pareto optimal solution is not necessarily the best solution for the problem, but the best solution is always among the Pareto optimal solutions. In the end it comes down to how important each of the two objectives are compared to the other.

2.2.1 Pareto optimality plot

A Pareto optimality plot has the objectives along the axes and solutions represented with their result for each objective along the axes. Given a problem with two objectives that should be minimized, the solutions forming the lower left edges of the solution space are the Pareto optimal solutions. An example of a Pareto optimality plot is shown in Fig. 2.3, here the goal is to minimize lead emissions while keeping down the costs.

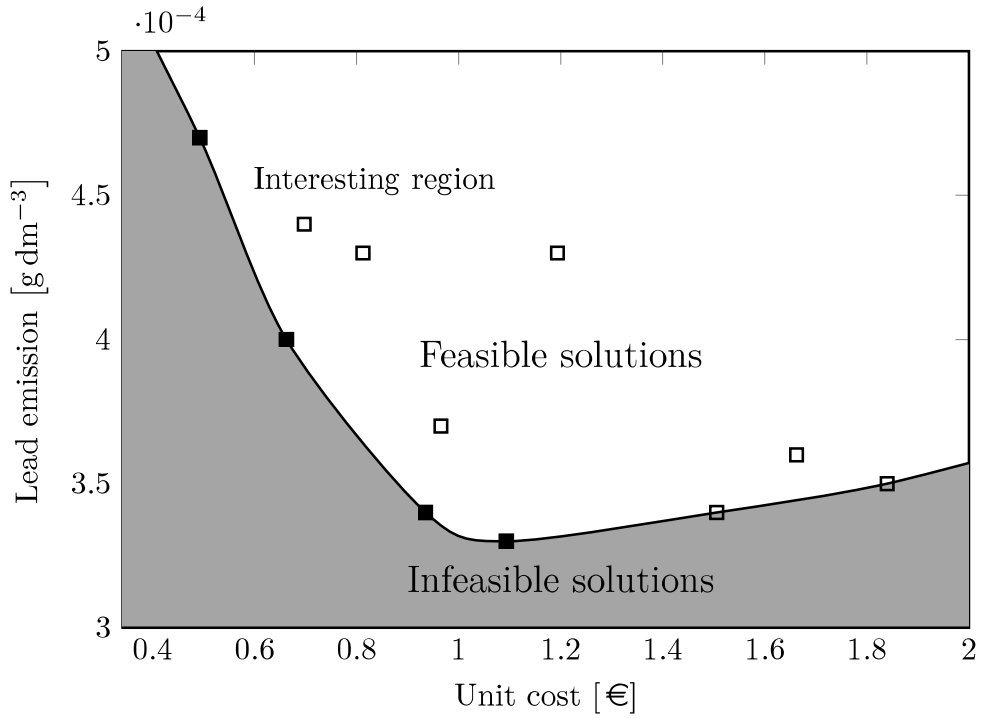

Figure 2.3: Pareto optimality plot for minimization of lead emissions and unit cost.

In the plot all squares represents solutions and all filled squares Pareto optimal solutions. Together the filled squares make up the Pareto optimal front which borders on the infeasible region. All of these are valid choices for a final solution, and the plot does not necessarily favor a particular point. The final decision depends on how heavily each objective is weighted.

Chapter 3

Modeling

Using the theoretical framework described in Section 2.1, a model of the system can be made. Section 3.1 describes the system, and defines the different state variables. Section 3.2.2 finds the differential equations governing the state, while Section 3.5 does the same on an energy basis. Section 3.3 describes the disturbances affecting the system and their validity. The modeling and simulation of the system using MATLAB and SIMULINK is described in Chapter 5 while all parameter and state values are shown in Appendix B.

3.1 System description

The system of interest is a hot water tank. Hot water tanks are not complex systems, and need not consist of more than a tank with inlet, outlet and a heater. This makes it a great system for modeling as the mathematical approximation is likely to be quite accurate. The hot water tank should be controlled in a way that ensures hot water to the end user, and minimizes the cost J. The cost of operating the hot water tank is defined as

$$J = \int_0^t pQ\mathrm{d}\tau. \tag{3.1}$$

Where p is the energy price and Q is the energy used by the tank heater.
A schematic of the hot water tank system is shown in Fig. 3.1. This particular hot water tank has a somewhat unconventional design. Firstly the cold water enters at the top of the tank and leaves out the bottom, while most tanks have cold water entering at the bottom. This does not have any implications for the model as our other unusual property, perfect mixing, makes the entering point of inlets and outlets trivial. Cruickshank and Harrison (2010) found that perfect mixing decreases the energy efficiency of the system, so this would definitely be a factor for other processes.

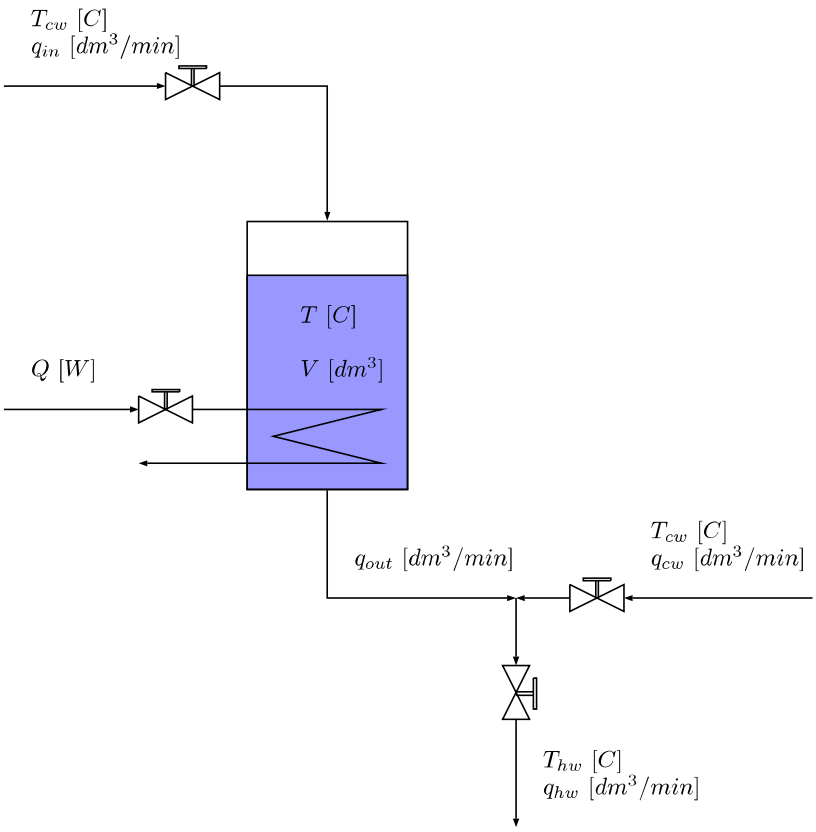

Figure 3.1: The hot water tank system.

The hot water tank system has cold water entering the tank with flow defined as q_{in} and T_{cw}. Because of perfect mixing, there is no temperature gradient and the tank temperature T is also the temperature out of the tank. The flow out of the tank q_{out} is mixed with a cold water flow q_{cw} to give desired temperature to the end user T_{hw}. It is assumed that control of q_{cw} is used to achieve this and that the control is perfect. This gives one of the key assumptions in our model, Eq. 3.2.

$$T_{hw} = T_{hw,s} \tag{3.2}$$

The flow out of the system q_{hw} is controlled by the end user, and will generally be referred to as the hot water demand.

3.2 State space representation

A hot water tank has two tasks, to provide water, and to ensure that the water provided holds the correct temperature. Keeping this in mind, the first choice of states $\mathbf{x}$ for the system is rather intuitive: Volume of water V and temperature T in the tank. In addition to those two, the cost function J is selected as the final state.
The states will have to be controlled using the valves available in Fig. 3.1. The

valve controlling the flow of cold water to be mixed q_{cw} is already used to control the outgoing temperature T_{hw}, while the flow of water out of the system q_{hw} is controlled by the end user. Thus the only valves left to control our states are the flow of cold water into the system q_{in} and the heater duty Q. This will be sufficient as we do not want to try to control the cost directly (Skogestad, 2014). J will therefore not be included in the outputs, $\mathbf{y}$. Disturbances are everything else entering or affecting the system that is not controlled. Putting this into state space notation gives

$$\mathbf{x} = \begin{bmatrix} V \\ T \\ J \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} V \\ T \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} Q \\ q_{in} \end{bmatrix} \qquad \mathbf{d} = \begin{bmatrix} q_{hw} \\ p \\ T_{hw,s} \\ T_{cw} \end{bmatrix}. \tag{3.3}$$

The disturbances $\mathbf{d}$ are not immediately obvious just from looking at Fig. 3.1, instead these are found from deriving the state derivatives, shown in Section 3.2.2. As specified in Eq. 2.1 state derivatives for linear systems are functions of inputs, disturbances and parameters. Thus, finding the state derivatives when inputs and states are known will make the disturbances obvious.

3.2.1 Constraints

Both the outputs $\mathbf{y}$ and inputs $\mathbf{u}$ are constrained. The constraints on the q_{in} and Q are due to physical limitations of the valve and heater. The constraints on V and T are there partly for safety reasons, such as making sure that the tank doesn't take in more liquid than it can hold, that water does not start boiling and that the heating element always is covered by water. The other reason is simply to meet control objectives, which is not possible if there is no water in the tank or the tank temperature T is below the hot water temperature set point $T_{hw,s}$. Input and output constraints are shown in Table B.4.

3.2.2 Finding the state derivatives

State derivatives show how the system states, in this case T, V and J, changes with time. They are generally found from manipulating mass and energy balences. Setting up the mass and energy balances gives Eqs. 3.4 and 3.5.

$$\frac{d(\rho V)}{dt} = \rho_{in} q_{in} - \rho_{out} q_{out} \tag{3.4}$$

$$\frac{dH}{dt} = \dot{H}_{in} - \dot{H}_{out} + Q - Q_{loss} \tag{3.5}$$

Here ρ is the density of a stream in kg/l and q is the volumetric flow in l/s. $\dot{H}_{in}$ and $\dot{H}_{out}$ are the enthalpies of the streams in and out of the system in J/s. Q is the duty delivered from the heater and Q_{loss} is the loss of energy to the environment.

By assuming the density change with temperature to be negligible the mass balance can be simplified to Eq.3.6.

$$\frac{dV}{dt} = q_{in} - q_{out} \tag{3.6}$$

The flow leaving the hot water tank, q_{out}, depends on the hot water demand and the temperature of the tank. If the tank holds a high temperature, q_out and q_{cw} will be roughly equally large. If T is low however, q_{out} will be much larger than q_{cw}, to meet the desired hot water temperature of 50 °C. q_{out} is a function of state variables and has been derived in Appendix A.2. By inserting Eq. A.16 into Eq. 3.6 the volume derivative becomes

$$\frac{dV}{dt} = q_{in} - q_{hw}\frac{T_{hw,s} - T_{cw}}{T - T_{cw}}. \tag{3.7}$$

The full derivation of the temperature balance from the energy balance in Eq. 3.5 is shown in Appendix A.1. It gives the temperature derivative

$$\frac{dT}{dt} = \frac{1}{V}\left(q_{in}(T_{cw} - T) + \frac{Q - Q_{loss}}{\rho c_p}\right), \tag{3.8}$$

where c_p is the specific heat capacity of water.
The final state derivative is the cost, which is found by differentiating Eq. 3.1.

$$\frac{dJ}{dt} = pQ \tag{3.9}$$

All of the state derivatives put into state space notation then gives

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{V} \\ \dot{T} \\ \dot{J} \end{bmatrix} = \begin{bmatrix} q_{in} - q_{hw}\frac{T_{hw,s} - T_{cw}}{T - T_{cw}} \\ \frac{1}{V}\left(q_{in}(T_{cw} - T) + \frac{Q - Q_{loss}}{\rho c_p}\right) \\ pQ \end{bmatrix}. \tag{3.10}$$

3.3 Disturbance analysis

The disturbances affecting the system are q_{hw}, p, $T_{hw,s}$ and T_{cw}, of which the latter two are assumed to be constant. Technically constant disturbances could be considered parameters, but as they are included as disturbances in the model they will be in the thesis as well. The reasoning behind assuming $T_{hw,s}$ to be constant, is that the user would have some other cold water source to mix the water with if a colder temperature was wanted. For T_{cw} there might be some seasonal changes, but for a simulation running for one month the variation is likely to be negligible. The two remaining disturbances are described in sections 3.3.1 and 3.3.2.

3.3.1 Hot water demand, q_{hw}

The hot water demand is controlled by the end user; the owner of the domestic hot water tank. The hot water is used for task such as: Baths, showers, dishwasher

and in smaller or larger bursts bursts for cooking or cleaning. The use of hot water for one day in an average household was simulated by de Oliveira et al. (2013), using controlled randomness to generate different profiles for each simulation. The function `getProfile.m` and its subfunctions are shown in Appendix D and has not been made or modified as part of this thesis.
By running the program repeatedly an average hot water demand profile could be generated. The average hot water demand for each hour of the day is shown in Fig. 3.2.

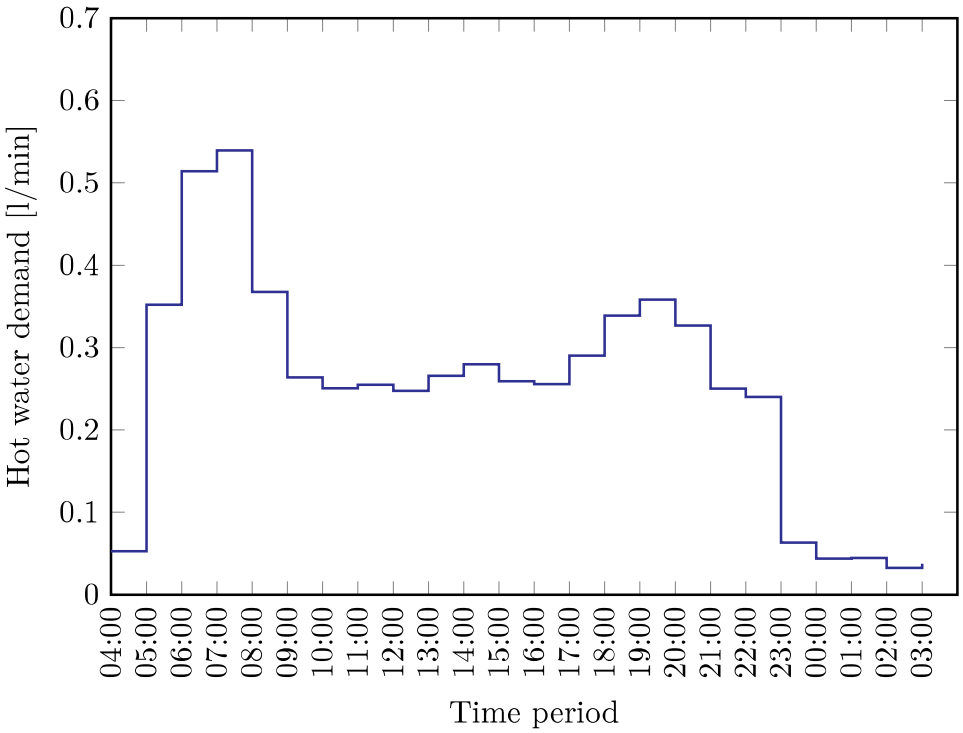

Figure 3.2: The average demand for each hourly period, as estimated from the result of 365 unique demand simulations.

3.3.2 Electricity price, p

Electricity price is strongly dependent on demand, and as such also varies according to time of day and season. General price trends in Norway are low prices during the night and higher during the day. There is generally a spike in price in the morning and sometimes a spike in the evenings as well. The average electricity price for each hourly slot is shown in Fig. 3.3.

In Norway the electricity market is regulated by the Norwegian Water Resources and Energy Directorate (NVE). Together with national regulators in Estonia and other Nordic countries they operate the energy trading market. NVE has granted Nord Pool Spot a license to operate a marketplace for trade of electricity in the Norwegian market (Lund, 2012). In its function as marketplace Nord Pool Spot (2014) also provides some interesting public data. Among these electricity prices

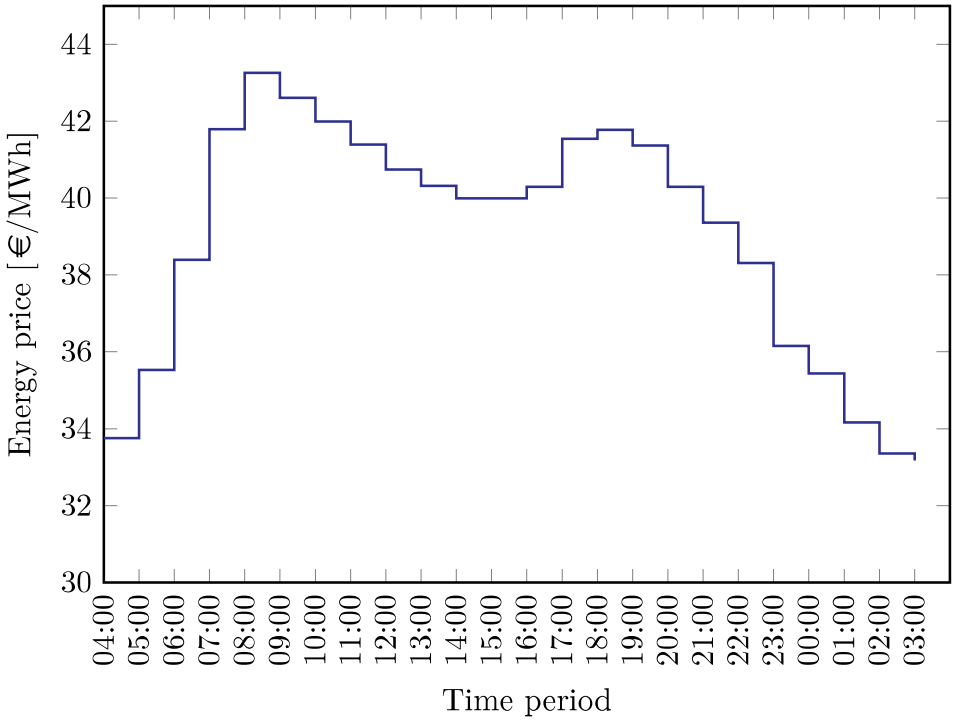

Figure 3.3: The average price for each hourly period, from Nord Pool Spot prices for the Trondheim area 2013. Nord Pool Spot (2014)

for every hour in the Nordic countries from 2011 and onwards and daily prices dating back to 1996. The hourly price data from March 2013 is used to represent the disturbance p in this thesis. The reasoning behind using this specific data set is twofold. Firstly March 1st contains 2013's highest energy price, which makes it interesting as the model would have to work for high disturbances. Another advantage is that higher price variations should give clearer results, as no price variation would give the same cost no matter when heating takes place.
The choice of starting at 4:00 in the morning is based on giving reasonable initial and final conditions. From figures 3.3 and 3.2 it can be seen that the price and demand is generally lowest right around 4:00. Assuming that it is optimal for the tank to be full and at maximum temperature at this time seems reasonable. By choosing 4:00 as a starting point our initial and final condition is very likely to be optimal, which strengthens the accuracy of our model. A look at the electricity price for the first four days of March is shown in Fig. 3.4.

From the plot it can be observed that prices vary quite a bit from day to day, but still tends to follow the same pattern as seen in the average graph. Intuitively this tells us that a simple policy based on heating the tank during the night could give good results, which will be investigated in Section 4.3.

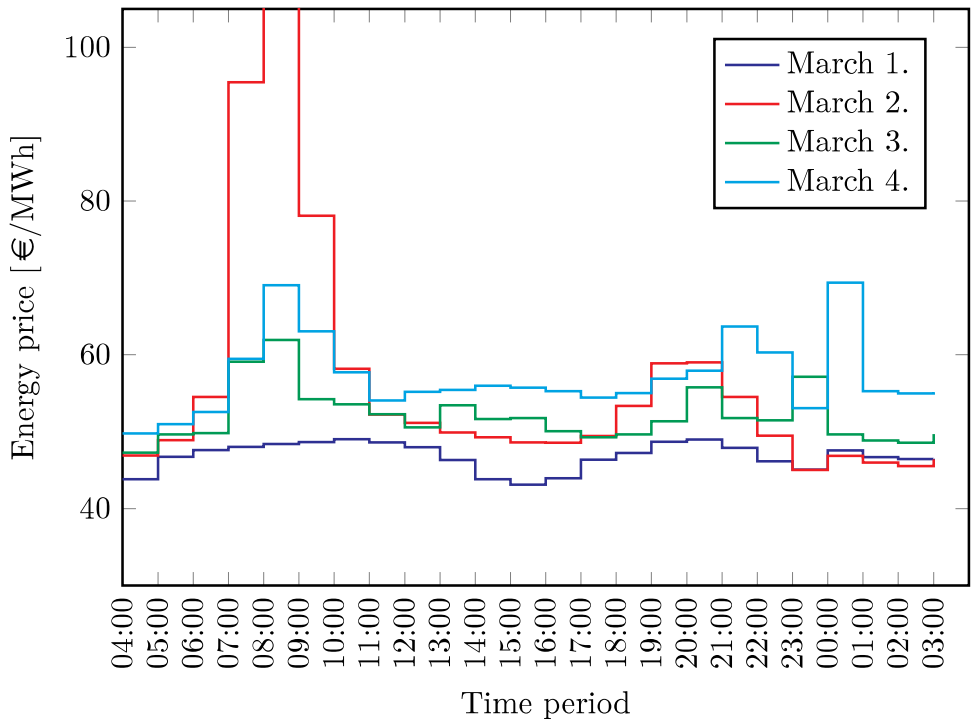

Figure 3.4: Price for each hourly period for the first four days of March, from Nord Pool Spot prices for the Trondheim area 2013 (Nord Pool Spot, 2014).

3.4 Controllers

The pairing of inputs as well as choice and tuning of controllers is selected to be the same as proposed by Johansson (2013). This was done both to speed up the process of recreating the model and to keep the systems consistent. The method used for tuning the controller and the theory behind it will therefore not be presented here, but can be found in the thesis of Johansson.
Feedback control with PI controllers was used to stabilize the system states T and V. The pairing of inputs with states was as would be expected; Q was used to control T while q_{in} controls V. The control scheme is depicted in Fig. 3.5. It has been assumed that the valve controlling the flow of cold water q_{cw} is used to ensure that $T_{hw} = T_{hw,s}$, and that this control is perfect. The two other controllers have been tuned by Johansson, which found the tuning parameters shown in Table B.1.

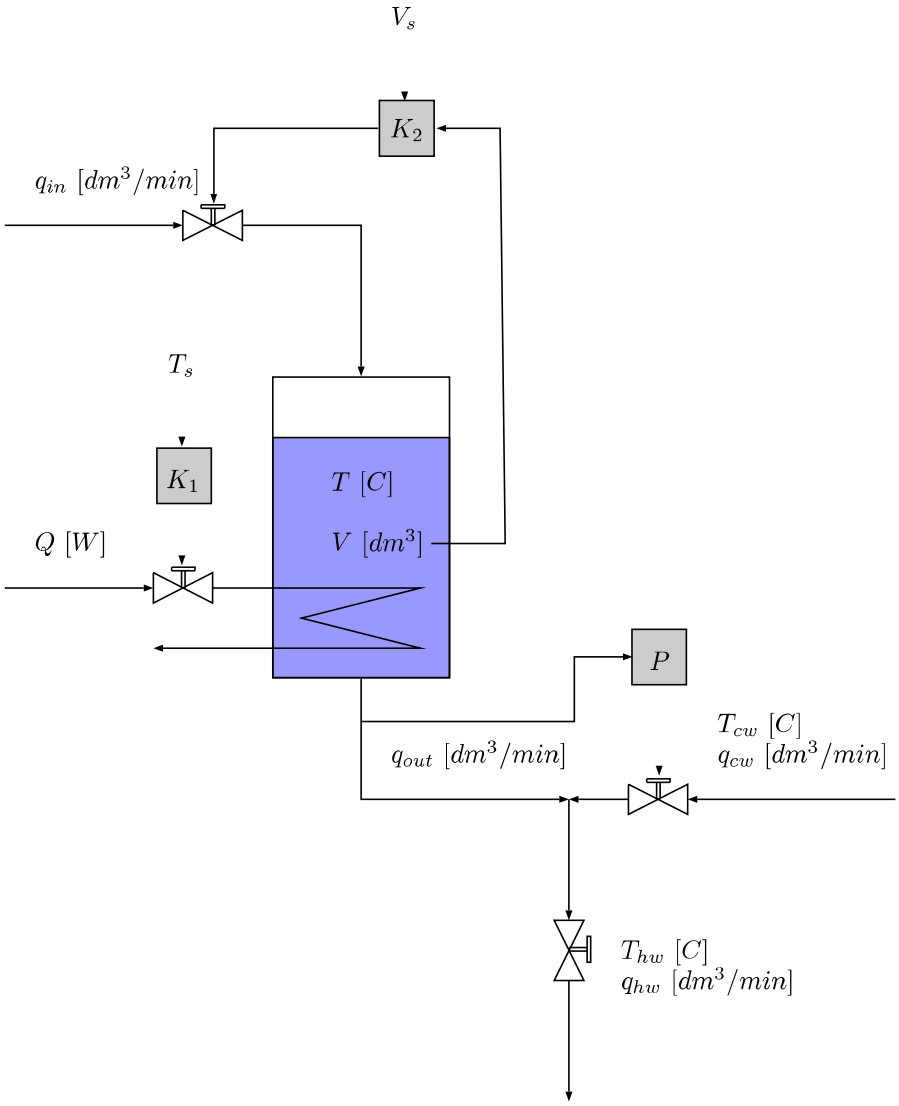

Figure 3.5: Control scheme for the hot water tank system.

3.5 Describing the system on an energy basis

Describing the system on an energy basis is easier than using a temperature and volume basis. It is rarely used for control as energy is hard to measure. In this thesis the energy level in the tank will be used in Section 4.2. The energy in the tank E is defined as

$$E = \rho c_p V (T - T_{cw}). \tag{3.11}$$

The upper and lower bounds on energy follow from the bounds on T and V as

$$E_{max} = \rho c_p V_{max} (T_{max} - T_{cw}), \tag{3.12}$$

$$E_{min} = \rho c_p V_{min} (T_{hw,s} - T_{cw}). \tag{3.13}$$

The change of energy in the tank depends on heater duty Q, energy lost due to demand Q_d and heat loss Q_{loss}. Assuming negligible heat loss yields the modified energy balance

$$\frac{\partial E}{\partial t} = Q - Q_d. \tag{3.14}$$

Where Q_d can be calculated from the energy in the hot water flow q_{hw}, and is defined as

$$Q_d = q_{hw} c_p \rho (T_{hw} - T_{cw}). \tag{3.15}$$

3.6 Optimal control, simplifying the problem

The previous section has shown how the outputs and inputs have been paired to achieve a stable system. It does not show how to control inputs to achieve additional objectives, like minimizing the total cost J. In essence, controlling the model optimally is all about finding a solution to the optimization problem in Eq. 3.16.

$$\min_{T,V} J(T, V) \tag{3.16}$$

Subject to the constraints shown in Table B.4. Solving this problem by optimizing the inputs requires resolving the problem very often to handle any new disturbances. This is why almost all cases of control optimization simplifies the problem by finding optimal input set points and let regular controllers handle the disturbance rejection. Doing so lets us keep the controllers found in Section 3.4 and gives the simplified problem

$$\min_{T_{set}, V_{set}} J(T_{set}, V_{set}). \tag{3.17}$$

Subject to the same constraints as in Eq. 3.16.
Solving an optimization problem with multiple variables is not fast, and so it would be useful if the number of variables could be reduced by assuming one variable to be a function of the other,

$$\min_{x,y} f(x, y) \approx \min_{x} f(x, y(x)), \tag{3.18}$$

or by finding one variable to be close to optimal at some constant value, known as a self-optimizing variable (Skogestad, 2004). The goal of self-optimizing control is to find a variable that is insensitive to disturbances at the optimum. A good guess for a slow moving variable for our system would be the volume while the tank is fully filled. High volume means increased resistance to disturbance, and it seems intuitive that $V = V_{max}$ would be an active constraint for periods when the price is low, as the goal is to store as much possible energy as possible. Similarly when the demand is high, as this would help resist disturbances. Overall it seems likely that keeping $V = V_{max}$ would be a near optimal option for large sections of the simulation.
If the demand is too high however, continuously filling the tank with cold water would lead to the temperature falling below its constraint. If this happens the flow of cold water into the tank must be closed until the temperature rises above

its set point. For cases where water volume reaches its minimum constraint the temperature constraint would have to be abandoned and the volume of water kept constant to ensure that water covers the heating element.
If the volume could be controlled as stated above, it could provide close to optimal behavior to just optimize on the temperature. Putting this into logical statements gives the rules that de Oliveira et al. (2013) proposed for near optimal behavior:

1. Manipulate q_{in} to keep $V = V_{max}$ while $T > T_{hw,s}$
2. Set $q_{in} = 0$ while $T = T_{hw,s}$ and $V > V_{min}$
3. Manipulate q_{in} to keep $V = V_{min}$ while $T < T_{hw,s}$

mplementing this requires a control structure that changes with temperature. This can be implemented in SIMULINK using the switch block object. The control structure will have two different modes of operation. The first mode is active for all periods where $T < T_{hw,s}$, in this mode $V_s = V_{max}$. The second mode is active for all states where $T \leq T_{hw,s}$, here $V_s = V_{min}$.

What this accomplishes is changing the set points once the active constraint region changes (Jacobsen, 2011). This way the self-optimized variable can remain near optimal for new constraint regions. Fig. 3.6 shows how the system behaves with a constant hot water demand of $2.5l/min$.

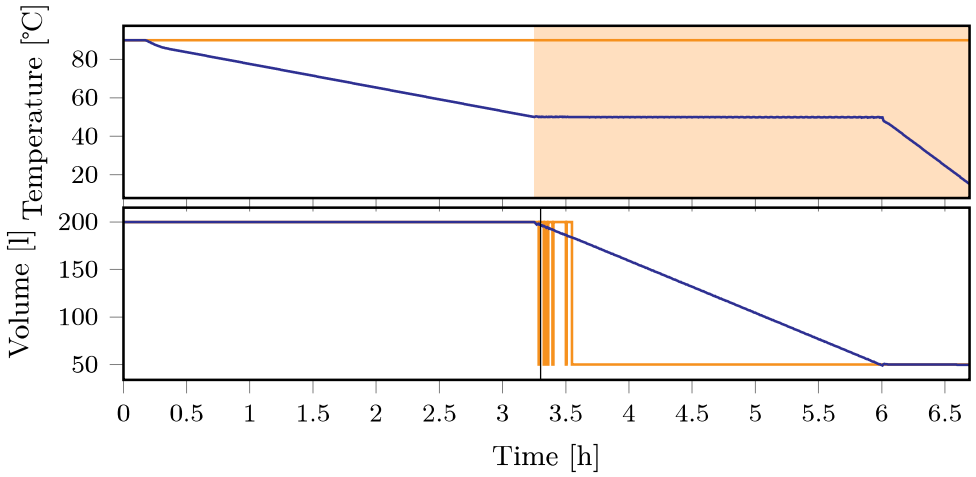

Figure 3.6: System response to a constant hot water demand of $2.5l/min$ with parameters $V_{max} = 200$, $V_{min} = 50$ and $T_{hw,s} = 90$.

From Fig. 3.6 all three optimizing rules can be observed.

- In the first section $V = V_{max}$ as recommended from rule 1
- In the second section $T = T_{hw,s}$ as recommended from rule 2
- In the third section $V = V_{min}$ as recommended while $T < T_{hw,s}$ in rule 3.

With implementation of this rule, described in Section 5 the optimization problem is reduced to Eq. 3.19.

$$\begin{aligned} &\min_{T_{set}} J(T_{set}) \\ &\quad T_{min} \leq T \leq T_{max} \\ &\quad V_{min} \leq V \leq V_{max} \end{aligned} \tag{3.19}$$

This will be the problem to be investigated in the case studies.

Chapter 4

Cases

4.1 Case I: The base case

In Section 3.6 logical control rules were used which simplified the optimization of the system down to one variable: The temperature set point, T_s. The easiest way to handle the variable is to keep it at a constant value which is what have be done in Case I. Here the temperature set point is kept constant at the maximum temperature, 90°C. The advantage of this method is that the temperature in the tank is very unlikely to go below the minimum temperature constraint with the disadvantage being that it is economically far from optimal. There is no adjustment to the electricity price or planning ahead involved, which means that the energy storage potential of the tank is not utilized.
Case I will mainly serve as a measuring stick for the more complex cases.

4.2 Case II: The optimized case

For Case II the temperature set point is going to be determined using model predictive control (MPC). The MPC controller finds a set of optimal temperature set points by solving an optimization problem. The optimization problem being to minimize the total cost, J for a 24 hour period subject to the constraints in Table B.4. This problem is resolved for every hour of simulation using updated system states. Theoretically this should give temperature set points that grants both a stable system and low cost.

4.2.1 Modifying the optimization problem

The goal of any optimization problem is to maximize or minimize a function while abiding system constraints. Here we want to minimize is the cost function J while keeping the constraints specified in Table B.4. The cost function J is an integral

containing the price p and heater duty Q. By discretizing the problem into 24 hour long sections the minimization problem becomes

$$\min f(u) = \sum_{i=0}^{n} Qp\frac{N}{n}, \tag{4.1}$$

where N is the size of the prediction horizon and n specifies the granularity of the discretization both being 24 for our case. The only variable in the minimization problem that the MPC controller can influence is Q, which will be controlled using the temperature set point, T_s.

To optimize on a temperature basis is difficult as the change in temperature is not linear, see Eq. 3.8. A simplification would be to instead optimize the total energy level in the tank. From Eq. 3.5 it can be seen that this will give a linear problem, which would make optimizing really fast. Each new set point is calculated using

$$E_{i+1} = E_i + \frac{\partial E}{\partial t}\Delta t \tag{4.2}$$

$$= E_i + (Q - q_{hw}c_p\rho(T_{hw} - T_{cw}).\Delta t \tag{4.3}$$

Instead of using the temperature constraints the energy constraints listed in Eqs. 3.5 and 3.5 must be used.
Several other assumptions must be made to make Eq. 4.2.1 implementable. The unknown hot water temperature T_{hw} must be assumed to be equal to its set point $T_{hw,s}$ and the demand has to be replaced by a predicted demand. To predict the demand hourly average demands from one year of simulations was found. This average demand, $q_{hw,av}$ was then used to predict the next energy state.

For initial and final conditions it has been assumed that the tank is filled and heated both at the beginning and at the end of simulation.

$$E_0 = E_{max} \tag{4.4}$$

$$E_{end} = E_{max} \tag{4.5}$$

Where E_{max} is defined in Eq. 3.5. The temperature set points could then be extracted from the energy set points by assuming constant volume.

4.3 Case III: The Two-phase Case

It is of interest to find how Case II compares to a simpler heating policy. An example of such a policy is to increase T_s to the maximum at 23:00 and keep it constant at this level until 6:00 to take advantage of generally low prices in this time period. The other hours of the day T_s would be at a minimum level to conserve energy. This policy has two phases, the heating phase from 23:00-6:00 and the conserving phase taking up the rest of the day. It will therefore be referred to as a two-phase case.

In order to find the best two-phase policy multiple cases with varying length of heating phase and heating start time have been investigated. The best of the simple cases, have been defined as Case III.

4.4 Variable analysis

In addition to comparing the cases, it is of interest to find out how some of the key variables affects each case. A primary objective of the hot water tank is to always be able to deliver water of sufficiently high temperature. To do so the temperature in the tank must always be above or equal to the minimum temperature for the hot water set point,

$$T \geq T_{hw,s}. \tag{4.6}$$

When the PI controller has a set point exactly equal to $T_{hw,s}$ there will be quite some violation of Eq. 4.4, as the temperature would be expected to be oscillating around the set point. To decrease the time below the set point, and so the amount of time violating the temperature constraint, back-off can be implemented. Adding a back-off implies adding a buffer to the temperature set point. Instead of setting the temperature set points at the exact value that is wanted, a safety margin is incorporated.

$$\text{Back-off} = \text{Constraint} - \text{Set point}. \tag{4.7}$$

If a constraint can be violated (soft constraint) the back-off only need to compensate for measurement error. If not, (hard constraint) the back-off must also compensate for dynamic controller error. With improved tuning back-off can be reduced. This is referred to as "squeezing and shifting" as improved tuning squeezes the variance, which allow for the set point to be shifted (Rawlings and Stewart, 2008). Implementation of back-up should be expected to reduce t_v but increase J as less volume is usable for control. By measuring t_v and J for different amounts of back-off, an optimal tradeoff between the two can be found.
It is also interesting to see how the size of the hot water tank relative to q_{hw} affects the results of simulation. With the assumption of $Q_{loss} = 0$ there will be no increased loss of heat due to increased surface area of the tank, so a bigger tank should be able to outperform smaller tanks for the optimized case. Both in terms of disturbance rejection, as the disturbances do not scale with volume, and in terms of smaller cost. For the base case the size of the tank size should not affect the result, though the disturbance rejection should improve with tank size.

Chapter 5

Implementation

The modeling of the physical system was carried out using SIMULINK and MATLAB. SIMULINK is a block based simulation environment, that is fully integrated with MATLAB, meaning code-snippets can be used to expand on the model. This is used to run the optimization that estimates temperature set points for Case II.

5.1 Program structure

The program is built up to work with MATLAB as the running environment and `main.m` as the primary program. The input argument `inmode` specifies which case `main.m` should run. If the simulated case is Case I, the simulation is carried out as shown in Algorithm 2. if it is Case II or Case III, the simulation is carried out as shown in Algorithm 3.

Algorithm 2 Solution procedure for `inmode`= 1

1: `main.m` specifies initial conditions and tells SIMULINK to run until $t = t_{end}$
2: SIMULINK runs model for given inputs and time, returns state variables
3: `main.m` saves the results to a .mat file

In addition to the hierarchy shown in algorithms 2 and 3, there is one layer above the `main.m` function. These scripts changes one or several of the initial conditions and calls `main.m` repeatedly. This has been used to test how the model responds to changes in tank volume and back-off. It has also been used to find the best two-phase heating policy. The scripts that do this are `simplec.m`, `sizing.m` and `bu.m`
The function `main.m`, its incoming arguments and the programs that call it are all described in more detail in Appendix C. Here all MATLAB scripts and functions used for the simulations are also supplied.

Algorithm 3 Solution procedure `inmode`= 2 or 3

1: `main.m` specifies initial conditions
2: **for** $t = 0, 1, 2, ...t_{end}$ **do**
3: `main.m` finds the next T_s using either two-phase method or MPC
4: `main.m` tells the SIMULINK model to run for one hour with given conditions
5: SIMULINK runs model for given inputs and time, returns state variables
6: `main.m` updates conditions
7: $t = t + 1$
8: **end for**
9: `main.m` saves the results to a .mat file

5.2 SIMULINK model

The mathematical system described in Chapter 3 was implemented in SIMULINK. The finished model is shown in Fig. 5.1.

For each timestep in the model, the PI controllers determines the inputs $\mathbf{u}$ to the system. Simultaneously the disturbances $\mathbf{d}$ are defined in MATLAB and imported into SIMULINK in the block [profile]. The inputs and disturbances are both sent to the block [system.m]. Here the next state of the system is calculated from solving Eq. 2.3, using the state derivatives found in Section 3.1. The measured states $\mathbf{y}$ are then sent to their respective PI controllers, to calculate inputs for the next timestep of the simulation. This cycle continues for each new timestep until the end of the simulation. The simulation length is specified by `main.m`, and has for this thesis been set to 30 days.

The rest of the model focuses on controlling the set points of the PI controllers $\mathbf{y_s}$. In Section 3.6 it was described how self-optimizing control can reduce the difficulty of the optimization problem. By using logical rules to control the set point of the volume, the optimization problem was reduced to one variable the temperature set point T_s. This left the volume set point V_s to be set according to the value of the temperature. The logical rule is implemented using the block [Switch], which has the temperature T and the two constraints V_{max} and V_{min} as input. The switch block adjusts V_s according to the rule shown in Algorithm 4.

Algorithm 4 Switch block rules

1: **if** $T \leq T_{hw,s}$ **then**
2: $V_s = V_{min}$
3: **else**
4: $V_s = V_{max}$
5: **end if**

The only difference between the three cases are the incoming temperature set point T_s, which is calculated in MATLAB and imported into SIMULINK in the block [90].

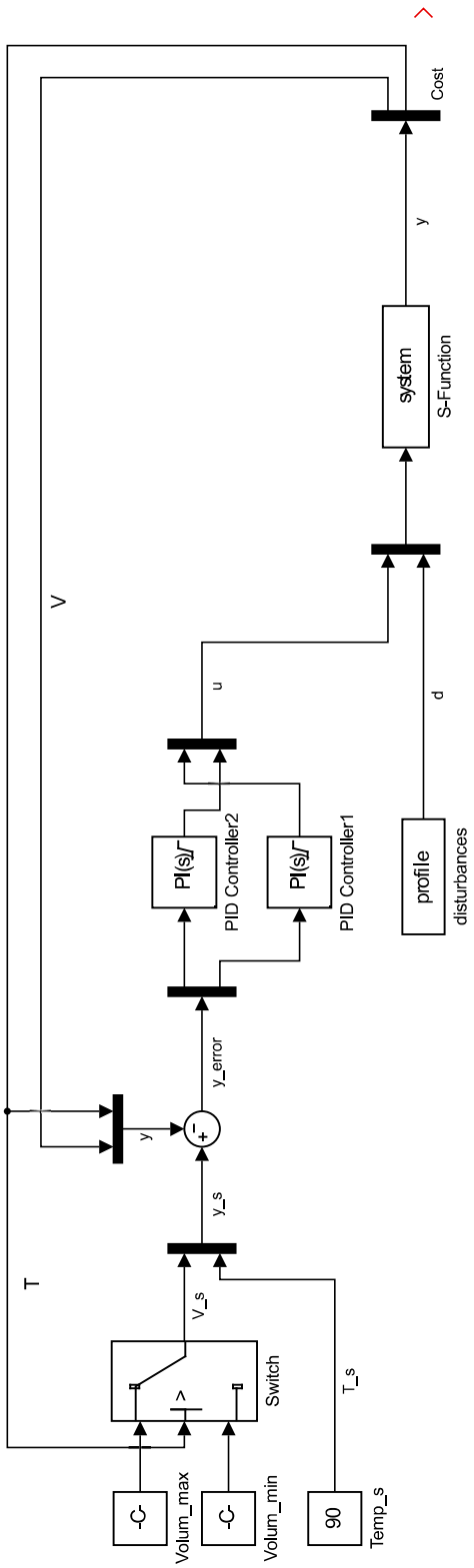

Figure 5.1: The hot water tank system modeled in Simulink.

Chapter 6

Results

This chapter presents the results from simulations of the base case, the optimized case and the two-phase case. For all state plots except Fig. 6.11, blue lines represent a state variable while orange lines represent a set point.

6.1 72-hour plots for cases I, II & III

This section shows how the implemented heating policies impacts the state for the three first days of simulation. For Case I inputs and disturbances are also included. This is to give the reader a general understanding of how the control system responds to disturbances. Fig. 6.1 shows how the states develop during the first three days of simulation for Case I, while Fig. 6.2 shows the inputs and disturbances. The constant disturbances T_{cw} and $T_{hw,s}$ are not included. For Case II, its states and set points along with the evolution of the price are shown in Fig. 6.3. The very same variables have been plotted for Case III in Fig. 6.4.

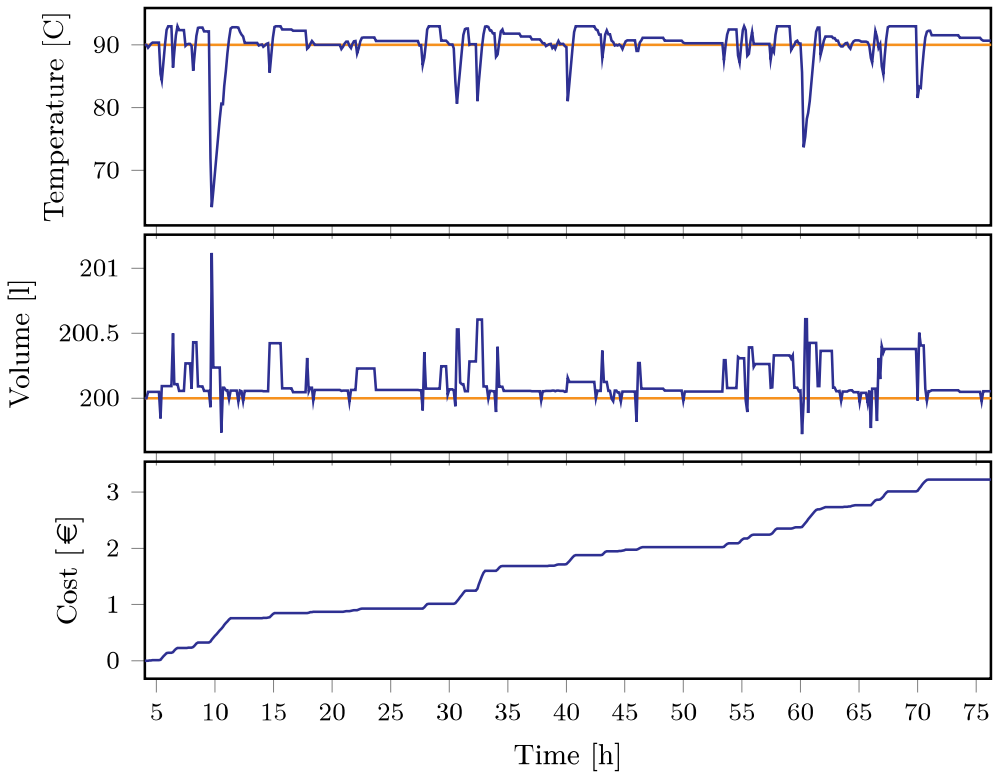

Figure 6.1: Temperature and volume with set points in orange for Case I. The plots show outputs from three days of simulation with the cost function included.

The volume set point is constant at 200l with the state keeping tight to the set point. Meanwhile the temperature has several deviations from its set point, with the biggest disturbance happening at approximately 9 hours into the simulation. When the temperature drops the volume tends to spike slightly just afterwards, which typically indicates a small overshoot in the controller that refills the tank.

The hot water demand q_{hw} and the flow into the tank q_{in} appear almost identical but there is a difference in scale along the y-axis for the two plots. The spike in demand that caused the temperature in the tank to drop can be noticed after approximately 9 hours. This spike also pushes the heater duty Q to work at maximum intensity for over two hours. There is no hot water demand when the price is at its highest, but this is just a coincidence as the hot water demand is randomly generated and do not correspond to a real date.

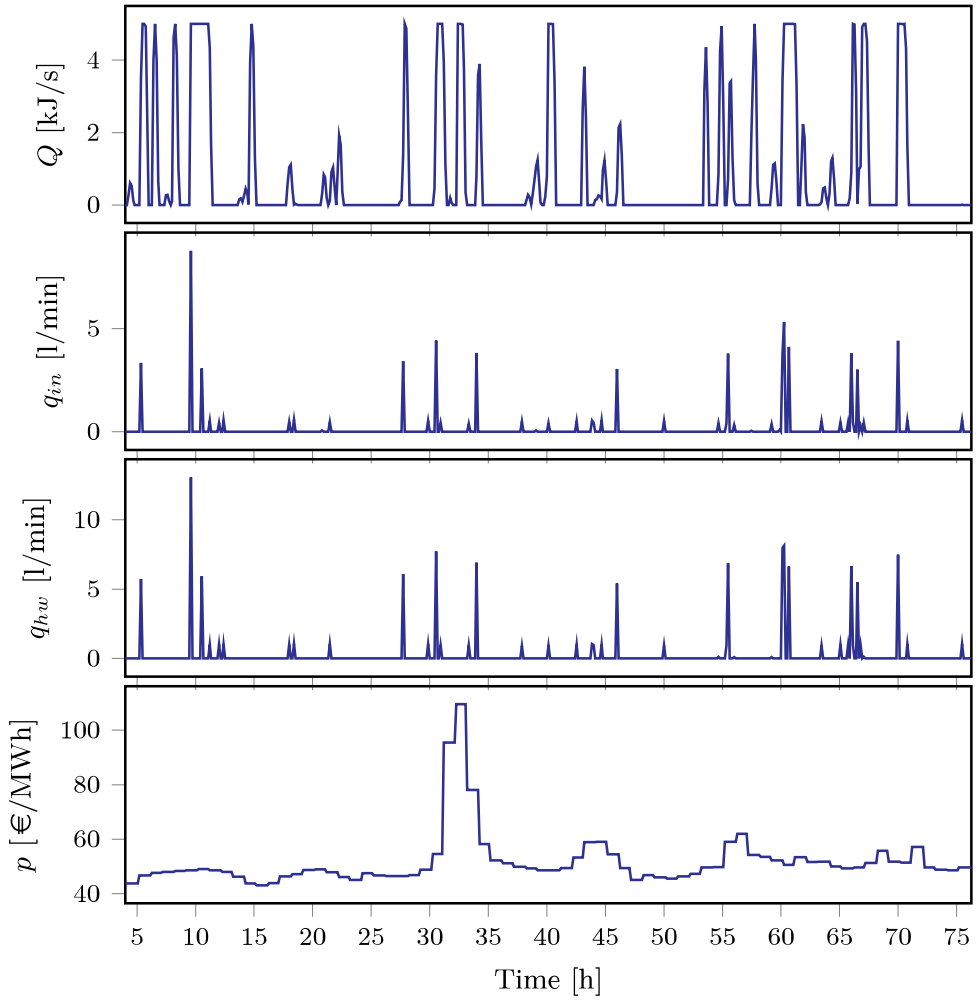

Figure 6.2: Inputs and disturbances during three days simulation of Case I.

Case II and Case III has apparent changes due to spikes in demand, leading to a change in volume set point for Case II after 9 hours and case III after 70 hours. This is the expected response once the temperature drops below $T_{hw,s}$ and is described in Section 3.6.

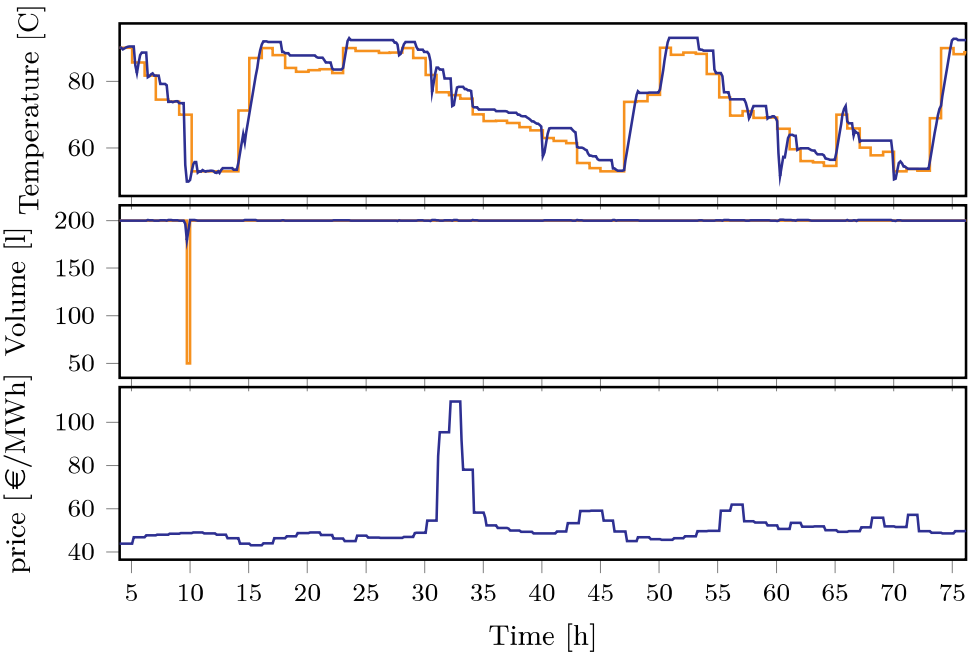

Figure 6.3: Case II Temperature and volume with their respective set points in orange. The price function is also included.

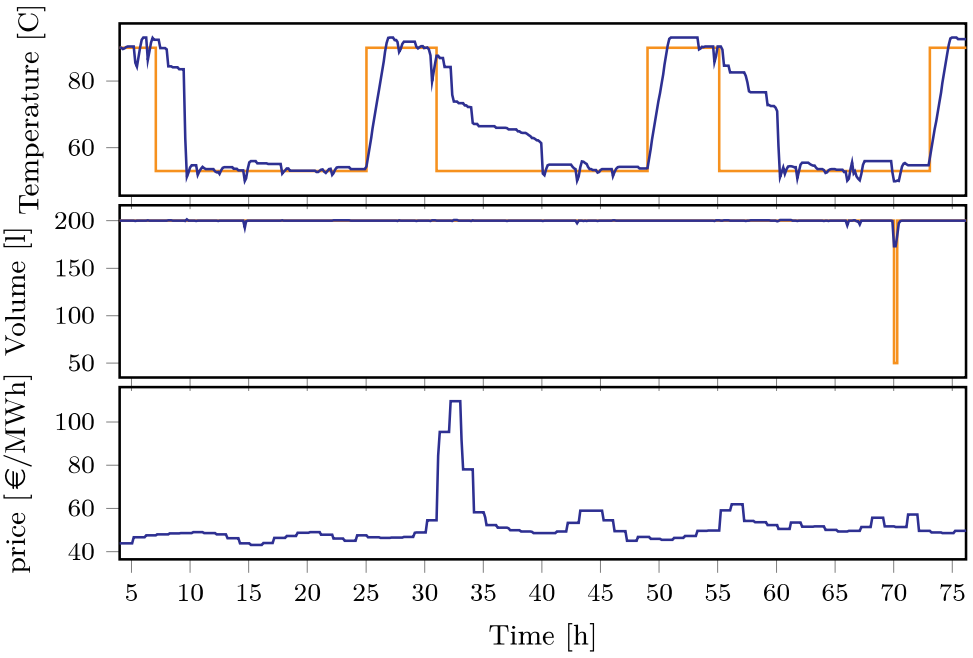

Figure 6.4: Temperature and volume with set points in orange for Case III. The plots show results from three days of simulation with the price function included.

6.2 Resulting t_v and J for cases I, II & III

Two objectives are used to qualify the cases; the amount of time that the tank temperature is below the minimum hot water temperature set point, t_v, and the final cost J. The evolution of J with time for each of the cases is shown in Fig. 6.8.

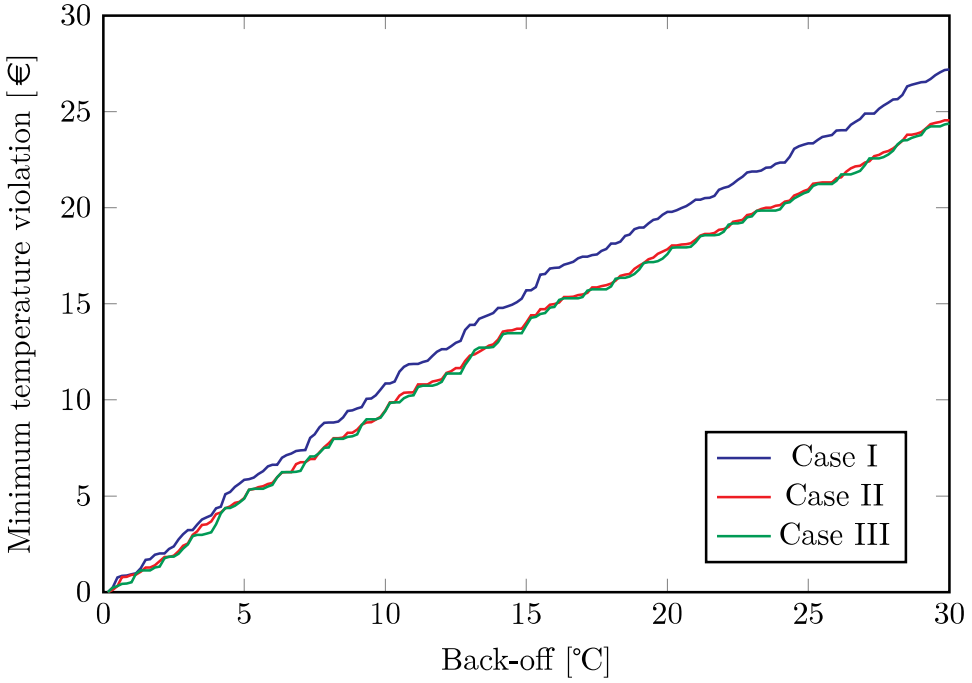

Figure 6.5: The cost J as a function of time for Case I, Case II and Case III.

Which shows that Case I has a noticeably higher cost than Case II and III, while the latter two are hard to separate. The cost at the end of simulation for each case is presented in tabular form in in Table: 6.1. Here the performance in terms of t_v for each case is also included. The cost for Case II is a fraction higher than for Case

Table 6.1: Resulting J and t_v for Case I, II and III.

Objective	Case I	Case II	Case III
J	27.19	24.85	24.71
t_v	0	4.56	6.62

III. Both Case II and III have violation of the temperature set point, with Case III having some hours more. Case I has no temperature violation. The savings compared to Case I are 8.61% and 9.12% for Case II and Case III, respectively. This added up to a profit of profit of 2.34 € and 2.48 € for Case II and Case III.

6.3 Selection of optimal back-off

Case II and III were tested with different back-offs with particular weight put on the results from Case II. The optimal back-off was determined by measuring the two objectives t_v and J. To To help decide between the different alternatives, a Pareto optimality plot of the solutions was made. This is shown in Fig. 6.6. From the plot it was decided to use a back-off of 3 °C, the selected solution is marked in the figure as O.S.

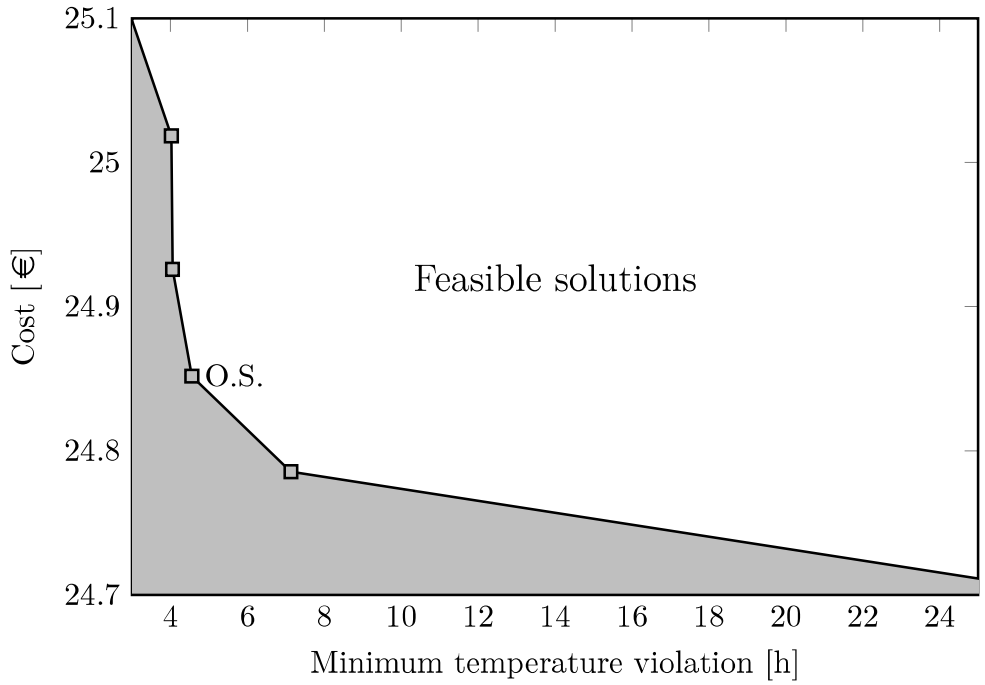

Figure 6.6: Pareto plot of the different back-off values, where each square represents a different solution. The solutions are all Pareto optimal. O.S. signifies the selected solution.

As all solutions are Pareto optimal, the choice of back-off is not self-evident. This is discussed further in Chapter 7.
Cost as a function of back-off for both Case II and Case III is shown in Fig. 6.7, while t_v as a function of back-off is shown in Fig. 6.8. It can be observed that Case II is more affected by the change from no back-off to 1 °C of back-off, but that the cases otherwise seem to perform rather similar. For both cases J increases with back-off whilet_v decreases. for the tested back-offs the improvement in t_v from increasing the back-off decreases after 3°C.

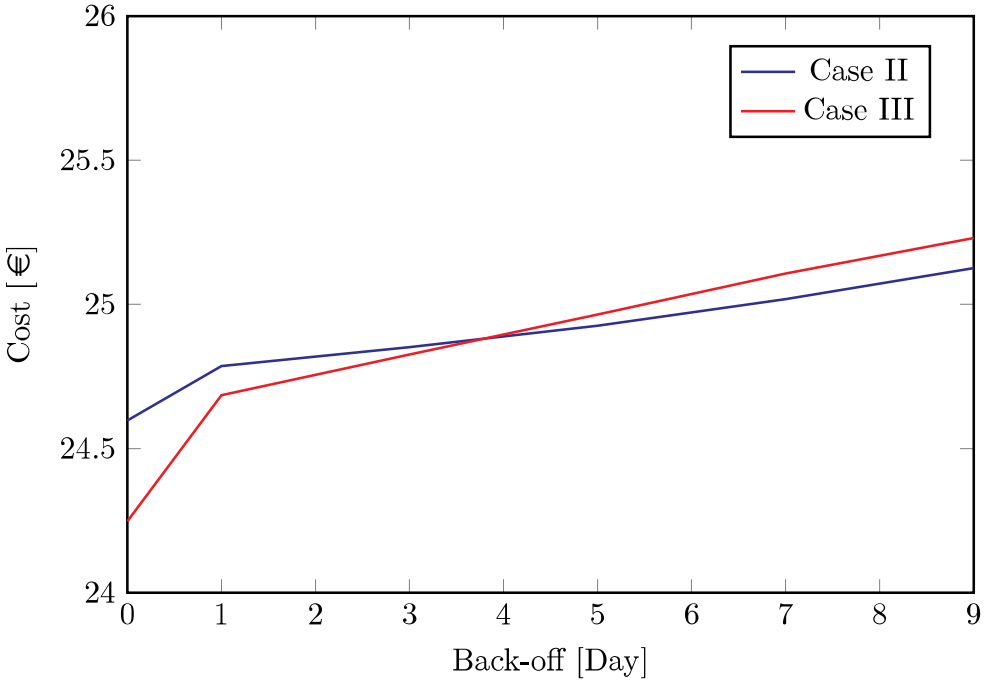

Figure 6.7: Cost for Case II and Case III as a function of back-off. Simulated for 30 days with a tank volume of 200l.

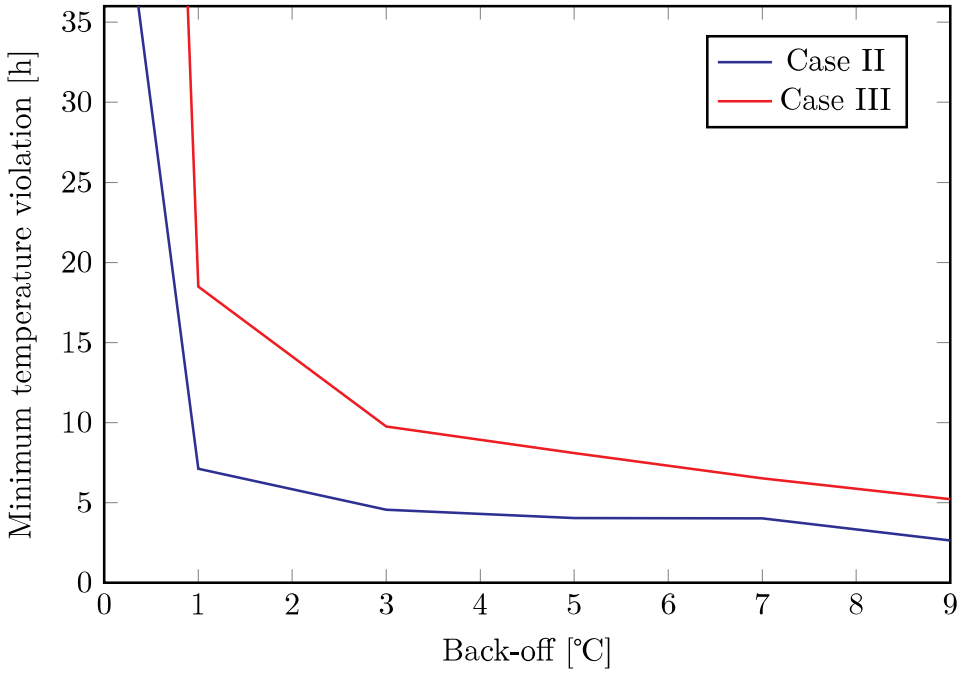

Figure 6.8: Hours of $T \leq T_{hw,s}$ for Case II and Case III as a function of back-off. Simulated for 30 days with a tank volume of 200l.

6.4 Selection of two-phase policy

The two-phase case used in the other sections was chosen after comparison of 24 different policies. The tested control policies were similar in that they all contained two phases, a high temperature set point phase (heating phase) and a low temperature set point phase (conserving phase). The simple control policies differed from each other by the time spent in each phase and the starting time of the heating phase. The starting time of the heating phase varied from 22:00 to 1:00 while the length of the heating phase varied from 3-8 hours. Table B.6 shows the different cases and their performance in terms of minimum temperature violation and cost for thirty days of simulation with 3°C back-off.
To choose one policy to continue with from the cases, an Pareto optimality plot was made, the plot is shown in Fig. 6.9. The plot shows three alternative Pareto

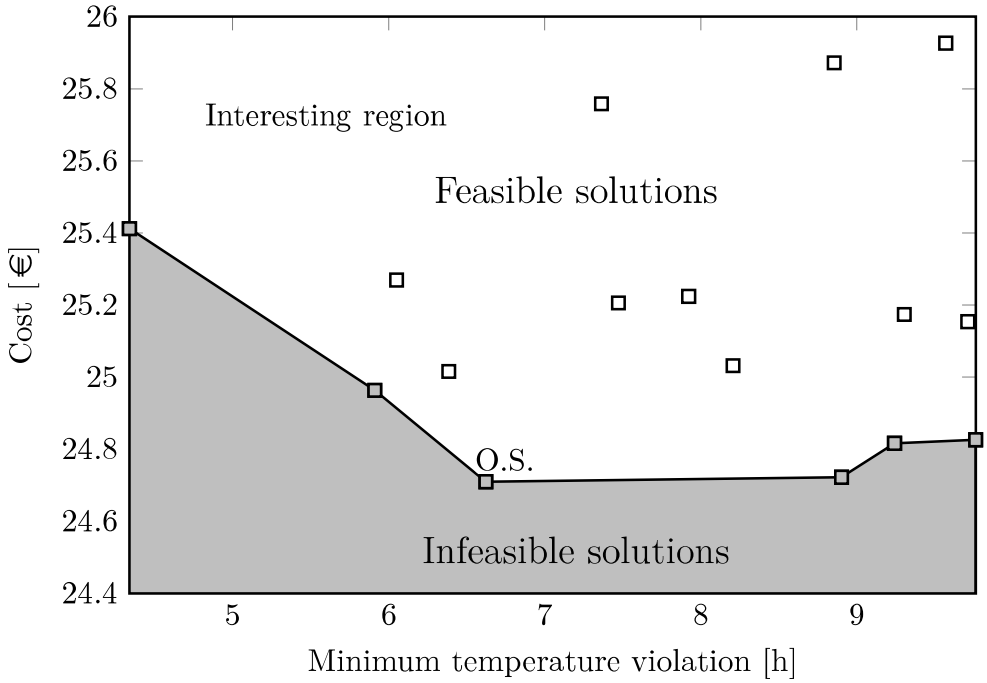

Figure 6.9: Pareto plot of two-phase control cases, where each square represents a heating policy. O.S. signifies the selected optimal solution.

optimal solutions, located in the lower left region. The solutions are all bordering the infeasible region and within the interesting region. From these three the solution with lowest cost was chosen, denoted as the optimal solution (O.S.) in the plot. The chosen policy consisted of heating for six hours starting at 01:00, known as policy 16 from Table B.6.

6.5 Effect of tank size on optimization results

To investigate the effect of tank size all cases were simulated using different tank volumes. In Table 6.2 the cost of each case for varying volumes is listed.

Table 6.2: The effect of tank volume on cost for Case I, Case II & Case III. A back-off of 3 ℃ is used for both cases.

Volume	J Case I	J Case II	J Case III
800	27.3	24.26	23.36
400	27.2	24.39	23.51
250	27.2	24.64	24.28
200	27.19	24.85	24.71
150	27.18	25.22	25.18
100	27.18	25.07	23.69

The absolute cost generally decreases with volume for all but Case I, where the cost increases. It can be seen that this trend does not hold true when tank volume is 100l either, which gives lower costs than when tank volume is 150l. In Fig. 6.10 the savings compared to the cost of Case I is shown for cases II and III.

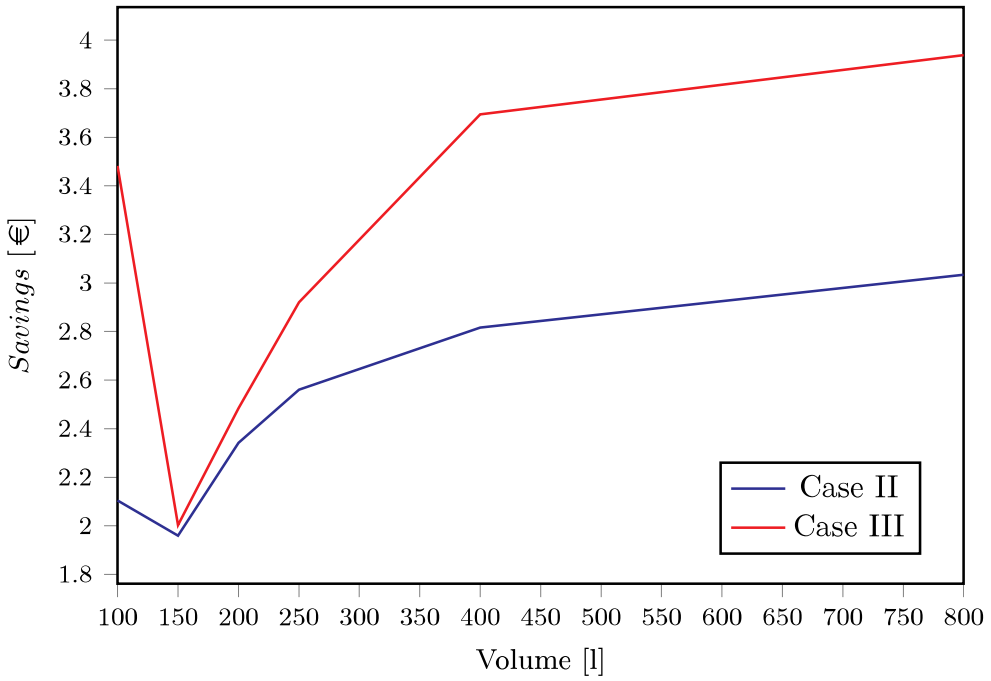

Figure 6.10: Savings for different volumes using case II & III compared to Case I.

The savings for both cases increase with volume, the exception being that savings are higher for a volume of 100l than 150l. This is discussed further in Chapter 7.
The effect of tank size on t_v is shown in Table 6.3.

Table 6.3: The effect of volume on hours spent violating the minimum temperature set point for Case I & Case II. A back-off of 3 °C is used for both cases.

Volume	t_v Case I	t_v Case II	t_v Case III
800	0	0	0
400	0	$4.3 \cdot 10^{-2}$	0.4
250	0	1.22	5.09
200	0	4.56	6.62
150	0.28	8.26	13.37
100	2.94	15.48	21.87

The t_v decreases with volume for all cases. There is no temperature violation for any of the cases with a tank volume of $800l$.

6.6 Case II with known and average demand in predictor

This section shows the difference between predicting the hourly demand based on q_{hw} and yearly average data. The hourly demand based on q_{hw} (known demand) was used in the MPC-solver to find new optimal temperature set points. The results was then compared with results from using yearly average data (yearly demand). This resulted in $J = 24.63$ and $t_v = 4.81$ using known demand, as opposed to $J = 24.85$ and $t_v = 4.56$ for the regular optimized case with yearly demand. A plot showing how the temperature set points and average profiles differ is shown in Fig. 6.11, where the real demand q_{hw} is also included.

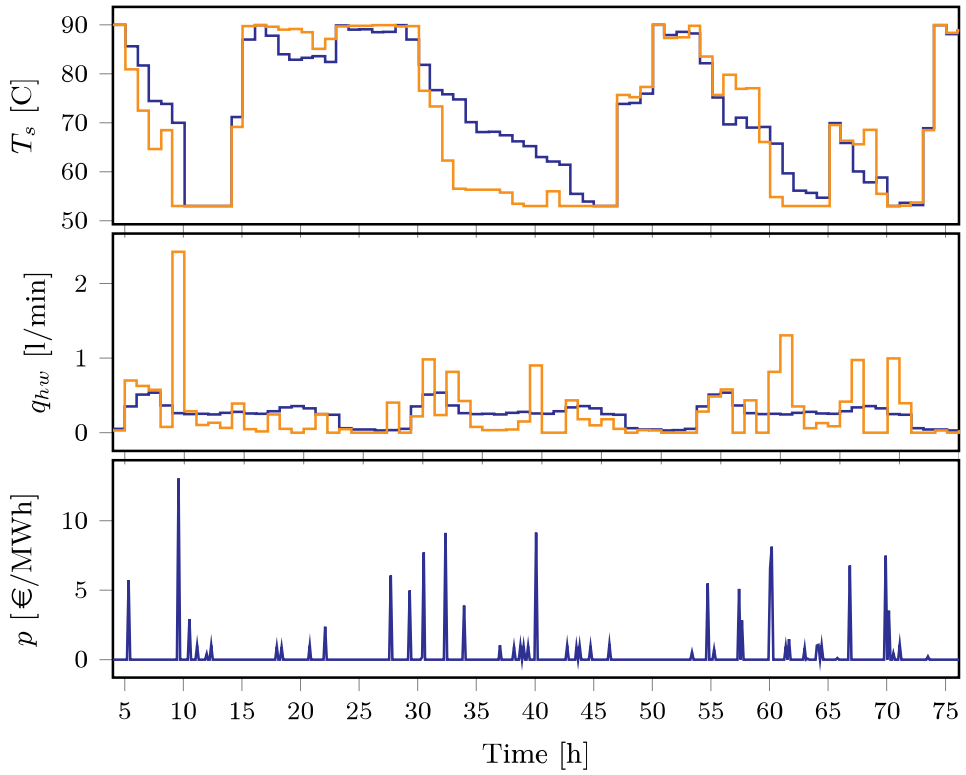

Figure 6.11: Case II temperature set points and average hot water demand. The orange lines represent data for a known average demand while blue lines represent yearly average demand. The real demand is also included.

From the figure it can be seen that both average demands are flatter than q_{hw}, but that the known average demand has more variation than the yearly average.

Chapter 7

Discussion

The goal of this project was to optimize on the hot water tank system and find simplifications that made the problem solvable even with limited computing resources. Balancing an easy computable solution with achieving sufficient accuracy is not simple, as such the majority of this chapter will deal with analyzing the accuracy of the results and the implications each simplification might have had on the solution.

7.1 Evaluation of results

Overall Case II and Case III perform equally well; the slightly lower cost of Case III cancelling out the lower temperature set point violation of Case II. The benefit of Case II is that the solution is generalized and could work with any kind of incoming price data. Also, if Case III came into widespread use, having the hot water heaters all starting to heat at the same time would create a peak of demand and increased price. If this came to pass a solution would be to cycle the start and end points, and use some of the less optimal policies described in Section 6.4. Even though both cases outperformed Case I, the profit of 2.48 € and 2.34 € for one month was not very impressive. The saving of were significantly lower than those found by Johansson (2013), who found savings roughly 3 times greater in terms of percentage. Part of this could be because of the price data used, but the difference seems too large for this to be the only factor.

7.2 Model performance

The model performs reasonably well for most simulated cases, but there are some problems dealing with smaller tank sizes and large spikes in demand. This is due to the models slow disturbance rejection when it comes to temperature. Because of

the simplification made in Section 3.6 that adjusts V_s depending on T, the volume will be kept at maximum until $T \leq T_{hw,s}$. This means that the inflow will not be adjusted until this point, at which the tank will be fully filled and slow to regain temperature. The system therefore has problems with keeping to its prescribed temperature set points.

7.2.1 Violation of temperature set points

Both Case II and Case II has t_v of over four hours. This is certainly more than would be expected by most end users, and means that some compensation would have to be made. As the typical compensation would be money, and the savings are so low that almost no money is saved this almost disqualifies the solution by itself. Decreasing t_v by adding back-up was attempted. But even with significantly improved results. It proved difficult to remove all of the hot water demand. It was discovered that this was mostly caused by an outlier in the demand profile. As changing the demand profile would essentially be "massaging the data" everything was kept as is.
Part of the reason t_v is so large is due to the assumptions made in Section 3.6. The assumption of maximum volume being near optimal for most situations, means that the volume in the tank is at maximum once the temperature falls below the set point. As the volume is as large as possible, reheating the water takes longer time that it would have otherwise.

7.2.2 Back-off

The choice of back-off was made based on the Pareto optimality plot of Case II. The reasoning behind using just Case II to decide the amount of back-off for both Case II and Case III was as follows.

- If the optimal back-off was decided using Case II before the different simple policies had been determined, the choice of back-off would affect the results of the simple policies and the best policies for the given back-off could be chosen.
- If back-off was to be determined based on Case III however. The best two-phase policy would have to be determined before the optimal back-off. The choice of policy would influence the optimal amount of back-off which again would require finding a new optimal policy with the optimal back-off.

The choice of back-off was based on the high increase in cost that would be required to reach reasonable levels of t_v, a choice of 3°C improved t_v substantially without sacrificing much in terms of savings.

7.2.3 Size of the hot water tank

The size of the hot water tank impacted the results as would be expected. Since no heat loss had been assumed, the bigger tanks outperformed the smaller tanks

particularly when it came to disturbance rejection. The savings for the biggest tank ($800l$) was roughly twice as large as for the smallest well-behaved tank ($150l$). Tanks with volume lower than $150l$ did not work as expected. This reason for this is discussed in Section 7.3.2.
For the larger tanks it should be noted that controller tunings might have an effect on the result. Controller tunings are made based on the volume of the tank, so once this changes the tunings are no longer optimal. This is probably why there is any difference at all in results for the different sized tanks for Case I. Somewhat surprising is the fact that Case III clearly outperformed Case II for high tank volumes. This could be due to poor controller tunings affecting Case II more than Case III as more adjustments are required for Case III.

7.3 Assumptions and simplifications

7.3.1 Disturbances and parameters

The two disturbances T_{cw} and $T_{hw,s}$ have been assumed to have constant values during the simulation. Constant temperature of cold water seems reasonable as any change would be very slow. If feed-forward control was used on q_cw however, this might cause problems. Assuming that $T_{hw,s}$ does not change also seems reasonable as long as q_{hw} is mixed with a cold water stream at a later stage. This would allow the user to control the water temperature. If the temperature set point did change, that would not necessarily require big changes to the optimization of the system. The constraint of $T \geq T_{hw,s}$ would have to change to $T \geq T_{hw,s,max}$, while q_{cw} still would be used to achieve $T_{hw} = T_{hw,s}$. Implementing this would require big changes to the program that create the q_{hw} profile though, as the program generates demand for a certain amount of 50 °C hot water.
Since the program generating the q_{hw} profile is not part of the thesis, the accuracy of the profiles it generates will not be thoroughly assessed. But it does seems prudent to make some comments about how the objective of the program differ from the objective of the thesis. The program is intended to generate a random demand profile of a household. It is not intended to generate the demand profile for one specific household. This difference is important as the variation between different households is expected to be greater than the variation from day to day within one household. Because of this the demand profile used in this thesis represents one of a household where every day at midnight the house changes tenants and new people move in. Obviously predicting the demand in such a house would be a lot more difficult than the demand of a regular household. The effect of this might not be that big however. In Section 6.6 it was proven that improving the predicted demand did not majorly improve the cost.
The Price data would definitely affect the results, but the chosen values are not uncommon compared to other months. The assumption of perfect knowledge of the price is discussed at length in 7.3.3. The density of the water in and out of the tank have been assumed constant at 1 kg/l, which is reasonable as the density of liquid water does not vary that much with temperature. Constant heat capacity

of $4.19kJ/kg$,℃ is also unlikely to impact the results.

7.3.2 Perfect control

Assuming perfect control of q_{cw} implies that $T_{hw} = T_{hw,s}$. This assumption is fine for most simulations, though not quite realistic. For simulations with high demand spikes and low tank volume it does give problems. The flow out of the tank q_{out} is adjusted by assuming that $T_{hw} = T_{hw,s}$. Therefore the outflow from the tank will be adjusted to achieve this whether there is water left in the tank or not. This will lead to much higher outflow than inflow and eventually negative volume in the tank. During the simulation this only occurred for tank volumes where the tank volume was $100l$ or less. Negative volumes does make the tank very easy to heat however, which is probably why the cost J decreases for these cases. If the system was implemented in practice, this could be solved by adding some constraint on the the outflow from the tank Q_{out}. Adding extra back-off to correct for dynamic controller error of q_{cw} would also be a good idea.

7.3.3 Roughness of discretization

The optimization problem is solved once every hour, which compared to regular MPC is quite slow. Usually MPC control updates set points on the scale of minutes, so reoptimizing only once per hour might be a bit to simple. The key advantage of only solving once per hour is speed, which is arguably the main objective of our model. Increasing the fineness of the discretization would require solving the problem more often and, assuming uniform time intervals, would make each optimization problem bigger. In retrospect, the disadvantages could have been made up for by using larger time intervals towards the end of the optimization. It also seems like optimizing this rarely has created problems with ensuring that $T \geq T_{hw,s}$ for the optimized case. This is because the predicted demand uses average values to determine optimal set points, which makes it poor at predicting sudden spikes of demand. When these spikes do happen the temperature will often drop below the set point. For Case II with known demand profile using a finer discretization could be useful. This would have led to better prediction of the demand, as spikes that would otherwise come as a surprise to the solver would be noticeable. This should give lower temperature violation.

7.3.4 Moving prediction horizon

It has been assumed that electricity price is known twenty-four hours in advance. In reality that is not entirely true. Electricity price for the coming day is published by Nord Pool Spot (2014) at midnight, but it does not change by the hour to provide data twenty-four hours into the future. One obvious alternative would be to change the MPC's prediction horizon from moving to shrinking, and such have a solver that only uses known price data. This would speed up the simulation because the optimization problem gets shorter for each iteration, but increase errors due to end-effects (the temperature and volume at the beginning and end of each day would

have to be specified). The loss from optimality by specifying the temperature and volume to be at a maximum at 4:00 would probably not be significant as the price tends to always be very low in the preceding hours. If only known price data should be used however, 24:00 would have to be the end point as this is where the price data stops. Some sort of pseudo-optimal solution, using an estimated price for the period from 24:00-4:00 and then updating it once real data was available might have been interesting as well.
A small advantage of using a shrinking horizon would be the possibility to make multiple single day evaluations instead of simulating all the days consecutively. This would make it faster to test multiple days as the processor could run several days in parallel, known as multithreading. The usefulness of this is limited to testing purposes however; If the system was to be implemented it would not be possible or necessary to optimize for more than one day because of lacking price data. Overall using a shrinking prediction horizon seems very promising.

Chapter 8

Conclusion

The objective has been to minimize operational cost while still meeting the hot water demands of the end user. To achieve this, a hot water tank system has been modeled using Simulink and Matlab and a feedback control structure implemented to stabilize the system. By using ideas from self-optimizing control the cost function has been simplified to a form that makes it solvable even with limited computation resources. The simplified problem has then been solved using an MPC-solver. The resulting optimized case (Case II) has been compared with a simple policy of heating for a set amount of hours at night (Case III) and holding a constant temperature in the tank (Case I). The mode works well and gives reasonable results for all cases except those with very small tank volumes. Though major simplifications to the optimization problem has been made Case II still gives decent, but not spectacular results, in terms of savings and making sure the end user always has hot water.

Based on the findings in this thesis, using MPC to find optimal tank energy levels does not yield any significant benefit over using a simple policy of heating the tank during the night. While both cases are improvements compared to having constant tank temperature, the savings of 2.48 € and 2.34 € per month is not high enough to recommend implementation. The proposed policies could still be useful in areas with more pronounced price variation.

8.1 Further work

Several adjustments could be made to the solver of the optimization problem, some that will improve computation time. An adjustment that need not affect computation time is using a non-uniform prediction horizon. It also seems like a major opportunity to simplify the optimization problem was missed by deciding to use a moving horizon.

The set of rules used for finding near optimal volume set points has not been verified by comparing it to other cases. A simple way this could be tested is by adjusting at which temperature the volume set point changes, currently $T = T_{min}$. If the currently implemented "tipping point" gave the best results the set of rules would be verified. If not, better rules could be found.

Bibliography

Cruickshank, C. A., Harrison, S. J., 10 2010. Heat loss characteristics for a typical solar domestic hot water storage. Energy and buildings 42 (10), 1703–1710.

de Oliveira, V., Jäschke, J., Skogestad, S., 2013. Optimal operation of energy storage in buildings; use of hot water systems. Tech. rep., NTNU, Norway, presented at DYCLOPS Mumbai, December 2013.

Doman, L. E., 2013. The international energy outlook 2013. Tech. rep., U.S. Energy Information Administration.

Edenhofer, O., Pichs-Madruga, R., Sokona, Y., Farahani, E., Kadner, S., Seyboth, K., Adler, A., Baum, I., Brunner, S., Eickemeier, P., Kriemann, B., Savolainen, J., Schlömer, S., von Stechow, C., Zwickel, T., Minx, J., 2014. Summary for policymakers, in: Climate change 2014, mitigation of climate change. Tech. rep., The Intergovernmental Panel on Climate Change.

Ericson, T., 2009. Direct load control of residential water heaters. Energy Policy 37 (9), 3502 – 3512.

Foss, B. A., Balchen, J. G., Andresen, T., 2003. Reguleringsteknikk, 5th Edition. Wiley.

Holene, A. L., 6 2013. Performance and robustness of smith predictor control and comparison with pid control. Master's thesis, NTNU.

Jacobsen, M. G., 11 2011. Identifying active constraint regions for optimal operation of process plants. Ph.D. thesis, NTNU.

Johansson, E. M., 12 2013. Optimal operation of energy storage in buildings. Master's thesis, NTNU.

Lerouge, C., Atlan, N., 2013. European energy markets observatory.

Lund, P. T. J., 6 2012. Annual report 2011; The Norwegian Energy Regulator. Tech. Rep. 1, The Norwegian Water Resources and Energy Directorate.

Mayne, D., Rawlings, J., Rao, C., Scokaert, P., 2000. Constrained model predictive control: Stability and optimality. Automatica 36 (6), 789 – 814.

Charles River Associates, 2003. Dm programs for integral energy, final report. Tech. rep.

Nord Pool Spot, June 2014. Elspot prices 2013, hourly in EUR.
URL `http://goo.gl/tVxJS`

Olje- og energidepartementet, 2012. Meld, st. 14 (2011/2012).

Olje- og energidepartementet, 2014. Press release nr. 026/14: Forslag om elektrifisering av utsirahøyden.

Rawlings, J. B., Stewart, B. T., 2008. Coordinating multiple optimization-based controllers: New opportunities and challenges. Journal of Process Control 18 (9), 839 – 845.

Ruscio, D. D., 2009. System theory state space analysis and control theory. Tech. rep., Høgskolen i Telemark.

Seborg, D. E., Edgar, T. F., Mellichamp, D. A., 9 2003. Process Dynamics & Control, 2nd Edition. Wiley.

Shamsuzzoha, M., 10 2013. Simple analytic rules for model reduction and pid controller design. Industrial & Engineering Chemistry Research (52), 12973–12992.

Skogestad, S., 2003. Simple analytic rules for model reduction and pid controller design. Journal of Process Control (13), 291–309.

Skogestad, S., 2004. Near-optimal operation by self-optimizing control: from process control to marathon running and business systems. Computers and Chemical Engineering (29), 127–137.

Skogestad, S., 3 2014. Control structure selection. Chapter for Springer Encyclopedia on Control Systems.

Appendices

Appendix A

In-depth Derivations

A.1 Deriving $\frac{\partial T}{\partial t}$ from the energy balance

The generalized energy balance for a tank is shown in Eq. A.1.

$$\frac{dE}{dt} = \dot{E}_{in} - \dot{E}_{out} + Q_{tot} + \dot{W}_s + \dot{W}_b + \dot{W}_f \tag{A.1}$$

Where E is the total energy inside the control volume, $\dot{E}$s are energy streams and Q_{tot} is the heat added to the system. The work terms W_s, W_b and W_f depend on shaft work, pressure difference and expansion of the control volume respectively. Shaft work is negligible and as both the volume and pressure is constant all the work terms can be ignored. The total energy E consists of several smaller energy terms, of which most can be ignored for this case. The expression is reduced to

$$\begin{aligned} E &= U + U_k + U_p + ..., \\ E &\approx U, \\ U &= H + \int PdV + \int VdP, \\ U &= H, \\ E &\approx H. \end{aligned} \tag{A.2}$$

Where U is the internal energy, U_k the kinetic energy, U_p the potential energy and P is the pressure in the tank. Inserting this into Eq. A.1 gives Eq. A.3.

$$\frac{dH}{dt} = \dot{H}_{in} - \dot{H}_{out} + Q_{tot} \tag{A.3}$$

Which equals Eq. 3.5 if the Q_{tot} term is expanded. The enthalpies in Eq.A.3 can be expressed as functions of temperature

$$\begin{aligned} H &= \rho V c_p (T - T_{ref}), \\ \dot{H}_{in} &= \rho_{in} q_{in} c_p (T_{cw} - T_{ref}), \\ \dot{H}_{out} &= \rho_{out} q_{out} c_p (T - T_{ref}). \end{aligned} \tag{A.4}$$

Assuming that the temperature and volume are the only non-constant variables, inserting Eq. A.4 into the left hand side (LHS) of Eq. A.3 gives Eq. A.5.

$$\begin{aligned} (LHS) &= \frac{dH}{dt} \\ &= \frac{d(\rho c_p V (T - T_{ref}))}{dt} \\ &= \rho c_p \left(\frac{d(VT)}{dt} - \frac{dV}{dt} T_{ref} \right) \\ &= \rho c_p \left(\frac{dV}{dt} (T - T_{ref}) + \frac{dT}{dt} V \right) \\ &= \rho c_p \left(q_{in} (T - T_{ref}) - q_{out} (T - T_{ref}) + \frac{dT}{dt} V \right) \end{aligned} \tag{A.5}$$

Here the expression for $\frac{dV}{dt}$ from Eq. 3.6 has been used. Inserting Eq. A.4 into the right hand side (RHS) of Eq. A.3 gives

$$\begin{aligned} (\text{RHS}) &= \dot{H}_{in} - \dot{H}_{out} + Q - Q_{loss}, \\ &= \rho c_p \left(q_{in} (T_{cw} - T_{ref}) - q_{out} (T - T_{ref}) \right) + Q - Q_{loss}. \end{aligned} \tag{A.6}$$

The expressions for the (LHS) and (RHS) are then inserted back into Eq. A.3. Which gives Eq. A.7 as the q_{out} terms cancel each other out.

$$\begin{aligned} (\text{LHS}) &= (\text{RHS}) \\ \frac{dH}{dt} &= \dot{H}_{in} - \dot{H}_{out} + Q - Q_{loss} \\ \rho c_p \left(q_{in} (T - T_{ref}) + \frac{dT}{dt} V \right) &= \rho c_p \left(q_{in} (T_{cw} - T_{ref}) \right) + Q - Q_{loss} \end{aligned} \tag{A.7}$$

From this point on, only moving terms around is needed to yield the final temperature balance

$$\begin{aligned} \frac{dT}{dt} V + q_{in} (T - T_{ref}) &= q_{in} (T_{cw} - T_{ref}) + \frac{Q - Q_{loss}}{\rho c_p}, \\ \frac{dT}{dt} &= \frac{1}{V} \left(q_{in} (T_{cw} - T) + \frac{Q - Q_{loss}}{\rho c_p} \right). \end{aligned} \tag{A.8}$$

A.2 Finding the flow out of the tank, q_{out}

From Fig. 3.1 it is apparent that the mass and temperature balances over the mixing point of q_{cw} and q_{out} are

$$q_{hw} = q_{cw} + q_{out}, \tag{A.9}$$

$$T_{hw} = \frac{q_{cw}}{q_{cw} + q_{out}} T_{cw} + \frac{q_{out}}{q_{cw} + q_{out}} T. \tag{A.10}$$

By assuming that $T_{hw} = T_{hw,s}$, there are only two unknown variables left: q_{cw} and q_{out}. Solving Eq. A.10 for q_{cw} and inserting the resulting expression into Eq. A.9 gives an expression for q_{out}.

$$T_{hw,s} q_{cw} + T_{hw,s} q_{out} = q_{cw} T_{cw} + q_{out} T \tag{A.11}$$

$$q_{cw}(T_{hw,s} - T_{cw}) = q_{out}(T - T_{hw,s}) \tag{A.12}$$

$$q_{cw} = q_{out} \frac{T - T_{hw,s}}{T_{hw,s} - T_{cw}} \tag{A.13}$$

Inserting q_{cw} into A.9 gives

$$q_{out}(1 + \frac{T - T_{hw,s}}{T_{hw,s} - T_{cw}}) = q_{hw}, \tag{A.14}$$

$$q_{out}(\frac{T - T_{cw}}{T_{hw,s} - T_{cw}}) = q_{hw}, \tag{A.15}$$

$$q_{out} = q_{hw} \frac{T_{hw,s} - T_{cw}}{T - T_{cw}}. \tag{A.16}$$

Appendix B

Tables

This appendix contains tables that were deemed to large to insert into the main report. It also contains all the values for parameters, constrains, disturbances and initial conditions described in Chapter 3 and implemented in Chapter 5.

B.1 Model data

Tuning and model parameters are shown in tables B.1 and B.2. The disturbances and constraints are shown in tables B.3 and B.4, respectively. Initial and final conditions for Case II is shown in Table B.5.

Table B.1: Tuning parameters.

Tuning parameters	Explanation	Value
K_{c1}	Temperature controller gain	3
τ_{i1}	Temperature controller integral time	80
K_{c2}	Volume controller gain	0.21
τ_{i2}	Volume controller integral time	80

Table B.2: Model parameters.

Parameter	Explanation	Value	Units
Q_{loss}	Heat loss from the tank	0	kW
ρ	Density of water	1	kg/l
c_p	Heat capacity of water	4.19	kJ/kg,°C

Table B.3: Disturbance values.

Disturbances	Explanation	Value	Units
T_{cw}	Temperature of cold water	5	℃
$T_{hw,s}$	Hot water temperature set point	50	℃
p	Electricity price	varies	€/MWh
q_{hw}	Hot water demand	varies	l/min

Table B.4: Input and output constraints.

Constraints	Explanation	Value	Units
T_{min}	Minimum temperature in the tank	50	℃
T_{max}	Maximum temperature in the tank	90	℃
V_{min}	Minimum water volume in the tank	50	l
V_{max}	Maximum water volume in the tank	200	l
$q_{in,min}$	Minimum volumetric inlet flow	0	l/min
$q_{in,max}$	Maximum volumetric inlet flow	10	l/min
Q_{min}	Minimum heater duty	0	kW
Q_{max}	Maximum heater duty	5	kW

Table B.5: Initial and final conditions.

Initial conditions	Explanation	Value	Units
V_0	Volume at the start of simulation	200	l
T_0	Temperature at the start of simulation	90	℃
V_{end}	Volume at the end of simulation	200	l
T_{end}	Temperature at the end of simulation	90	℃

B.2 Additional results

Table B.6 shows the results from simulating the different two-phase policies in Section 6.4. Length specifies the length of the heating phase and starting time the time of day the heating phase starts.

Table B.6: Comparison of two-phase policies.

Policy	Starting time	Length	Temperature violation	Cost
1	22	3	9.57	25.93
2	23	3	10.93	25.38
3	0	3	9.3	25.17
4	1	3	9.76	24.83
5	22	4	9.8	25.93
6	23	4	10.93	25.37
7	0	4	9.71	25.15
8	1	4	9.24	24.82
9	22	5	10.48	25.91
10	23	5	10.75	25.35
11	0	5	9.97	25.14
12	1	5	8.9	24.72
13	22	6	9.98	25.89
14	23	6	10.63	25.34
15	0	6	8.21	25.03
16	1	6	6.62	24.71
17	22	7	8.85	25.87
18	23	7	7.92	25.22
19	0	7	6.39	25.02
20	1	7	5.91	24.96
21	22	8	7.36	25.76
22	23	8	7.47	25.21
23	0	8	6.05	25.27
24	1	8	4.34	25.41

Appendix C

Main MATLAB Scripts

This Appendix contains the scripts that define the initial conditions and run the SIMULINK simulation. It contains three scripts, `sizing.m`, `simplec.m` and `backoff.m`, and the functions `main.m` and `system.m`.
`main.m` contains the majority of coding and is where the simulation is initialized. It takes input arguments from the three scripts, and will be called repeatedly with different arguments. In the case of `sizing.m` the changing input argument is the maximum volume of the tank, for `simplec.m` different two-phase policies and for `backoff.m` different amounts of back-off. The two-phase policies are received as a binary lists, which instructs the controllers which hours to keep the temperature in the tank at maximum (heating phase) and which hours to keep it as low as possible (conserving phase).
`system.m` is the coding behind the block [system] in the SIMULINK model. This is were differential equations for the states are calculated and the new system state found.

C.1 Running the SIMULINK model: `main.m`

This is the main function, it initializes SIMULINK variables and calls the model. The function has five arguments, all being optional: `inmode`, `involume`, `inplot`, `inmatrix` and `inbo`. The `inplot` variable controls whether or not to plot the results in MATLAB, while the rest of the arguments impact the results of the function. `inmode` controls which case to run, the options being: Case I, Case, II and Case II which correspond to `inmode` having a value of 1, 2 and 3 respectively. As previously stated, the main function is generally called repeatedly by scripts with varying arguments. The varying arguments are `involume`, `inmatrix`and `inbo`. which modifies the maximum volume of the tank, the two-phase policy or the back-off.

`main.m` is a long function as it contains three different cases and each case has optional plotting and storage of data to .mat files. The first part of the function

is focused on initializing the variables that are to be used by SIMULINK. After initializing the variables the program splits to deal with the three different cases. These are separated using a switch statement. The program returns two variables, the cost `J_end`, and the temperature violation `T_violation`.

Listing C.1: Initializing and running the SIMULINK model.

```
function [T_violation,J_end] = main(inmode,involume,inplot,inmatrix,inbo)
% main.m runs the simulink model hwtank.mdl with different temperature
% set points and starting conditions.
%
%Function arguments:
% inmode:   Which case to simulate:{1 = constant set point case,
%                                   2 = optimized case,
%                                   3 = two-phase case}
% involume:  The maximum volume of water contained in the tank
% inplot:    Whether to plot state profiles (1) or not(0)
% inmatrix:  Contains the two phase policy, only matters if mode = 3
% inbo:      The back-off, (back off = set point-constraint) defines
%            how far away from the constraint the smallest set points are
%
%Function output:
% T_violation:  The number of hours that T<T_hws in the simulation
% J_end:        The total cost at the end of simulation
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

simdays = 3; %controls the amount of days to simulate for.

%Default arguments:
%Reads the function arguments. If not specified the defualt args are used
mode = 1; V_max = 200; ploton=0; backoff=0;
rulem=[1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1];
if nargin > 0, mode = inmode;end
if nargin > 1, V_max = involume;end
if nargin > 2, ploton = inplot;end
if nargin > 3
    if isempty(inmatrix)
                % if the matrix is empty the default one should be used
    else
        rulem = inmatrix;     % otherwise use the incoming matrix instead
    end
end;
if nargin > 4, backoff = inbo;end

warning off
%clc
%clear all
close all

%% Initializing
global par
par.c_p  = 4.19;                      % heat capacity of water [kJ/(kgC)]
par.rho  = 1;                            % density of water [kg/l]
par.Q_loss = 0;                        % loss of heat from the tank
```

```
%disturbances and set point limitations
T_hws  = 50;      % Temperature wanted on the hot water reaching the user
T_cw   = 5;                                % Temperature of the cold water
T_smax = 90;                              % Maximum tepmperature set point
T_smin = T_hws+backoff;           % adjusts minimum temperature set point
V_min  = 50;                                    % minimum water in the tank
%V_max is defined on line 24

totfailures = [];
Ekfailures = [];
Eendy= [];

%simulation parameters (initial conditions)
T_s = T_smax;        %For base case
par.V = V_max;       %volume starting point
par.T = T_smax;      %Temperature starting point

%Controller tunings
Kc1  = 3;    % gain for controller of volume
Kc2  = 0.21; % gain for controller of temperature
tau1 = 1/80; % Really 80 but simulink uses tuning paramer I=1/tau_i in its
tau2 = 1/80; % PI controllers.
%input limitations
q_in_min = 0; % dm^3/min
q_in_max = 10;% dm^3/min
Q_min    = 0; % kJ/s
Q_max    = 5; % kJ/s

%Generating the demand/price and the setpoints:
%
% In general we prefer to load a pregenerated profiles (longprofile)
% to ensure that we get comparable results for all experiments.
% For the hot water setpoint T_hws and cold water temperature T_cw a
% constant value is wanted. These are created using the "ext" variable.
% The electricity price are loaded from a price.csv file.
%
test=load('langprofil.mat');
a = test.testlong;   %contains the pregrnerated demand and price profiles
profile.time=(a.time'+0.167)*3600;
ext = ones(length(a.time),1);    % used to extend the constant set points

%Generating variable containing all disturbances for the given period:
profile.signals.values =  [a.flow, T_hws.*ext, T_cw.*ext, a.price];

simlength = 24*simdays;                % number of hours to simulate for

% Load pregenerated average energy demand and average price profiles,
% this is just used in case II:
av.demand = csvread('average.csv',0,0,[0,0,23,0]);       % q_hw, dm^3/min
predictedprice = getPriceHour(4,1,4,31);    % 31 because we need the full
                    % prediction horizon at the end of the 30th day too
switch mode
    case 1                        % CONSTANT TEMPERATURE CASE, T_s = T_smax
        T_s =T_smax;                              % constant max set point

        % Define the matrices used to store state variables
```

```
        ptime_base=[]; %changing time from seconds to hours.
        sp_base=[]; % extracting setpoints
        y_base=[]; % T, V and modified cost fx
        d_base=[]; % disturbances
        u_base=[]; % inputs

        mdl = 'hwtank.mdl';
        load_system(mdl);
        set_param('hwtank/Temp_s', 'Value', num2str(T_s));
        out1 = sim(mdl,'SrcWorkspace','current','StopTime', ...
            (num2str(3600*simlength)), 'SaveFinalState', 'on', ...
            'LoadInitialState', 'off', 'SaveCompleteFinalSimState', ...
            'on','FinalStateName', 'xFinal');

        %update the state variables
        ptime_base=[out1.get('time')/3600]; %time from seconds to hours.
        sp_base=[out1.get('sp')]; %extracting setpoints
        y_base=[out1.get('y')]; % T, V and modified cost fx
        d_base=[out1.get('d')]; % disturbances
        u_base=[out1.get('u')]; % inputs

        if ploton
            figure(3)
            subplot(311)
            plot(ptime_base,d_base(:,1))
            set(gca,'FontSize',14)
            title('Hot water demand')
            ylabel('q_o_u_t, dm^3/min')
            hold on

            subplot(312)
            plot(ptime_base,u_base(:,2),'red')
            set(gca,'FontSize',14)
            ylabel('q_i_n, dm^3/min')
            hold on

            subplot(313)
            plot(ptime_base,y_base(:,1))
            set(gca,'FontSize',14)
            ylabel('V, dm^3')
        end

        % If a 30 day simulation, save the data points to .mat file
        if simdays == 30 && backoff
            if exist(['_datasetsV',num2str(V_max),'.mat'],'file')
                save(['_datasetsV',num2str(V_max)],'ptime_base',...
                'sp_base','y_base','d_base','u_base','-append');
            else
                 save(['_datasetsV',num2str(V_max)],'ptime_base',...
                'sp_base','y_base','d_base','u_base');
            end
        end

        % Variables for calculating T_violation and J_end
        listlength= length(y_base(:,2));
        yvar = y_base;
    case 2        % OPTIMIZED CASE, T_s = changing according to predictor
```

```
%Bounds
Emax = par.rho*par.c_p*V_max*(T_smax - T_cw) %kJs
Emin = par.rho*par.c_p*V_min*(T_hws - T_cw); %kJ
E0   = Emax; % intial condition
Qmax = Q_max;


%plotting
if ploton
    figure(1);   % start the MPC-plot (must be done outside loop)
end
%Make lists to store state variables
ptime_optim=[]; %changing time from seconds to hours.
sp_optim=[]; %extracting setpoints
y_optim=[]; % T, V and modified cost fx
d_optim=[]; % disturbances
u_optim=[]; % inputs


%% iteration
for i=1:simlength
    Ns=24; % size of horizon (in sample times)
    j = i;
    while j>24
        j= j-24
    end
    % shifting the average demand and price function
    Qdemand = [av.demand(j:Ns);av.demand(1:j-1)]*(par.rho...
              *par.c_p*(T_hws -T_cw))/60 ;%  kJ/s
    prices  = [predictedprice(i:Ns+(i-1))];
    %startposition for both price and demand depends on time of day
    dT      = 3600;  % sampling time of 1hour

    %% formulating the optimization problem
    constraints=[]; % vector containing all constraints
    Qk=sdpvar(Ns,1); % decision variables as a vector of size Ns
    Tri=tril(ones(Ns,Ns),0);
    %state trajectory: Equivalent to computing Ek+1=Ek+(Qk-Qdk)*dT
    %k=0...Ns
    Ek=Tri*(Qk*dT-Qdemand*dT)+E0;
    %update the constraints
    constraints=constraints+set(Emin<=Ek<=Emax);
    constraints=constraints+set(Ek(end)==Emax);
    constraints=constraints+set(0<=Qk<=Qmax);

    cost=prices'*Qk*dT;  % cost function, what we want to opt!!

    %% solve optimization
    opts = sdpsettings('solver','sedumi','sedumi.numtol',1e-3);
    opts.verbose = 0;
    tic
    sol=solvesdp(constraints,cost,opts) %solving the "MPC"
    toc
    if sol.problem == 0
        %filler, don't need to do anything if it works
    else % stop if the solver doesnt find a solution
```

```
        display('Hmm, something went wrong!');
        sol.info
        yalmiperror(sol.problem)
    end
    cost=double(cost) %the results from the optimization
    Qv=double(Qk);
    Ev=double(Ek);

    %%Transform from energy set points to temperature set point
    Ts=double(Ev/(par.rho*par.c_p*par.V)+ T_cw);
    %Modify result if reccomended set points breaks constraints
    if Ts(1) > T_smax
        T_s = T_smax
    elseif Ts(1)< T_smin
        T_s = T_smin
    else
        T_s = Ts(1)
    end
    Volume = par.V % for bug testing purposes

    %% Simulate with set point
    mdl = 'hwtank.mdl';
    load_system(mdl);
    if i == 1
        %Simulate until end of step. Save state.
        set_param('hwtank/Temp_s', 'Value', num2str(T_s));
        out3 = sim(mdl,'SrcWorkspace','current','StopTime', ...
            (num2str(dT*(i+4))), 'SaveFinalState', 'on', ...
            'LoadInitialState', 'off', ...
            'SaveCompleteFinalSimState', 'on',...
            'FinalStateName', 'xFinal');
        y_step = out3.get('y'); % the y variable (T,V and cost)

    else
        %Load SimState and run until end of step.
        %The start time value must remain unchanged.
        set_param('hwtank/Temp_s', 'Value', num2str(T_s));
        xFinal = out3.get('xFinal');
        out3= sim(mdl, 'SrcWorkspace','current', 'StopTime', ...
            num2str(dT*(i+4)), 'SaveFinalState', 'on', ...
            'LoadInitialState', 'on', 'InitialState','xFinal',...
            'SaveCompleteFinalSimState', 'on',...
            'FinalStateName', 'xFinal');
        y_step = out3.get('y'); % the y variable (T,V and cost)
    end
    % extracting initial conditions for next iter.
    Tnew = y_step(end,2);
    Vnew = y_step(end,1);
    Enew = Vnew*par.rho*par.c_p*(Tnew -T_cw);
    E0   = Enew;
    par.V    = Vnew;

    % scale, so Ek stays within the interval [0 1]
    Ev=(Ev-Emin)/(Emax-Emin)
    if ploton
        subplot(411)
        stairs((1+(i-1):Ns+(i-1)),Ev)
```

```
                xlabel('Time,h')
                ylabel('Energy level')
                hold on
                subplot(412)
                stairs((1:Ns),Qv)
                ylabel('Heat power')
                xlabel('Time,h')

                subplot(413)
                stairs((1+i:Ns+i),prices,'Color',[i/simlength 0 1])
                xlabel('Time,h')
                ylabel('Energy prices')
                hold on
                subplot(414)
                stairs((1+(i-2):Ns+(i-2)),Ts,'Color',[i/simlength 0 1])
                xlabel('Time,h')
                ylabel('Teperature setpoint')
                hold on
            end
            %changing time from seconds to hours.
            ptime_optim=[ptime_optim;(out3.get('time')/3600)];
            sp_optim=[sp_optim;out3.get('sp')]; %extracting setpoints
            y_optim=[y_optim;out3.get('y')]; % T, V and modified cost fx
            d_optim=[d_optim;out3.get('d')]; % disturbances
            u_optim=[u_optim;out3.get('u')]; % inputs

        end

        if ploton
            figure(2)
            subplot(611)
            plot(ptime_optim, sp_optim(:,1),'green',...
                 ptime_optim, y_optim(:,1))
            set(gca,'FontSize',14)
            title('Model implementation ')
            ylabel('V, dm^3')

            subplot(612)
            plot(ptime_optim,u_optim(:,2),'red')
            set(gca,'FontSize',14)
            ylabel('q_i_n, dm^3/min')
            hold on

            subplot(613)
            plot(ptime_optim, sp_optim(:,2),'green', ...
                 ptime_optim,y_optim(:,2))
            set(gca,'FontSize',14)
            ylabel('T, ^oC')
            hold on

            subplot(614)
            plot(ptime_optim,u_optim(:,1),'red')
            set(gca,'FontSize',14)
            ylabel('Q, kW')
            hold on

            subplot(615)
```

```
            plot(ptime_optim,d_optim(:,4))
            set(gca,'FontSize',14)
            ylabel('P, EUR/kWh')
            hold on

            subplot(616)
            plot(ptime_optim,y_optim(:,3))
            set(gca,'FontSize',14)
            ylabel('J, EUR')
            % hold on

            figure(3)
            subplot(311)
            plot(ptime_optim,d_optim(:,1))
            set(gca,'FontSize',14)
            title('Hot water demand')
            ylabel('q_o_u_t, dm^3/min')
            hold on

            subplot(312)
            plot(ptime_optim,u_optim(:,2),'red')
            set(gca,'FontSize',14)
            ylabel('q_i_n, dm^3/min')
            hold on

            subplot(313)
            plot(ptime_optim,y_optim(:,1))
            set(gca,'FontSize',14)
            ylabel('V, dm^3')

        end
        if simdays == 30 && backoff
            if exist(['_datasetsV',num2str(V_max),'.mat'],'file')
                save(['_datasetsV',num2str(V_max)],'ptime_optim',...
                'sp_optim','y_optim','d_optim','u_optim','-append');
            else
                 save(['_datasetsV',num2str(V_max)],'ptime_optim',...
                'sp_optim','y_optim','d_optim','u_optim');
            end
        end
        T_s = 90; % return simulink file to original condition
        set_param('hwtank/Temp_s', 'Value', num2str(T_s));

        %%Calculating the final cost and temperature violation
        listlength= length(y_optim(:,2)); %length of dataset
        yvar=y_optim;

    case 3                      %%Two phase case, T_s = either max or min

        %plotting
        ptime_sim=[]; %changing time from seconds to hours.
        sp_sim=[]; %extracting setpoints
        y_sim=[]; % T, V and modified cost fx
        d_sim=[]; % disturbances
        u_sim=[]; % inputs
```

```
%% iteration
for i=1:simlength
    Ns=24; % size of horizon (in sample times)
    j = i; % the hour of the day
    while j>24
        j= j-24;
    end
    if rulem(j)==1              % determening the course of action
        T_s= T_smax;
    else
        T_s = T_smin;
    end
    %Running the Simulink model
    mdl = 'hwtank.mdl';
    load_system(mdl);
    if i == 1 % if the very first step of simulation..
        %Simulate until end of step and save the final state:
        set_param('hwtank/Temp_s', 'Value', num2str(T_s));
        out3 = sim(mdl,'SrcWorkspace','current','StopTime', ...
            (num2str((i+4)*3600)), 'SaveFinalState', 'on', ...
            'LoadInitialState', 'off', ...
            'SaveCompleteFinalSimState', 'on',...
            'FinalStateName', 'xFinal');
    else
        %Load the SimState and run for an additional step.
        %The start time value must remain constant
        set_param('hwtank/Temp_s', 'Value', num2str(T_s));
        xFinal = out3.get('xFinal');
        out3 = sim(mdl, 'SrcWorkspace','current','StopTime', ...
            num2str((i+4)*3600), 'SaveFinalState', 'on', ...
            'LoadInitialState', 'on', 'InitialState','xFinal',...
            'SaveCompleteFinalSimState', 'on',...
            'FinalStateName', 'xFinal');
    end
    %update the state variables after each simulation
    ptime_sim=[ptime_sim;(out3.get('time')/3600)]; % sec to hours
    sp_sim=[sp_sim;out3.get('sp')];        % extracting setpoints
    y_sim=[y_sim;out3.get('y')];      % T, V and modified cost fx
    d_sim=[d_sim;out3.get('d')];                  % disturbances
    u_sim=[u_sim;out3.get('u')];                        % inputs

end

% variables for calculating T_violation and J_end
listlength= length(y_sim(:,2));
yvar=y_sim;

%saving if serious
if simdays == 30 && backoff && isempty(inmatrix)
    if exist(['_datasetsV',num2str(V_max),'.mat'],'file')
        save(['_datasetsV',num2str(V_max)],'ptime_sim',...
        'sp_sim','y_sim','d_sim','u_sim','-append');
    else
         save(['_datasetsV',num2str(V_max)],'ptime_sim',...
        'sp_sim','y_sim','d_sim','u_sim');
    end
end
```

```
    otherwise % you tried to run a case that didn't exist
        error(['Mode = ',num2str(mode),...
        '. The only implemented modes are 1,2 and 3.'])

end

%%calculating the temperature violation and total cost.
n = 0;
for k=1:listlength
    if yvar(k,2) < 50
        n=n+1;
    end
end
T_violation= (n/listlength)*simlength                % hours of violation
J_end = yvar(end,3)                                                  % EUR

T_s = T_smax;                % return simulink file to original condition
set_param('hwtank/Temp_s', 'Value', num2str(T_s));
end
```

C.2 Two-phase policies: `simplec.m`

This script generates several different two-phase policies and then runs `main.m` for each one. The resulting cost and violation of temperature set point is stored in a .mat file. The policies are represented by 1×24-arrays containing zeros and ones. The integers in the arrays are interpreted as hourly temperature set points, with zero signifying $T_s = T_{s,min}$ and one signifying $T_s = T_{s,max}$.

Listing C.2: Comparing the performance of different two-phase policies.

```
%%%%%%%% Testing different simple control rules %%%%%%%%%%%%%%%%%%%%%%%%%
% runs main.m with different simple control structures. The control
% structures all have two phases, the heating phase and the conservation
% phase. In the heating phase the temperature set point is maximized,
% in the conservation phase the temperature set point is minimized.
% The different control structures differ by when the heating phase starts
% and for how long it lasts.
%Key variables:
% t,        how many hours to have max tempemrature set point
% s-1,      which hour to start maximizing the temperature set point
% emlist,   the binary vector that contains 24-hour temp set points
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
bo=0;
rmat = [];                                              % the results matrix
for t=3:8                           % number of hours spent in heating phase
    for s=[23,24,1,2]                   % hour to start increasing T_s (+1!)
        emlist=zeros(1,24);                    % defining the emlist vector
        t_temp = t;             % counting var. hours left of heating phase
        if s>22           % if heating start is before midnight. (24-1=23)
            if s==23
                emlist(23) = 1;     % modify list to reflect heating phase
```

```
                emlist(24) = 1;
                t_temp = t_temp-2;                          % update count
            else
                emlist(24) = 1;
                t_temp = t_temp -1
            end
        end
        if s>1 && s<22       % adjusting start and endpoints of while loop
            startp=s-1;
            t_temp=t_temp+(s-1);
        else
            startp= 0;
        end;
        while t_temp> startp   % If still hours left of phase: modify list
            emlist(t_temp) = 1;
            t_temp = t_temp-1;
        end
        emlist = [emlist(5:end),emlist(1:4)];    % start profile at 4.00AM
        [T_violation,J_end]=main(3,200,0,emlist,bo);%find T_viol and J_end
        rmat = [rmat;[T_violation,J_end,s-1,t,emlist]];%save result to mat
    end
end

% Naming of the saved file
if bo >0
    name = ['cstructures','BO',num2str(bo)];
else
    name = ['cstructures1'];
end
save(name,'rmat')                                % save restult to .mat file
```

C.3 Comparing tank sizes: `sizing.m`

This script runs calls the `main.m` function for several different volumes and modes. The resulting J_{tot} and t_v for each mode and volume are stored in a .mat file.

Listing C.3: Running the SIMULINK model for different tank sizes.

```
%%%%%% Testing different tank sizes %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This program runs main.m with different volumes and modes, then stores
% the resulting minimum temperature-violation and cost in a .mat-file.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

bo = 0;                                                        % back-off
vlist = [800,400,250,200,150,100,60]';           % Volumes to be compared
modes = [1,2,3];                                   % Cases to be compared
Tresult = zeros(length(vlist),max(modes));          % empty result matrix
Jresult = zeros(length(vlist),max(modes));
for i=1:length(vlist)               % run main.m for all volumes and modes
    for j=modes
        [Tresult(i,j),Jresult(i,j)] = main(j,vlist(i),1,[],bo);
    end
end
```

```
%% naming the saved file
if bo >0
    addon = ['_BO',num2str(bo)];
else
    addon = '';
end;
temp = num2str(modes);                              % turn modes into list
name = ['_TJ_data_V',num2str(min(vlist)),'-', num2str(max(vlist)), ...
        '_m',temp(1:3:end),addon];
save(name,'Tresult','Jresult','vlist')        % save restult to .mat file
```

C.4 Finding optimal back-off: `backoff.m`

This script is built based on `sizing.m` and works the same way. The only major difference is that here volume is kept constant while different back-offs are tested for each mode. To save simulation time Case I is left out of the for-loop. This is because Case I is unaffected by back-off, and would yield the same result for each iteration.

Listing C.4: Simulating cases with different amount of back-off.

```
%%%%%% Testing different amounts of back-off %%%%%%%%%%%%%%%%%%%%%%%%%%%
% This program runs main.m with different back-off and stores the
% resulting T violation and J.
% The program is based on simzing.m so some of the coding is suboptimal.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
volume = 200;
boList = [9,7,5,3,1,0]';                              % Cases to be compared
Tresult = zeros(length(boList),3);
Jresult = zeros(length(boList),3);

%               The base case is unaffected by back-off. We don't want to
[var1,var2]= main(1,volume,0,[],boList(1)); %     ..repeat the simulation
for i=1:length(boList)
    Tresult(i,1) = var1;             % assigning the precalculated T_viol
    Jresult(i,1) = var2;                                    % ditto for cost
    for j=[2,3]        % for the other cases, run main with all back-offs
        [Tresult(i,j),Jresult(i,j)] = main(j,volume,0,[],boList(i));
    end
end

%% naming the saved file
temp  = num2str(modes);  % lists are turned to strings with spaces first,
temp2 = num2str(boList'); % ..then the spaces are ignored using (1:3:end)
name = ['_TJ_data_V',num2str(200),'_m',temp(1:3:end),'_BO',...
        temp2(1:3:end)];
save(name,'Tresult','Jresult','boList')       % save restult to .mat file
```

C.5 Calculating the new states: `system.m`

This program contains the coding behind the block [system] in the SIMULINK model. This is were differential equations for the states are calculated and the new system state found, but it is also were the initial conditions is implemented into the SIMULINK model. `system.m` is an s-function, depending on the flag it receives from SIMULINK it executes a different procedure.

Listing C.5: Initializing the problem and calculating the new states.

```
function  [sys,x0,str,ts] = system(t,x,u,flag)
% This S-function is based on a genral example written by Vinicius de
% Oliveira.
% S-functions store all the information behind a simulink block and
% therefore behaves differently appearing on the flag sent from
% Simulink. At first the function will be called to initialize the number
% of outputs/inputs/states that the block should have.(flag = 0)
% Then Simulink will call the function with different flags to find the
% derivatives and the updataed states. (flag = 1 and 3 respectively)
switch flag

    % Initialization

    case 0
        global par    % Load par for updated initial conditions
        sys = [3,     % Number of continuous states
            0,        % Number of discrete states
            3,        % Number of outputs--NUMBER OF CVS
            6,        % Number of inputs (to the block)
            0,        % reserved must be zero
            0,        % direct feedthrough flag
            1];       % number of sample times

        %INITIAL CONDITIONS FOR THE STATES
        x0  = [par.V, par.T, 0];
        str = [];
        ts  = [0 0];                      % sample time: [period, offset]

    case 1
        global par
        % Model parameters
        c_p    = par.c_p;                          %heat capacity of water
        rho    = par.rho;                               %density of water
        Q_loss = par.Q_loss;                  %loss of heat from the tank


        % Extract information from Simulink model
        % inputs
        Q     = u(1);     % kW
        q_in  = u(2)/60; % l/min to l/s
        q_hw  = u(3)/60; % l/min to l/s
        T_hws = u(4);     % degrees celcius
        T_cw  = u(5);     % degrees celcius
        p     = u(6)/1000;%EUR/MWh to EUR/kWh
```

```
        % States
        V = x(1);
        T = x(2);
        J = x(3); % not used in any of the formulas

        % Differential equations
        dxdt(1) = q_in - q_hw/(1+( (T-T_hws)/(T_hws-T_cw) ) );      %[l/s]
        dxdt(2) = (1/V)*(q_in*(T_cw-T) + (Q-Q_loss)/(rho*c_p));     %[C/s]
        dxdt(3) = (p*Q)/3.6e3;                                     %[EUR/s]
        % divided by 3600 to transform from [EUR/h] to [EUR/s]
        sys = dxdt;

    case 2                                          % Discrete state update

        sys = []; % do nothing

    case 3

        % --------OUTPUTS------------

          %all the states are outputs in this case - it could be subset
          sys=x;

    case 9                                                % Terminate

        sys = []; % do nothing

    otherwise
        error(['unhandled flag = ',num2str(flag)]);
end
```

Appendix D

Generating Demand & Price Profiles

The price p and hot water demand q_{hw} are disturbances in the hot water tank system. This chapter contains the functions that makes the demand and price profiles used for the simulations. The price profile is made by reading a database file containing the electricity price for every hour of 2013 in the Trondheim region(Nord Pool Spot, 2014). The hot water demand profile is generated by `genProfile.m` using several support functions. Both `genProfile.m` and the accompanying seven support functions are part of earlier inquiries into hot water tank heaters at the Department of Chemical Engineering. They are written by de Oliveira et al. (2013), and not part of the thesis work. They are still included in here as they are a necessity to reproduce any of the results.

D.1 Forming a timeseries object: `multiProfiles.m`

This function calls two other functions, `genPrice.m` and `genProfile.m`. The returned arrays are used to generate a timeseries object. This object is then saved to ensure that the same timeseries object is used for all of the main cases. Running the script again with the same input would provide new data, as `genProfile.m` is controlled randomized.

Listing D.1: Combining two disturbances into a timeseries object.

```
function multi = multiProfiles(starttime,startday,startmonth,totdays)
%1: Calls the getPrice function to find the electricity price
%2: Calls the getProfile function multiple times to get profiles for
%   several days. Combines these into one flow object.

%1:
days.price= getPrice(starttime,startday,startmonth,totdays)
```

```
%2:
days.flow = [];
days.time = [];
for i=1:1:totdays+1 % +1 here to get 24-hour multpile data. Compensates for
    day = getProfile() %...the hours missed on first day due to "starttime"
    if i == 1  % if days.time doesn't have values days.time(end) won't work
        days.time = [days.time, day.time(starttime*60:end)]
        days.flow = [days.flow; day.flow(starttime*60:end)]
    else if i == totdays+1          % don't want to run all day if last day
        days.time = [days.time, day.time(2:starttime*60)+days.time(end)]
        days.flow = [days.flow; day.flow(2:starttime*60)]
        else      % if one of the regular days, extract data for entire day
        days.time = [days.time, day.time(2:end)+days.time(end)]
        days.flow = [days.flow; day.flow(2:end)]
        end
end
multi = days
end
```

D.2 Transforming price data: `genPrice.m`

This function reads a database file containing electricity prices for 2013. The function's starting point is specified in the function arguments with date and hour along with how many days to collect data from. It transforms the price data to a form which can be easily incorporated into a timeseries object.

Listing D.2: Transforming price data from database files.

```
function pr = getPrice(starthour, startday, startmonth, days)
% Imports price.csv then extracts the 24 hour price estimate for a given
% number of days.
%
%Arguments:
% starthour  = delay from 00:00 to start reading data for first date
% startday   = day of the month to start reading data
% startmonth = month to start reading data
% days       = number of days to extract profiles from
%
%Output:
% pr        = electricity price profile for the given dates

clc
dayspermonth= [0,31,28,31,30,31,30,31,31,30,31,30,31]; % 0 for siplicity
datalength  = (24*days)     ;                    % 24 datapoints per day
startp = starthour + 24*((sum(dayspermonth(1:startmonth))) + startday-1);
endp   = startp + datalength;
% having 0 as the first entry in dayspermonth simplifies the startp-
% formula.

csv_data = csvread('price.csv',startp,0,[startp,0,endp-1,0]);
bar = ones(60,1);          % used to trasnform data from hours to minutes
pr = csv_data(1);                % adding start point (used in timeseries)
for i=1:1:datalength;      % transforming to same time scale as other data
```

```
    temp   = csv_data(i).*bar;
    pr = [pr; temp];
end
end
```

D.3 Extracting price from database: `genPriceHour.m`

This function reads a database file starting and stopping at points specified in the function arguments. It transfers the data untouched, as opposed to `genPrice.m`.

Listing D.3: Extracting price data from database files.

```
function pr = getPrice(starthour, startday, startmonth, days)
% Imports price.csv then extracts the 24 hour price estimate for a given
% number of days.
%
%Arguments:
% starthour  = delay from 00:00 to start reading data for first date
% startday   = day of the month to start reading data
% startmonth = month to start reading data
% days       = number of days to extract profiles from
%
%Output:
% pr        = electricity price profile for the given dates

clc
dayspermonth= [0,31,28,31,30,31,30,31,31,30,31,30,31]; % 0 for siplicity
datalength  = (24*days)     ;                    % 24 datapoints per day
startp = starthour + 24*((sum(dayspermonth(1:startmonth))) + startday-1);
endp   = startp + datalength;
% having 0 as the first entry in dayspermonth simplifies the startp-
% formula.

csv_data = csvread('price.csv',startp,0,[startp,0,endp-1,0]);
bar = ones(60,1);          % used to trasnform data from hours to minutes
pr = csv_data(1);                % adding start point (used in timeseries)
for i=1:1:datalength;      % transforming to same time scale as other data
    temp   = csv_data(i).*bar;
    pr = [pr; temp];
end
end
```

D.4 Making a daily demand profile: `genProfile.m`

This function and the remaining functions of Appendix D is not part of the thesis work, and are included uncommented and unaltered.

Listing D.4: Estimating daily hot water demand.

```
function day=getProfile

dist=prob_dist_shower;
shower_dist = make_prob_table(dist);
prob_dist_small_medium
small_medium_dist = make_prob_table(dist);
prob_dist_bath
bath_dist = make_prob_table(dist);


day.time = 0:1/60:24;            % CHANGED FROM 24 TO 24-1/60!!!
day.flow = zeros(length(day.time),1);

% flow types, number of incidents/day
% short,   28
% medium,  12
% bath,    .143
% shower,  2




% generate a normal day
average_inc_pr_day = 40;

inc_today = ceil(average_inc_pr_day + 4*rand);
inc_dist = [28,12,.143,2];
inc_dist = inc_dist./sum(inc_dist);
inc_prob = cumsum(inc_dist);
inc_dist_today = [0,0,0,0];

for i = 1:inc_today
    index = find(inc_prob > rand,1);
    inc_dist_today(index) = inc_dist_today(index) + 1;
end

disp(inc_dist_today)


for i = 1:4
number_of_inc_today = inc_dist_today(i);
switch i
    case 1
        mean_flow = 1;
        std_flow = .05;
        mean_duration = 1;
        std_duration = .05;
        time_dist = small_medium_dist;
```

```
    case 2
        mean_flow = 6;
        std_flow = 1;
        mean_duration = 1;
        std_duration = .05;
        time_dist = small_medium_dist;
    case 3
        mean_flow = 14;
        std_flow = 2;
        mean_duration = 10;
        std_duration = 2;
        time_dist = bath_dist;
    case 4
        mean_flow = 8;
        std_flow = 1;
        mean_duration = 5;
        std_duration = 2;
        time_dist = shower_dist;
end;

if number_of_inc_today > 0;
    for n = 1:number_of_inc_today
        %hour of the usage
        time = time_of_usage(time_dist);
        flow = mean_flow + std_flow*randn; %l/m
        flow = max(flow,0);
        duration = mean_duration + ceil(std_duration*randn); %min
        duration=duration/60;%[now in hours]
        day.flow(( day.time >= time & day.time <=time+duration))=flow;
    end
end

end
% stairs(day.time,day.flow)
```

D.5 Demand profile helper function 1: `genProfile.m`

Listing D.5: Estimating daily hot water demand.

```
%This function inserts new time intervals in betweem two demand peaks.
%There will be one new interval for every 2h. Vectors timstart, duration
%and flows must have the same length

function [ timesvec,flowsvec ] = getTimeIntervals(timestart, duration , flows,
finaltime )


% One extra phase every 1h
interval=1;
timesvec=[0];
for i=1:length(timestart)-1
    timesvec=[timesvec timestart(i)  timestart(i)+duration(i)];
```

```
    deltatime=(timestart(i+1)-timesvec(end));
    inbetweenPhases=ceil(deltatime/(interval*3600));
    inbetweentimes=linspace(timesvec(end),timestart(i+1),inbetweenPhases);
    timesvec=[timesvec inbetweentimes(2:end-1)  ];



end

timesvec=[timesvec timestart(end)  timestart(end)+duration(end)];
deltatime=(finaltime-timesvec(end));
inbetweenPhases=ceil(deltatime/(interval*3600));
inbetweentimes=linspace(timesvec(end),finaltime,inbetweenPhases);
timesvec=[timesvec inbetweentimes(2:end)];

flowsvec=zeros(length(timesvec)-1,1);


for i=1:length(flowsvec)

    if ismember(timesvec(i),timestart)

    [~, array_position] = min(abs(timesvec(i) -timestart ));
    flowsvec(i)=flows(array_position);
    end

end


end
```

D.6 Demand profile helper function 2: `genProfile.m`

Listing D.6: Estimating daily hot water demand.

```
function [duration_vec,flow_vec,timestart_vec]=getProfileDetails

dist=prob_dist_shower;
shower_dist = make_prob_table(dist);
prob_dist_small_medium
small_medium_dist = make_prob_table(dist);
prob_dist_bath
bath_dist = make_prob_table(dist);


day.time = 0:1/60:24;
day.flow = zeros(length(day.time),1);

% flow types, number of incidents/day
% short,   28
% medium,  12
```

```
% bath,     .143
% shower,   2



% generate a normal day
average_inc_pr_day = 20;

inc_today = ceil(average_inc_pr_day + 4*rand);
inc_dist = [28,12,.143,2];
inc_dist = inc_dist./sum(inc_dist);
inc_prob = cumsum(inc_dist);
inc_dist_today = [0,0,0,0];

duration_vec=[];
flow_vec=[];
timestart_vec=[];

for i = 1:inc_today
    index = find(inc_prob > rand,1);
    inc_dist_today(index) = inc_dist_today(index) + 1;
end

disp(inc_dist_today)


for i = 1:4
number_of_inc_today = inc_dist_today(i);
switch i
    case 1
        mean_flow = 1;
        std_flow = .05;
        mean_duration = 1;
        std_duration = .05;
        time_dist = small_medium_dist;
    case 2
        mean_flow = 6;
        std_flow = 1;
        mean_duration = 1;
        std_duration = .05;
        time_dist = small_medium_dist;
    case 3
        mean_flow = 14;
        std_flow = 2;
        mean_duration = 10;
        std_duration = 2;
        time_dist = bath_dist;
    case 4
        mean_flow = 8;
        std_flow = 1;
        mean_duration = 5;
        std_duration = 2;
        time_dist = shower_dist;
end;

if number_of_inc_today > 0;
```

```
    for n = 1:number_of_inc_today
        %hour of the usage
        time = time_of_usage(time_dist); %[h]
        flow = mean_flow + std_flow*randn; %l/m
        flow = max(flow,0);
        duration = mean_duration + ceil(std_duration*randn); %min
        duration=duration/60;%[now in hours]
        duration_vec=[duration_vec duration*3600]; %[in seconds]
        flow_vec=[flow_vec flow];
        timestart_vec=[timestart_vec time*3600 ];

    end
end

end
timestart_vec=sort(timestart_vec)
% stairs(day.time,day.flow)
```

D.7 Demand profile helper function 3: `genProfile.m`

Listing D.7: Estimating daily hot water demand.

```
function dist = make_prob_table(dist)

%ensuring probabillity sum to one
total_prob = trapz(dist.t,dist.p);
dist.p = dist.p/total_prob;

% interpolates the date one minute scale
inter_p.t = 0:1/60:24;
inter_p.p = zeros(length(inter_p.t),1);
for i = 1:length(inter_p.t)
    inter_p.p(i) = interp1(dist.t,dist.p,inter_p.t(i));
end
% calulating the probability minute by minute
% descrete probabillity
inter_p.P = zeros(length(inter_p.t)-1,1);
for i = 1:length(inter_p.t)-1
    inter_p.P(i) = trapz(inter_p.t(i:i+1),inter_p.p(i:i+1));
end

dist.prob_table.p = cumsum(inter_p.P);
dist.prob_table.t = inter_p.t;

end
```

D.8 Demand profile helper function 4: `genProfile.m`

Listing D.8: Estimating daily hot water demand.

```
dist.t = [
    0
    4.9
    5
    23
    23.1
    24
    ];

dist.p = [
    .01
    .01
    .05
    .05
    .01
    .01
    ];
```

D.9 Demand profile helper function 5: `genProfile.m`

Listing D.9: Estimating daily hot water demand.

```

function dist=prob_dist_shower
dist.t = [
    0
    5
    6
    7
    8
    9
    18
    19
    19.5
    21
    23
    24
    ];

dist.p = [
    0
    0
    .15
    .25
    .15
    .02
    .02
    .05
    .09
    .02
    .02
    0
```

```
    ];
```

D.10 Demand profile helper function 6: `genProfile.m`

Listing D.10: Estimating daily hot water demand.

```
dist.t = [
    0
    7
    8
    14
    17
    19
    21
    23
    24
    ];

dist.p = [
    0
    0
    .01
    .06
    .05
    .22
    .03
    .01
    0
    ];
```

D.11 Demand profile helper function 7: `genProfile.m`

Listing D.11: Estimating daily hot water demand.

```
function time = time_of_usage(dist)

time_index = find(dist.prob_table.p > rand, 1);
time = dist.prob_table.t(time_index);

end
```

Appendix E

MATLAB Support Functions

This chapter contains the MATLAB functions that deal with extracting data from databases and transforming them to a form that makes them well suited for plots or tables.

E.1 Storing files with simulation data: `genTable.m`

This program takes data from the .mat files, arranges it in tables and writes these tables to .dat files compatible with the pgfplots package. These files serve as basis for plots and tables used in this thesis. This ensures that plots and tables in the report are always up to date. If the .mat files are not available in the folder, it is necessary to run all of the cases at least once before running `genTable.m`, otherwise data that are attempted to be extracted in the script will not exist.

Listing E.1: Storing workspace data to files.

```
% This script takes data from stored .mat files and turns it into tables,
% thus the .mat file need to be available in 'folder' before this
% script is ran. The tables are written to files using writetable()
% The generated tables are used in pgfplot commands through the thesis.
% If anything in the simulation changes, this program just has to be ran
% once to update all plots in the thesis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
folder ='C:\Users\Vegard\Dropbox\Master\latex\fig\table\';
d = 'delimiter';
% Generate 24 hour data points. The date is just there to comply with
% ..pgfplot standards.(requires full date) The hour is all that matters.
twentyfour = [];      % 24-hour array. To contain strings, starting at 4.00
for i=4:27
    if i >9   % strings contained in array must have same length.
        if i >24              % if above 24 we want to start at 01.00 again
            twentyfour=[twentyfour;['{2009-08-19 0',int2str(i-24),':00}']];
        else
```

```
            twentyfour=[twentyfour;['{2009-08-18 ',int2str(i)    ,':00}']];
        end
    elseif i<=9 % adding an additional 0 to keep strings same length.
        twentyfour =   [twentyfour;['{2009-08-18 0',int2str(i)   ,':00}']];
    end
end

%Average demand and price table
av.price  = csvread('average.csv',0,1,[0,1,23,1]);              % P, EUR/MWh
av.demand = csvread('average.csv',0,0,[0,0,23,0]);          % q_hw, dm^3/min
historicav = table(twentyfour,av.demand,av.price,...
                   'VariableNames',{'Time' 'Demand' 'Price'});
writetable(historicav,fullfile(folder,'average.dat'),d,'\t')


%Current demand and price table
test=load('langprofil.mat');
a = test.testlong;
currentdem = table(a.time' +0.0167,a.flow,a.price,...     % q_out, dm^3/min
                  'VariableNames',{'Time' 'Demand' 'Price'});
writetable(currentdem,fullfile(folder,'pricedemand.dat'),d,'\t')

%Price and average price
weekdata = getPriceHour(4,1,4,7);              % get one week of price data
pricecompTbl = table(twentyfour, weekdata(1:24),weekdata(25:48), ...
                  weekdata(49:72),weekdata(73:96),weekdata(97:120), ...
                  weekdata(121:144),weekdata(145:168), av.price);
writetable(pricecompTbl,fullfile(folder,'pricecomp.dat'),d,'\t')

%Back-off
p='C:\Users\Vegard\Dropbox\Master\MATLAB\_TJ_data_V200_m123_BO975310.mat';
bo=load(p,'Tresult','Jresult','boList');
boTbl  = table(bo.boList, bo.Tresult(:,1),bo.Tresult(:,2),...
    bo.Tresult(:,3),bo.Jresult(:,1),bo.Jresult(:,2),bo.Jresult(:,3),...
    'VariableNames',...
             {'Backoff' 'Tmode1' 'Tmode2' 'Tmode3' 'Jmode1' ...
             'Jmode2' 'Jmode3' });
writetable(boTbl,fullfile(folder,'bo.dat'),d,'\t')



%Simple control rules
simplec=load('C:\Users\Vegard\Dropbox\Master\MATLAB\cstructuresBO3.mat',...
             'rmat');
htable  = (1:numberofelements(simplec.rmat(:,1)) );
simplecTbl = table(htable', simplec.rmat(:,3), simplec.rmat(:,4),...
             simplec.rmat(:,1),simplec.rmat(:,2),'VariableNames',...
             {'Case' 'startime' 'length' 'T_violation' 'J_end' });
writetable(simplecTbl,fullfile(folder,'simplec.dat'),d,'\t')

%Just the simple control rules with s=1
newtable =[];
for i=1:numberofelements(htable)
    %temp =[]
    if simplec.rmat(i,3) == 1
        newtable=[newtable;simplec.rmat(i,1:4)];
    end
```

```
end
simplec2Tbl = table(newtable(:,1), newtable(:,2),...
              newtable(:,3), newtable(:,4),'VariableNames',...
              {'T_violation' 'J_end' 'startime' 'length'  });
writetable(simplec2Tbl,fullfile(folder,'simplec2.dat'),d,'\t')


%Sizing
path='C:\Users\Vegard\Dropbox\Master\MATLAB\_TJ_data_V60-800_m123_BO3';
sizeBO=load(path,'Tresult','Jresult','vlist');
addCostTbl =table(sizeBO.vlist(1:6), sizeBO.Jresult(1:6,1)-...
            sizeBO.Jresult(1:6,2),sizeBO.Jresult(1:6,1)-...
            sizeBO.Jresult(1:6,3),'VariableNames',...
             {'Volume','addCostOpt','addCostSim'})
writetable(addCostTbl,fullfile(folder,'addCost.dat'),d,'\t')
%no deldta
sizeCostTbl =table(sizeBO.vlist(1:6), sizeBO.Jresult(1:6,1),...
            sizeBO.Jresult(1:6,2),sizeBO.Jresult(1:6,3),'VariableNames',...
             {'Volume','Jb','Jo','Jt'})
writetable(sizeCostTbl,fullfile(folder,'sizeCost.dat'),d,'\t')


addTviolTbl = table(sizeBO.vlist(1:6), sizeBO.Tresult(1:6,1),...
          sizeBO.Tresult(1:6,2),sizeBO.Tresult(1:6,3),'VariableNames',...
              {'Volume','TviolBase','TviolOpt','TviolSim'})
writetable(addTviolTbl,fullfile(folder,'addTviol.dat'),d,'\t')

%Effect of BO on sizing
path2='C:\Users\Vegard\Dropbox\Master\MATLAB\_TJ_data_V60-800_m123';
size=load(path2,'Tresult','Jresult','vlist');
%Add statement here checking if the two incoming volume lists are  the same
if sizeBO.vlist ~= size.vlist, raise.error('The vlists nonidentical!'); end
% create the table for temp-violation with and without BO
BuTempTbl =  table(size.vlist(1:6), sizeBO.Tresult(1:6,2)-...
             sizeBO.Tresult(1:6,1),...
             size.Tresult(1:6,2)-size.Tresult(1:6,1),...
             sizeBO.Tresult(1:6,2)-sizeBO.Tresult(1:6,1)-...
             (size.Tresult(1:6,2)-size.Tresult(1:6,1)),...
             sizeBO.Tresult(1:6,3)-sizeBO.Tresult(1:6,1),...
             size.Tresult(1:6,3)-size.Tresult(1:6,1),...
             sizeBO.Tresult(1:6,3)-sizeBO.Tresult(1:6,1)-...
             (size.Tresult(1:6,3)-size.Tresult(1:6,1)),'VariableNames',...
             {'Volume','addTempOptBO','addTempOpt','deltaTempOptBO',...
             'addTempSimBO','addTempSim','deltaTempSimBO'})
% create the table for cost with and without BO
BuCostTbl =  table(size.vlist(1:6), sizeBO.Jresult(1:6,1)-...
             sizeBO.Jresult(1:6,2),...
             size.Jresult(1:6,1)-size.Jresult(1:6,2),...
             sizeBO.Jresult(1:6,1)-sizeBO.Jresult(1:6,2)-...
             (size.Jresult(1:6,1)-size.Jresult(1:6,2)),...
             sizeBO.Jresult(1:6,1)-sizeBO.Jresult(1:6,3),...
             size.Jresult(1:6,1)-size.Jresult(1:6,3),...
             sizeBO.Jresult(1:6,1)-sizeBO.Jresult(1:6,3)-...
             (size.Jresult(1:6,1)-size.Jresult(1:6,3)),'VariableNames',...
             {'Volume','addCostOptBO','addCostOpt','deltaCostOptBO',...
             'addCostSimBO','addCostSim','deltaCostSimBO'})
```

```
writetable(BuTempTbl,fullfile(folder,'TviolBO.dat'),d,'\t')
writetable(BuCostTbl,fullfile(folder,'CostBO.dat'),d,'\t')



%% Making 3 day plots

std = load('C:\Users\Vegard\Dropbox\Master\MATLAB\_datasetsV200.mat');
length=51552+288*2; %3 days in simulation points (32 steps? (96)*3)
step = 96%96%288

baseTbl = table(std.ptime_base(1:step:length),...
           std.y_base(1:step:length,2),...
           std.sp_base(1:step:length,2),std.y_base(1:step:length,1),...
           std.sp_base(1:step:length,1),std.y_base(1:step:length,3),...
           std.d_base(1:step:length,4),std.d_base(1:step:length,1),...
           std.u_base(1:step:length,1),std.u_base(1:step:length,2),...
           'VariableNames',{'time' 'T' 'T_s' 'V' 'V_s' 'J_tot' 'price'...
               'q_hw' 'Q' 'q_in'});
writetable(baseTbl,fullfile(folder,'base3.dat'),d,'\t')


%optim case
optimTbl = table(std.ptime_optim(1:step:length),...
           std.y_optim(1:step:length,2),...
           std.sp_optim(1:step:length,2),std.y_optim(1:step:length,1),...
           std.sp_optim(1:step:length,1),std.d_optim(1:step:length,4),...
           'VariableNames',{'time' 'T' 'T_s' 'V' 'V_s' 'price'});
writetable(optimTbl,fullfile(folder,'optim3.dat'),d,'\t')

%two phase case
twophaseTbl = table(std.ptime_sim(1:step:length),...
           std.y_sim(1:step:length,2),...
           std.sp_sim(1:step:length,2),std.y_sim(1:step:length,1),...
           std.sp_sim(1:step:length,1),std.d_sim(1:step:length,4),...
           'VariableNames',{'time' 'T' 'T_s' 'V' 'V_s' 'price'});
writetable(twophaseTbl,fullfile(folder,'twophase3.dat'),d,'\t');


%known demand
%plotting the results of known and average demand optimization
kno = load('C:\Users\Vegard\Dropbox\Master\MATLAB\_pdatasetsV200.mat');
knownTbl = table(kno.ptime_optim2(1:step:length),...
            kno.sp_optim2(1:step:length,2),...
             std.sp_optim(1:step:length,2),std.d_optim(1:step:length,4),...
             std.d_optim(1:step:length,1),'VariableNames',...
             {'time' 'T_sk' 'T_sa' 'price' 'q_hw'});
writetable(knownTbl,fullfile(folder,'stdVsknown3.dat'),d,'\t');

%plotting average hourly demand for three days
time2=[4:1:75]';
averagedemand =[av.demand; av.demand; av.demand];
%plotting known hourly demand for three days
actualdemand = [];
for i=1:72
    temrep = a.flow( (60*(i-1)+1):(60*i) );
    actualdemand=[actualdemand;sum(a.flow( (60*(i-1)+1):(60*i) ))/60];
```

```
end
knownTbl2 = table(time2,averagedemand, actualdemand, 'VariableNames',...
             {'time2' 'h_avdemand' 'h_actualdemand'});

writetable(knownTbl2,fullfile(folder,'stdVsknown3b.dat'),d,'\t')

step=2880
%% Plotting total cost and table
%plotting cost functions for 30 days
lenthirty = 515521;
CostyTbl = table(std.ptime_base(1:step:lenthirty)/24,...
           std.y_base(1:step:lenthirty,3),...
           std.y_optim(1:step:lenthirty,3),...
           std.y_sim(1:step:lenthirty,3),...
           'VariableNames',{'time' 'J_base' 'J_optim' 'J_sim'})
writetable(CostyTbl,fullfile(folder,'Jplot.dat'),d,'\t');
%plotting cost at the end of simulation
Var =['{$ J $}';'{$t_v$}'];
Jbase= [sizeBO.Jresult(4,1);sizeBO.Tresult(4,1)];
Joptim= [sizeBO.Jresult(4,2);sizeBO.Tresult(4,2)];
Jsim=[sizeBO.Jresult(4,3);sizeBO.Tresult(4,3)];
CostxTbl = table(Var,Jbase,Joptim,Jsim,...
           'VariableNames',{'Objective' 'J_base' 'J_optim' 'J_sim'})
writetable(CostxTbl,fullfile(folder,'Jtab.dat'),d,'\t')


% switch block
step=5
pathswitch='C:\Users\Vegard\Dropbox\Master\MATLAB\switchplot';
iz=load(pathswitch);
swstart=1;
switchTbl=table(iz.ptime_base(swstart:step:end)-iz.ptime_base(swstart),...
         iz.y_base(swstart:step:end,1),iz.sp_base(swstart:step:end,1),...
         iz.y_base(swstart:step:end,2),iz.sp_base(swstart:step:end,2),...
         'VariableNames',{'time' 'V' 'Vs' 'T' 'Ts'})
writetable(switchTbl,fullfile(folder,'Switch.dat'),d,'\t')
```

E.2 Finding average demand: `historicProfile.m`

This function calls `getProfile.m` a specified number of times, saving the average demand for each one-hour slot of the day. It then calculates an average demand for each hour by combining the generated profiles.

Listing E.2: Estimating average demand for each one-hour slot.

```
% Generates demand estiates for n days. Finds the average demand within
% each hourly slot from the n estimates.
function averageDemand = historicProfile(n)
temp = [];
for j=1:n
    rprofile = getProfile();            % generate a new demand profile
    shortprofile = rprofile.flow(2:end);  % removing the starting point
    bar = [];                             % generate temporary matrix
```

```
    for i=1:1:24       % turn minute by minute data into horly averages
        tempy = sum(shortprofile( (i-1)*60+1 : 60+(i-1)*60)/60);
        % finds the average response from: (sum of minute responses)/60
        bar=[bar,tempy];                    % saves the hourly response
    end
    temp = [temp; bar];     % append new hourly data set to temp matrix
end


ad = sum(temp)/n             %add up hourly datasets and divide by days
averageDemand= [ad(5:end),ad(1:4)]    %time shifted to start at 4.00 AM
```

E.3 Finding average price: `historicPrice.m`

This function extracts all the electricity price data from 2012, groups data points from the same time slots together and calculates an average price for each one-hour period of the day.

Listing E.3: Finding average price for each one-hour slot.

```
%finds average price per hourly slot from one year of price data
function avPrice = historicPrice
fulldata = csvread('price.csv',0,0,[0,0,8759,0]);    %read price data
temp = zeros([365,24]);     % create empty matrix for hourly datasets
for j=1:days
    for i=1:1:24                      % for each hourly period per day
        temp(j,i) = fulldata(24*(j-1)+i);% append data to hourly slot
    end
end
ad = sum(temp)./365;               % divide sum by days to get average
avPrice = [ad(5:end),ad(1:4)];     % time shifted to start at 4.00 AM
```