Optimal and Model-Free Reinforcement Learning Control of Water Distribution Networks

Project Report

Group 932

Aalborg University
Department of Electronic Systems
Control and Automation Section

Department of Electronic Systems
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Optimal and Model-Free Reinforcement Learning Control of Water Distribution Networks

Theme:
9th Semester Project Report

Project Period:
Spring 2022

Project Group:
932

Participant(s):
Jeppe Nørgaard Jensen
Mathias Clement Frederiksen

Supervisor(s):
Carsten Skovmose Kallesøe
Abhijit Mazumdar

Copies: 1

Page Numbers: 62

Date of Completion:
Dec 21, 2022

Abstract:

The focus of this project is development and application of optimal model-free machine learning control methods on a small scale water distribution network test setup with one pumping station, one consumer zone and one elevated water reservoir. Methods examined are; tabular Q-Learning and two versions of function approximation Q-Learning - first with one continuous state and later with two continuous states. A cost function is formulated such that pump power is minimised, while water level in the elevated water reservoir is kept within a certain region such that the chance of dry-out or overflow is minimised and consumer demand is met. Convergence properties of methods are compared in a simulated environment in order to find the most suitable control method candidate. This method is tested on a small scale water distribution network test setup in Aalborg Universities Smart Water Infrastructures Laboratory to verify that the method is applicable as a real world control method.

Contents

Preface

This report was written by members of group CA932 under the guidance of Carsten Skovmose Kallesøe and Abhijit Mazumdar during the 9th semester of Master's in Control and Automation, Aalborg University in the Autumn of 2022.

The central theme of this report is the use of the machine learning method reinforcement learning as a control method for water distribution networks.

Aalborg University, December 2022

Mathias Clement Frederiksen
<Mfrede21@student.aau.dk>

Jeppe Nørgaard Jensen
<jnje16@student.aau.dk>

Summary of Notation

Tabular Reinforcement Learning notation

α	Learning rate/Step size
γ	Discount factor
ϵ	ϵ-greedy parameter deciding for random action selection
$s(k)$	State at iteration k
$a(k)$	Action taken at iteration k
$\mathcal{S}, \mathcal{A}$	Closed sets of states and actions
$R(k+1)$	Reward for bringing the environment from $s(k) \rightarrow s(k+1)$
$J(k+1)$	Cost for bringing the environment from $s(k) \rightarrow s(k+1)$
$Q\Big(s(k), a(k)\Big)$	Estimates of action-value function Q_π at iteration k

Function Approximation Reinforcement Learning Notation:

$\mathbf{w}$	Weight vector
$\hat{Q}\Big(s(k), a(k), \mathbf{w}\Big)$	Approximate of action value function given weight vector $\mathbf{w}$
$\mathbf{x}\Big(s(k)\Big)$	Vector of features visible when in state $s(k)$
$\nabla_{\mathbf{w}} \hat{Q}\Big(s(k), a(k), \mathbf{w}\Big)$	Column vector og partial derivatives of $\hat{Q}(s, a, \mathbf{w})$ with respect to $\mathbf{w}$
$\mathbf{w}_{t(k),a(k)}$	Weight vectors for each discrete time t and action a at iteration k
$\mathbf{w}_{a(k)}$	Weight vectors for discrete each action a at iteration k

Water Distribution Network Notation:

$q(k)$	Water flow from pumping station
$d(k)$	Water flow into consumer section
h_{ewr}	Water height in elevated water reservoir
A	Cross-sectional area of elevated water reservoir
Δp	Differential pressure over pumping station
Δz	Drop in pressure due to geodesic pressure
$\lambda(\cdot)$	Drop in pressure due to friction i a pipe
$B\left(h_{ewr}(k)\right)$	Barrier function with water level in elevated water reservoir as input

Chapter 1

Introduction

This chapter first present the motivation behind the project, then briefly describes the difficulties of modelling a water distribution network. The chapter also presents the project description. Lastly, the report outline is presented.

1.1 Motivation

Water is a vital necessity for humans and all other creatures existence on planet earth, but water scarcity is still a huge issue that many countries face on a global level. Even though UN Sustainable Development Goal 6.1 and 6.2 respectively states [**UN_SDG6162**]:

- By 2030, (we must) achieve universal and equitable access to safe and affordable drinking water for all.
- By 2030, (we must) achieve access to adequate and equitable sanitation and hygiene for all and end open defecation, paying special attention to the needs of women and girls and those in vulnerable situations.

UNs 2021 update [**FAOandUNWater2021**] indicates that we are far away from reaching the goals - and we also were before the Covid-19 crisis.

Meeting the targets set for 2030 will require a 4x increase in pace of progress [**UN_SDG6**]. Furthermore the Russia-Ukraine war has created an energy-crisis, with the cost of energy skyrocketing all over the world, with electricity prices increasing by 57 percent in in Denmark in the first half of 2022 [**EuroStat**] and all except five European member countries experienced an increase during said period.

It is therefore critical that new control methods for water distribution are researched, such that supply of affordable water can be ensured on a worldwide scale in the close future.

I.e. meeting water demand with minimal use of energy. Ideally methods developed can be implemented without costly renovations to already existing infrastructure.

The purpose of a water distribution network is to supply water from a production facility to end-consumers. For critical infrastructure, such as a water distribution network, reliability is of the highest priority, which means that optimal control of these networks must be designed such that the control method used operates in a safe manner.

Water distribution networks have a complex topology [**MathiasJeppe730**, **Rathore1030**], which consists of different component such as pipes, valves, pumps and elevated water reservoirs. This means it might be difficult to model such a network, and even if it is possible, the model might not be sufficiently good to base control on. Hence this project focusses on model-free control of water distribution networks and will not devote time on developing models based on graph theory as the authors previously have done.

1.2 Project Description

The main objective of the semester-project is to use machine learning approaches to design a model-free and optimal controller for a water distribution network.

The dynamics of a water distribution network with an elevated reservoir is dictated by the input- and output flow of the network. To guarantee demands are met, the elevated water reservoir supplies a passive flow to the network when pumps are insufficient. This means that the control problem becomes control of reservoir water level to satisfy consumer demands. Furthermore, energy consumption is incorporated into the control problem. As the water level increases in the elevated water reservoir, so does the differential pressure across the pumping station. This results in higher power consumption at the pumping station when supplying the required flow into the system. This creates the optimality problem of minimising energy consumption without letting the reservoir dry out.

In practice pumping stations rarely have the ability to do speed control of individual pumps, but use on/off control of a finite set of pumps. Such discrete actions is embedded in the problem structure when using reinforcement learning methods, making it a suitable solution method.

This project will apply a machine learning technique called reinforcement learning to solve the before mentioned water distribution network optimality problem. Three different versions of reinforcement learning will be examined in order to compare their performance. The three methods are: tabular reinforcement learning, semi-continuous-state discrete action reinforcement learning, and continuous-state discrete-action reinforcement learning.

1.3 Report outline

The rest of the report is organised as follows:

Chapter 2 gives an introduction to reinforcement learning. First the interactions between agent and environment is explained. The Bellman optimality equation for the value function and action value function is presented. Two temporal difference methods are presented; on-policy Sarsa and off-policy Q-learning. Section 2.5 extends the concept of Q-learning into function approximation Q-Learning, which gives a solution to the dimensionality problem caused by big state-action tables in tabular Q-learning.

Chapter 3 presents the small scale water distribution network test setup. The test setup includes one pumping station, one consumer zone and an elevated water reservoir. A model of the elevated water reservoir dynamic is presented along with equations describing the power provided to the water by the pumping station. Water consumption data provided by Grundfos is presented in the end of the chapter.

Chapter 4 applies reinforcement learning methods developed in Chapter 2 to the test setup presented in Chapter 3. This includes development of a cost function, state and action selection based on the formulated cost function and action value function update laws for both tabular Q-learning and function approximation Q-learning methods. This chapter also presents basis functions used in function approximation reinforcement learning.

Chapter 5 presents results obtained from a simulated environment, using both tabular and function approximation reinforcement learning methods. Hyperparameters are swept to find good parameters for all methods. Convergence properties of the methods are compared in order to find the best control method candidate.

Chapter 6 presents results obtained when applying the best control method candidate, found in Chapter 5, to the test setup in the Smart Water Infrastructure Laboratory at Aalborg University.

Chapter 7 presents a discussion on the performance of reinforcement learning methods for simulation test. Furthermore real world implementation of reinforcement learning methods is discussed.

Chapter 8 concludes on the objectives achieved in the project and presents ideas for future work needed done before methods used in this project can be used as a real world control method.

Chapter 2

Reinforcement Learning

Reinforcement learning is a machine learning method approach concerned with optimal action selection based on current state. In this chapter a computational approach to learning through interactions with an environment is presented. The majority of information gathered is sourced to Sutton and Bartos *Reinforcement Learning - An Introduction* [**Sutton2020**] and this chapter can be seen as a preliminary chapter which can be skipped if the reader is well informed within the topic of both tabular and function approximation reinforcement learning methods.

2.1 Introduction to Reinforcement Learning

As discussed in Section 1.1 this project will focus on the application of the model-free method called reinforcement learning. Fig. 2.1 presents the basic structure of reinforcement learning and elements involved. There are two actors in the reinforcement learning framework, the *agent*, and the *environment*.

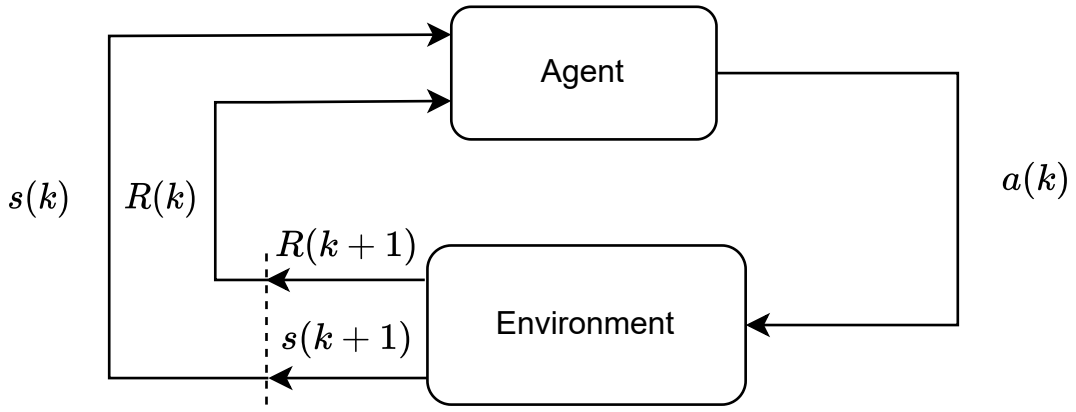


Figure 2.1: Interactions between agent and environment.

The agent applies control action $a(k)$ to the environment, based on the current state-combination $s(k)$ and receives reward $R(k+1)$, for bringing the environment to a new state $s(k+1)$. The process is repeated until the state is terminal. The reinforcement learning sequence is:

$$s(0), a(0), R(1), s(1), a(1), R(2), s(2), a(2), \cdots \tag{2.1}$$

For discrete state and discrete action reinforcement learning: $a(k) \in \mathcal{A}$, $s(k) \in \mathcal{S}$. That is, the available states and actions belongs to closed sets $\mathcal{S}, \mathcal{A}$. And the reward $R(k+1) \in \mathcal{R} \subset \mathbb{R}$. Notice that we take some notational freedom here, and exclude s and a from the R notation, R does inherently depend on state and action.

The agent seeks to maximise a series of rewards over time, called return (G), through choices of actions **[Sutton2020]**. Often a weighted sum of reward is used with $0 \leq \gamma \leq 1$:

$$G(k) = R(k+1) + \gamma R(k+2) + \cdots + \gamma^m R(k+m+1) = \sum_{n=0}^{m} \gamma^n R(k+n+1) \tag{2.2}$$

Where m is final time step of learning. In some problems learning is continuing and $m = \infty$. Learning problems with finite m are called *episodic*. This project deals with continuing learning, and we will use infinity throughout the report.

By choosing $0 \leq \gamma < 1$ it ensures that higher importance is given to immediate reward rather than future rewards. Rewriting Eq. (2.2) to a recursive form yields:

$$\begin{aligned} G(k) &= R(k+1) + \gamma R(k+2) + \gamma^2 R(k+3) + \cdots \\ &= R(k+1) + \gamma\big(R(k+2) + \gamma R(k+3) + \cdots\big) \\ &= R(k+1) + \gamma G(k+1) \end{aligned} \tag{2.3}$$

The value function is defined as the expected return under a policy $\mathbb{E}_\pi$, when starting in $a(k)$ state $s(k)$ and afterwards following a policy π and is a measure of the *value* of being in a state.

$$v_\pi\Big(s(k)\Big) = \mathbb{E}_\pi\Big[G(k) \mid s(k)\Big] = \mathbb{E}_\pi\Big[= R(k+1) + \gamma G(k+1) \mid s(k)\Big] \tag{2.4}$$

The policy π determines control actions, that is a mapping of the current state into an action - $s(k) \mapsto a(k)$. π is usually designed such that it selects actions resulting in highest expected return. Such a policy results in the optimal value-function:

$$v_\star\Big(s(k)\Big) = \max_\pi v_\pi\Big(s(k)\Big) \tag{2.5}$$

Since it is not realistic to maximise over all possible policies, we introduce the action value function in the pursuit of finding a different approach. The action value function is defined as the expected return when taking an action $a(k)$ in a state $s(k)$ and afterwards following policy π:

$$Q_\pi\Big(s(k), a(k)\Big) = \mathbb{E}_\pi\Big[G(k) \mid s(k), a(k)\Big] = \mathbb{E}_\pi\Big[R(k+1) + \gamma G(k+1) \mid s(k), a(k)\Big] \tag{2.6}$$

Optimal policies share same optimal action value functions which are defined as:

$$Q_\star\Big(s(k), a(k)\Big) = \max_\pi Q_\pi\Big(s(k), a(k)\Big) \tag{2.7}$$

$Q_\star$ can be written in terms of $v_\star$:

$$Q_\star\Big(s(k), a(k)\Big) = \mathbb{E}\Big[R(k+1) + \gamma v_\star\Big(s(k+1)\Big) \mid s(k), a(k)\Big] \tag{2.8}$$

This function computes the expected return for taking action $a(k)$ while being in state $s(k)$, and afterwards following an optimal policy. The entire value of following an optimal policy is contained in the optimal state value of the next state.

The optimal value of a state under an optimal policy must equal the expected return for the best action from that state. This action can be found by finding the action that yields the maximum value in the action value function.

$$\begin{aligned} v_\star\Big(s(k)\Big) &= \max_{a'} Q_{\pi\star}\Big(s(k), a'\Big) \\ &= \max_{a'} \mathbb{E}_{\pi\star}\Big[R(k+1) + \gamma G(k+1) \mid s(k), a'\Big] \\ &= \max_{a'} \mathbb{E}\Big[R(k+1) + \gamma v_\star\Big(s(k+1)\Big) \mid s(k), a'\Big] \\ &= \max_{a'} \sum_{s'} p\Big(s' \mid s(k), a'\Big)\Big[R(k+1) + \gamma v_*(s')\Big] \end{aligned} \tag{2.9}$$

Where $p(s'|s,a)$ is probability of transition to state s', from state $s(k)$ taking action a'. The last equation is the Bellman optimality equation, see the next Section 2.2. Similarly the Bellman equation for the action value function can be written as:

$$\begin{aligned} Q_\star\Big(s(k), a(k)\Big) &= \mathbb{E}\Big[R(k+1) + \gamma \max_{a'} Q_\star\big(s(k+1), a'\big) \mid s(k), a(k)\Big] \\ &= \sum_{s'} p\Big(s'|s(k), a(k)\Big)\Big[R(k+1) + \gamma \max_{a'} Q_*\big(s', a'\big)\Big] \end{aligned} \tag{2.10}$$

2.2 The Bellman Equation

Here a more detailed look at the Bellman equation in its general from is presented. It lays the foundation for reinforcement learning solutions.

$$v_\pi\Big(s(k)\Big) = \sum_{a'} \pi\Big(a'|s(k)\Big) \sum_{s'} p\Big(s'|s(k),a'\Big)\Big[R(k+1) + \gamma v_\pi(s')\Big] \tag{2.11}$$

where,

$v_\pi\Big(s(k)\Big)$	is the value of state $s(k)$ under policy π
$\pi\Big(a'\|s(k)\Big)$	is the conditional probability of taking action a' in state $s(k)$ under stochastic policy π
$p\Big(s'\|s(k),a'\Big)$	is the conditional probability of transitioning to state s' when taking action a' in state $s(k)$

Eq. (2.11) is the Bellman equation for the value function. The equation can be read as weighing the immediate reward and a discounted version of future state value by the probability of getting that reward and next state, when taking a specific action in a given state. When summing over all possible actions and their possible next states an expression for the expected value of being in that state is obtained. In some versions of Eq. (2.11) you also sum over all possible rewards, however, we assume it to be deterministic.

These optimality equations make systems of equations which can be solved, this is the Dynamic Programming approach. However, the set of equations are potentially unsolvable, and require a complete model of the system to know the transition probabilities. Therefore Section 2.4 considers model-free estimation methods *SARSA* and *Q-learning*.

2.3 Policies

The simplest policy chooses actions based on the highest expected return achievable in a state $s(k)$, that is, the highest action value:

$$a(k) = \underset{a' \in \mathcal{A}}{\arg\max}\, Q\Big(s(k),a'\Big) \tag{2.12}$$

Eq. (2.12) is called the *greedy-policy*. The policy clearly takes advantage of already known information about the action values, this is known as *exploitation*.

A greedy policy always chooses the action that it thinks is best at the current time, but this

may result in convergence to a sub-optimal policy. The algorithm can get tunnel vision on a specific policy if it never checks other options, options which might yield better returns, especially in the long run. For example scenarios where multiple bad steps must be taken before a big reward is achieved. A greedy policy is also not able to detect changes in the environment. Allowing the agent to *explore* new knowledge about the action value function can improve the agent's knowledge and lead to better decision making and performance. Eq. (2.13) presents the *Epsilon-Greedy policy* which is a very simple method that balances exploration and exploitation by randomly choosing between the two options:

$$a(k) = \begin{cases} \underset{a' \in \mathcal{A}}{\arg\max}\, Q\Big(s(k), a'\Big) & \text{exploitation with probability } 1-\epsilon \\ \text{Random action from } \mathcal{A} & \text{exploration with probability } \epsilon \end{cases}, \quad \epsilon \in [0,1] \tag{2.13}$$

If the ϵ-value is chosen low Eq. (2.13) behaves greedy most of the time, but sometimes, instead of following the greedy-policy, it randomly chooses an action from the closed set of actions with equal probability independent of best action value. Doing this means the entire state-action space is explored as $k \to \infty$. The downside is the immediate loss of reward from randomly choosing a worse action, this is especially noticeable as the algorithm gets closer to convergence, and shows the diminishing effectiveness of exploration.

2.4 Temporal Difference Methods

We choose to skip the topic of Monte Carlo simulation, as we wish to avoid its disadvantage of batch updating - see e.g. [**Busoniu2010**, **Sutton2020**] - and go directly to the topic of Temporal Difference learning. Temporal difference learning is a central idea of reinforcement learning. It makes it possible for the agent to learn while interacting with the environment for which no knowledge about the dynamics is known [**DaMottaSallesBarreto2008**], and in an iterative way finds the optimal action value function. It is also proven to converge to an optimal policy under the same conditions as other methods, and although temporal difference methods has not been proven to converge faster mathematically, empiric experiments show that temporal difference methods converge faster than Monte Carlo methods [**Sutton2020**].

SARSA is an abbreviation of State-Action-Reward-State-Action and forms the basis of TD(0) learning:

$$Q\Big(s(k),a(k)\Big) \leftarrow Q\Big(s(k),a(k)\Big) + \alpha\Bigg[\underbrace{R(k+1)+\gamma Q\Big(s(k+1),a(k+1)\Big)}_{\text{Temporal difference target/Sarsa target}} - Q\Big(s(k),a(k)\Big)\Bigg] \tag{2.14}$$

Unlike Monte Carlo methods, Temporal Difference learning updates an action value after every iteration. This results in least amount of time spent on making sub-optimal actions. The method updates an estimate of the action value function based on the same estimate. This is called bootstrapping.

Temporal Difference methods use *sample updates* and therefore does not require a system model. This means looking ahead and combining the value at the next state-action pair and the reward obtained along the way, to compute a backed-up value for the current iteration (k), which is used to update the original state [**Sutton2020**].

One of the most used reinforcement learning method is called Q-learning, which like Eq. (2.14) approximates the Bellman equation by updating the action value function iteratively as seen in Eq. (2.15).

$$Q\Big(s(k),a(k)\Big) \leftarrow Q\Big(s(k),a(k)\Big) + \alpha\Bigg[\underbrace{R(k+1)+\gamma \max_{a'} Q\Big(s(k+1),a'\Big)}_{\text{Q-Learning target}} - Q\Big(s(k),a(k)\Big)\Bigg] \tag{2.15}$$

where,

α is the *Learning rate*. $0 \leq \alpha \leq 1$.

Note that the only difference in the two update methods lies in the target. This difference is called *on-* and *off-* policy, depending on whether the update is based on strictly staying on policy, meaning target action is selected according to policy (SARSA), or target action is chosen optimally, independent of policy (Q-learning). This essentially means that Q-learning converges to the optimal policy immediately, where SARSA converges to a less optimal policy which also consider the randomness of the ϵ-greedy policy, such that a *safer* policy is obtained. Here safety means policies in which random actions can lead to large negative reward, achieve less value.

The learning rate α dictates the size of updates, but also affects if the algorithm can converge at all. The following properties on α grantees convergence [**Busoniu2010**], such that if:

$$\sum_{k=1}^{\infty} \alpha(k) = \infty \quad \wedge \quad \sum_{k=1}^{\infty} \alpha(k)^2 < \infty \tag{2.16}$$

Then:

$$Q(s,a) \rightarrow Q_{\star}(s,a) \text{ as } k \rightarrow \infty, \ \forall s, a \in \mathcal{S}, \mathcal{A} \tag{2.17}$$

Where $Q_{\star}$ denotes the optimal action value function. The first condition in Eq. (2.16) is required to guarantee that the steps are large enough to eventually overcome any initial conditions or random fluctuations. The second condition guarantees that eventually the steps become small enough to assure convergence [**Sutton2020**].

2.5 Function Approximation

One way of keeping track of the action values is to use a table. For large systems with high state-action dimensionality this is not desirable. The agent is required to visit all state-action combinations at least once to get an idea of optimal decision making. Visiting the whole state-action space could take an enormous amount of time resulting in long learning periods and convergence times.

Instead the action value function can be approximated [**Sutton2020**]. This allows us to generalise information about visited state-actions pairs to not yet visited pairs:

$$\hat{Q}\Big(s(k), a(k), \mathbf{w}\Big) \approx Q_{\pi}\Big(s(k), a(k)\Big) \tag{2.18}$$

The state-action function estimate is now represented as a parametrised function form with a weight vector $\mathbf{w} \in \mathbb{R}^d$. The linear[1] approximation of the action-value function is given by:

$$\hat{Q}\Big(s(k), a(k), \mathbf{w}\Big) = \mathbf{w}^{\mathsf{T}}\mathbf{x}\Big(s(k), a(k)\Big) = \sum_{i=1}^{d} w_i x_i\Big(s(k), a(k)\Big) \tag{2.19}$$

Where

$$\mathbf{x}\Big(s(k), a(k)\Big) = \Big[x_1\Big(s(k), a(k)\Big), x_2\Big(s(k), a(k)\Big) \cdots x_d\Big(s(k), a(k)\Big)\Big]^{\mathsf{T}} \tag{2.20}$$

[1] Linear with respect to weights

is the feature vector and each element x_i is called a feature of the states.

The weight vector is updated using stochastic gradient descent, which minimises the squared error between the true state-action function and the estimated function [**Overgaard2019**]:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \frac{1}{2}\alpha\nabla_{\mathbf{w}}\left[Q_\pi\Big(s(k),a(k)\Big) - \hat{Q}\Big(s(k),a(k),\mathbf{w}\Big)\right]^2 \\ &= \mathbf{w} + \alpha\left[Q_\pi\Big(s(k),a(k)\Big) - \hat{Q}\Big(s(k),a(k),\mathbf{w}\Big)\right]\nabla_{\mathbf{w}}\hat{Q}\Big(s(k),a(k),\mathbf{w}\Big)\end{aligned} \tag{2.21}$$

α is a positive step size, and $\nabla_{\mathbf{w}}\hat{Q}\Big(s(k),a(k),\mathbf{w}\Big)$ is the gradient of the function $\hat{Q}$ with respect to the weight vector $\mathbf{w}$. Eq. (2.21) requires that the actual action value function Q_π is known which is typically not the case. An estimate, U, of $Q_\pi\Big(s(k),a(k)\Big)$ is used:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left[U - \hat{Q}\Big(s(k),a(k),\mathbf{w}\Big)\right]\nabla\hat{Q}\Big(s(k),a(k),\mathbf{w}\Big) \tag{2.22}$$

Choosing the estimate as in [**Overgaard2019**, **Sutton2020**]:

$$U = R(k+1) + \gamma\max_{a'}\hat{Q}\Big(s(k+1),a',\mathbf{w}\Big) \tag{2.23}$$

Combining Eq. (2.22) and Eq. (2.23) yields the update law for the weight vector:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left[R(k+1) + \gamma\max_{a'}\hat{Q}\Big(s(k+1),a',\mathbf{w}\Big) - \hat{Q}\Big(s(k),a(k),\mathbf{w}\Big)\right]\nabla_{\mathbf{w}}\hat{Q}\Big(s(k),a(k),\mathbf{w}\Big) \tag{2.24}$$

Evaluating $\nabla_{\mathbf{w}}\hat{Q}\Big(s(k),a(k),\mathbf{w}\Big)$ in Eq. (2.19) yields:

$$\begin{aligned}\nabla_{\mathbf{w}}\hat{Q}\Big(s(k),a(k),\mathbf{w}\Big) &= \nabla_{\mathbf{w}}\Big(\mathbf{w}^\intercal\mathbf{x}\Big(s(k),a(k)\Big)\Big) \\ &= \mathbf{x}\Big(s(k),a(k)\Big)\end{aligned} \tag{2.25}$$

Eq. (2.26) presents the semi-gradient descent update rule for function approximation:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R(k+1) + \gamma \max_{a'} \hat{Q}\left(s(k+1), a', \mathbf{w} \right) - \hat{Q}\left(s(k), a(k), \mathbf{w} \right) \right] \mathbf{x}\left(s(k), a(k) \right) \quad (2.26)$$

Chapter 3

Water Distribution Networks

A water distribution network is a hydraulic network which consists of different components; pipes, pumps, valves and elevated water reservoirs. The network is usually divided into several consumer zones. A ring-type water distribution network can be seen in Fig. 3.1 with two consumer zones, two pumping stations, and an elevated water reservoir. As mentioned in Section 1.1 the purpose of an elevated water reservoir is to maintain pressure, but also to store water in case of an emergency. When the pressure is high in the network but the demand is low, water will flow into the reservoir and vice-versa. But from an economic viewpoint it is desired to keep the pumps running as little as possible resulting in lower water levels in the reservoir. Efficient operation of the water distribution network inherently means minimising energy usage, which extends efficient operation to also impact the green transition [**Val2021a**].

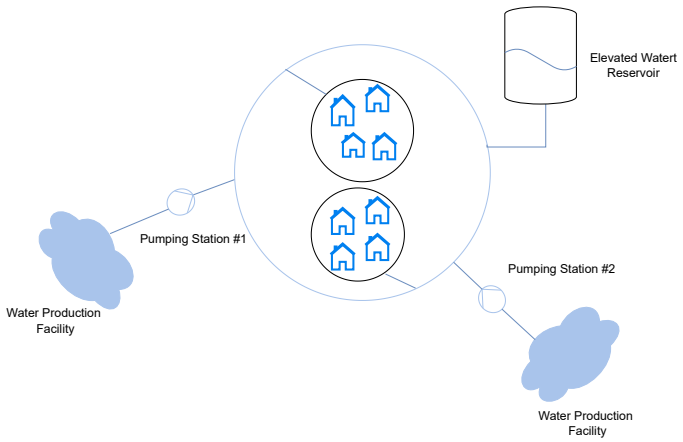


Figure 3.1: A typical ring-type water distribution network with two pumping stations, multiple consumers and an elevated reservoir.

In order to verify the methods of this project are applicable in the real world, they are implemented on a water distribution network test setup.

3.1 Test Setup

Fig. 3.2 presents the water distribution network test setup used in the project. The system consists of three parallel coupled pumps, acting as a pumping station which supplies water to a variable valve acting as a consumer zone, and an elevated water reservoir. For a comprehensive overview of component models in water distribution see [**Jensen**].

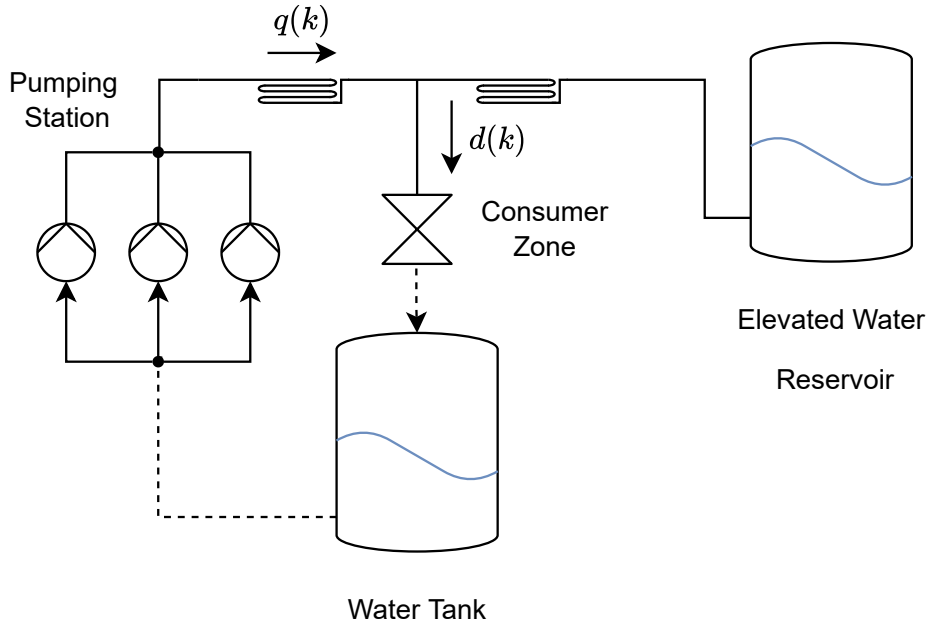


Figure 3.2: Water distribution network with pumping station, consumer zone and elevated water reservoir.

The water tank is used to store water for the pumping station and is kept at a constant level with the help of a PI-controller that controls a set of auxiliary pumps.

The only dynamic component of the network seen in Fig. 3.2 is the elevated water reservoir, and can be described by the following differential equation:

$$A\dot{h} = q + d \Leftrightarrow \dot{h} = \frac{q+d}{A} \tag{3.1}$$

Eq. (3.1) can be discretized by Euler's method:

$$h(k+1) = h(k) + \frac{q(k) + d(k)}{A} \tag{3.2}$$

where,

$h(k)$	is the water level in the elevated water reservoir	[m]
$q(k)$	is the flow of water through the pumps	[m^3/s]
$d(k)$	is the water demand	[m^3/s]
A	is the cross-sectional area of the elevated water reservoir	[m^2]

Note that the cross-sectional area of the elevated water reservoir is assumed constant along its vertical axis. [**Rathore930**, **MathiasJeppe730**] uses the same elevated water reservoir as this project and states the following parameters:

Diameter [m]	Height [m]	Cross sectional area [m^2]	Capacity [m^3]
0.6	0.706	0.283	0.2

Table 3.1: Elevated water reservoir parameters. Cross sectional is used in simulation.

The power delivered to the water from the pumping station in Fig. 3.2 is:

$$P(k) = q(k)\Delta p = q(k)(p_{out}(k) - p_{in}(k)) \tag{3.3}$$

Where the system pressure p_{out} is:

$$p_{out}(k) = \rho g h_{ewr}(k) + \Delta z + \lambda_1\Big(q(k)\Big) + \lambda_2\Big(q(k), d(k)\Big) \tag{3.4}$$

where,

ρ	is the density of water	[l/m^3]
g	is the gravitational acceleration	[m/s^2]
Δz	is the drop in pressure due to geodesic level of tank	[Pa]
$\lambda(\cdot)$	is the drop in pressure due to friction in a pipe-segment	[Pa]
p_{out}, p_{in}	are the absolute pressure at outlet and inlet side of pumping station	[Pa]

It is assumed that the pressure drop in the pipe connected to the pumping station, due to friction, takes the form:

$$\lambda_1\Big(q(k)\Big) = r_1|q(k)|q(k) \tag{3.5}$$

and for a pipe connected to an elevated water reservoir the pressure drop takes the form:

$$\lambda_2\Big(q(k), d(k)\Big) = r_2\big|q(k) - d(k)\big|\Big(q(k) - d(k)\Big) \tag{3.6}$$

with friction constants r_i.

The pump inlet is assumed open to atmosphere and the inlet pressure becomes $p_{in} = 0$. The power delivered from the pumping station is:

$$P(k) = q(k)\Big(\rho g h_{ewr}(k) + \Delta z + \lambda_1\Big(q(k)\Big) + \lambda_2\Big(q(k), d(k)\Big)\Big) \tag{3.7}$$

3.2 Consumption Model

Real water consumption data is analysed. A historical dataset of daily water consumption over a three months period has been provided by Grundfos, which contains data sampled four times per hour. Fig. 3.3 depicts consumption of a 24 hour period. Note that the unit of the consumption data is unknown and that Fig. 3.3 is normalised.

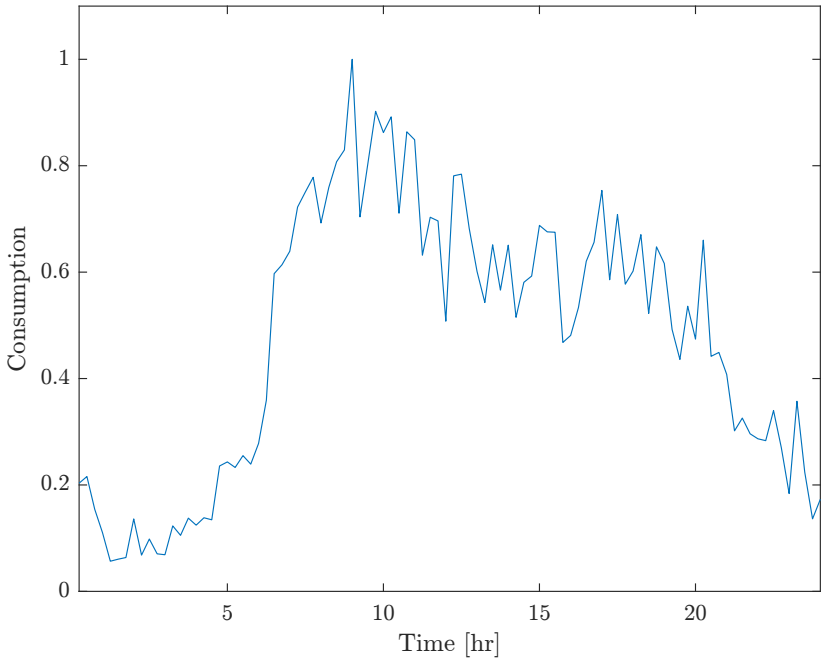


Figure 3.3: Consumption profile in a day.

The power equation presented in Eq. (3.7) will in the following section be used to select states and action of the system, such that reinforcement learning can be used as a control method. The consumer data will be used to emulate the consumption of a consumer zone, and will play a role when doing full state function approximation Q-learning.

Chapter 4

Water Distribution Networks and Reinforcement Learning

In this chapter it is shown how reinforcement learning methods from Chapter 2 are used as control methods on the water distribution network presented in Chapter 3. A cost function is formulated based on minimisation of energy consumption of pumps, combined with a barrier function. System states and action is selected based on the cost function.

Chapter 2 describes two learning algorithms; the on-policy learning method Sarsa and the off-policy learning method Q-learning. The Sarsa algorithm Eq. (2.14) will in time learn the optimal ϵ-greedy policy, that is the Q-values will converge to some sub-optimal Q-value function. Using Q-learning will result in the convergence to an optimal policy [**Sutton2020**], therefore the Q-learning algorithm will be used for the remainder of the project.

Notice from here onward, we distinguish between the height state and actual water level height, by denoting them $h(k)$ and $h_{ewh}(k)$ respectively.

4.1 Cost Function and State-Action Selection

Chapter 2 presented Reinforcement Learning and the idea of maximisation a reward function to get the greatest return. The remainder of this project will discuss minimization of a cost functions rather than maximisation of a reward functions. This means that the action value becomes related to an expected cost for which lower values are better. The ϵ-greedy policy should then be altered such that the next action is found based on lowest action value rather than the biggest action value:

$$a(k) = \begin{cases} \underset{a' \in A}{\arg\min}\, Q\Big(s(k), a'\Big) & \text{exploitation with probability } 1-\epsilon \\ \text{Random action from } \mathcal{A} & \text{exploration with probability } \epsilon \end{cases}, \quad \epsilon \in [0,1] \tag{4.1}$$

The cost function J forms the basis of the optimality problem. We formulate the cost function such that the power consumed by the pumping station is minimised and on the fact that we wish to stay within some water level limits in the elevated water reservoir:

$$J(k+1) = P(k) + B\Big(h_{ewr}(k+1)\Big) \tag{4.2}$$

where,

$P(k)$	is the power consumed by the pumping station	[W]
$B\Big(h_{ewr}(k+1)\Big)$	is a barrier function	$(\cdot)$

Note that the cost is based on pump power at time k and the barrier cost is evaluated at time $k+1$ resulting in high cost given to actions that takes the water level into the barrier. Combining Eq. (4.2) and Eq. (3.7) yields:

$$J(k+1) = q(k)\Big(\rho g h_{ewr}(k+1) + \Delta z + \lambda\Big(q(k)\Big) + \lambda\Big(q(k), d(k)\Big)\Big) + B\Big(h_{ewr}(k+1)\Big) \tag{4.3}$$

The barrier function is designed such that high costs is given to water levels that are outside certain heights bounds. The barrier helps in avoiding dry-out, this is important since it counteracts the minimisation of energy inclination to empty the reservoir. Avoiding dry-out also guarantees that demand will be satisfied, because demand will drain the reservoir if pumps supply insufficient water.

Eq. (4.4) presents the quadratic polynomial barrier function used:

$$B\Big(h_{ewr}(k), a, b\Big) = \begin{cases} (h_{ewr}(k) - a)^2 & \text{if } h_{ewr}(k) < a \\ 0 & \text{if } h_{ewr}(k) \in \big[a, b\big] \\ (h_{ewr}(k) - b)^2 & \text{if } h_{ewr}(k) > b \end{cases} \tag{4.4}$$

With a and b being the zeros of the left and right side polynomial respectively. A example of a barrier function of this form is shown in Fig. 4.1:

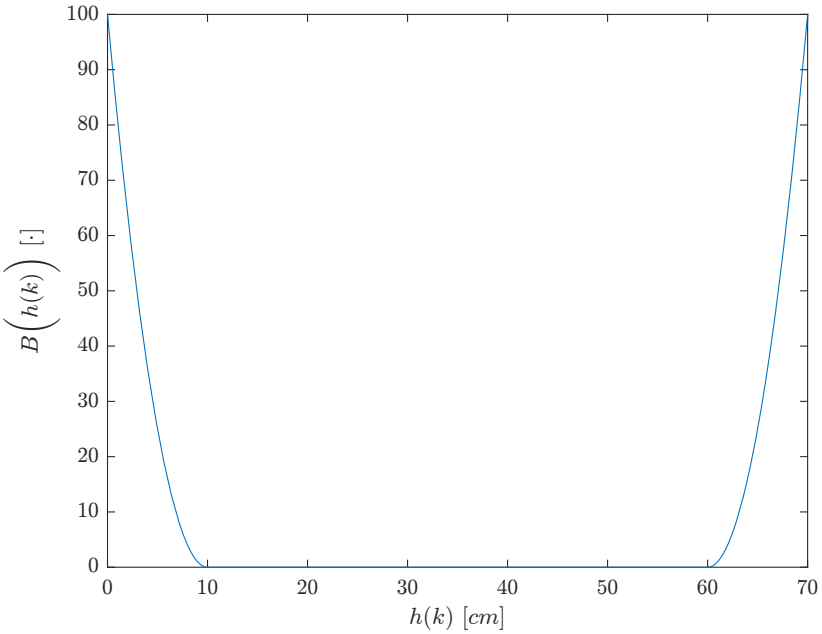


Figure 4.1: Piecewise defined quadratic barrier function from Eq. (4.4) with zeros in 10 and 60.

States and action for the water distribution network reinforcement learning problem is chosen based on the cost function in Eq. (4.3). We choose:

- $s_1 = h_{ewr}$: Water level of tank scales linearly to the pressure in the tank, and determines barrier cost.
- $s_2 = t$: Time of day. It is assumed that the consumer demand $d(k)$ behaves periodically from day to day.
- $a = q(k)$: actions are three unique pumping-station presets, each with a different resulting flow.

4.2 Tabular Q-learning

For tabular Q-Learning the action values are stored in a table format: $Q\Big(h(k), t(k), a(k)\Big)$. One table dimension holds discrete height states, one discrete time states and one discrete actions.

This section explains how the states are discritised in order to fit the water distribution network described in Chapter 3.

Time is discretised to 24 states, matching hours of the day. A finer discretisation is not used for dimensionality issues, and it is assumed dynamics are slow enough to be captured on a hourly basis.

Discretisation of water levels in the elevated water reservoir is visualised in Fig. 4.2. The first and last height states consists of wider height segments than the middle states. This minimises the chance of dry-out and overflow. The water level is measured in centimetres and the height state $h(k)$ will be chosen based on the actual water level:

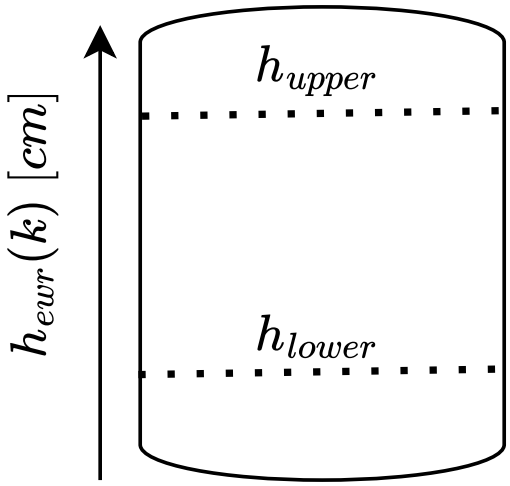


Figure 4.2: Height state in elevated water reservoir.

$$h(k) = \begin{cases} 1 & if\ h_{ewr}(k) \leq h_{Lower} \\ ceil\Big(h_{ewr}(k)\Big) & if\ h_{lower} < h_{ewr}(k) < h_{upper} \\ h_{max} & if\ h_{ewr}(k) \geq h_{upper} \end{cases} \tag{4.5}$$

Where the placements of h_{lower} and h_{upper} decide the number of height states, and the resulting upper limit of height states is denoted h_{max}. *ceil* denotes the *ceiling* function, which rounds any number to the nearest larger integer. Our use of the ceil function requires the argument to be measured in centimetres for proper height state assignment, therefore any measurement not in centimetres must first be scaled.

The number of actions is dictated by the number of pumps in the pumping station, and is by nature already discrete.

The Q-learning update law for the action value function described in Eq. (2.15) will in the case of minimising the cost function in Eq. (4.2) be:

$$Q\Big(h(k),t(k),a(k)\Big) \leftarrow Q\Big(h(k),t(k),a(k)\Big) + \alpha\Big[J(k+1) + \gamma \min_{a'} Q\Big(h(k+1),t(k+1),a'\Big) - Q\Big(h(k),t(k),a(k)\Big)\Big] \tag{4.6}$$

with $h(k) \in \mathcal{H}$, $t(k) \in \mathcal{T}$ and $a(k) \in \mathcal{A}$. That is the available states and actions belongs to closed sets.

Algorithm 1 presents the tabular Q-learning algorithm applied to the water distribution network presented in Section 3.1.

Algorithm 1 Tabular Q-Learning algorithm

1: **Input:** $\alpha \in [0,\ 1]$, small $\epsilon > 0$, and $0 \leq \gamma < 1$
2: **Initialise:** $Q(h,t,a)$ for all $h(k) \in \mathcal{H}$, $t(k) \in \mathcal{T}$ and $a(k) \in \mathcal{A}$, and states $t(1)$ and $h(1)$
3: **for** k=1,2,... **do**
4: Take action according to ϵ-greedy policy:

$$a(k) = \begin{cases} \underset{a' \in \mathcal{A}}{\arg\min}\, Q\Big(h(k),t(k),a'\Big) & \text{exploitation with probability } 1-\epsilon \\ \text{Random action from } \mathcal{A} & \text{exploration with probability } \epsilon \end{cases}$$

5: Observe next states $h(k+1)$, $t(k+1)$ and cost $J(k+1)$
6: Update Q according to:

$$Q\Big(h(k),t(k),a(k)\Big) \leftarrow Q\Big(h(k),t(k),a(k)\Big) + \alpha\Big[J(k+1) + \gamma \min_{a'} Q\Big(h(k+1),t(k+1),a'\Big) - Q\Big(h(k),t(k),a(k)\Big)\Big]$$

7: **end for**

4.3 Function Approximation Q-learning

To ease notation, and since the following sections does not require discretised height state, we will once again denote the actual water level as $h(k)$.

Section 2.5 established that the action value function could be approximated as a linear combination of a weights vector and a set of basis functions:

$$\hat{Q}\Big(s(k), a(k), \mathbf{w}\Big) = \mathbf{w}^{\mathsf{T}}\mathbf{x}\Big(s(k), a(k)\Big) \tag{4.7}$$

Applying function approximation to the water distribution network described in Chapter 3 with states and actions chosen as in Section 4.1 yields:

$$\hat{Q}\Big(h(k), t(k), a(k), \mathbf{w}\Big) = \mathbf{w}^{\mathsf{T}}\mathbf{x}\Big(h(k), t(k), a(k)\Big) \tag{4.8}$$

The approximation shown above is true for the case where all states and actions are continuous which is not the case for this project. This section shows how to deal with the following cases of continuous/discrete states and action combinations:

- Continuous water level, and discrete time and action.
- Continuous water level and time, and discrete action.

4.3.1 Continuous Height, and Discrete Time and Action

In the continuous height, and discrete time and action case we have $s = (t, h)$ where t is in a finite set and h is continuous. It is necessary to create a weight-vector for every time-action combination. The action-value function will be approximated as:

$$\hat{Q}\Big(h(k), \mathbf{w}_{t(k),a(k)}\Big) = \mathbf{w}^{\mathsf{T}}_{t(k),a(k)}\mathbf{x}\Big(h(k)\Big) \tag{4.9}$$

The subscripts of $\mathbf{w}$ refer to one specific weight vector. There exists one weight vector for each discrete time, discrete action combination: For the case of 24 discrete times and 3 discrete actions there exist 96 weight vectors.

Radial basis functions are used for the height feature vector. They are easy to design and provide good generalisation due to their Gaussian bell shapes:

$$x_i\Big(h(k)\Big) = \exp\left(\frac{-||h(k) - c_i||^2}{2\sigma_i^2}\right) \tag{4.10}$$

The radial basis function is shaped by the feature's width σ_i, and the feature response depends only on the distance to the state center c_i [**Sutton2020**]. Fig. 4.3 shows radial basis functions in one dimension:

10 centers are placed uniformly in the range of 0 to 70cm according the the elevated water reservoir height presented in Table 3.1:

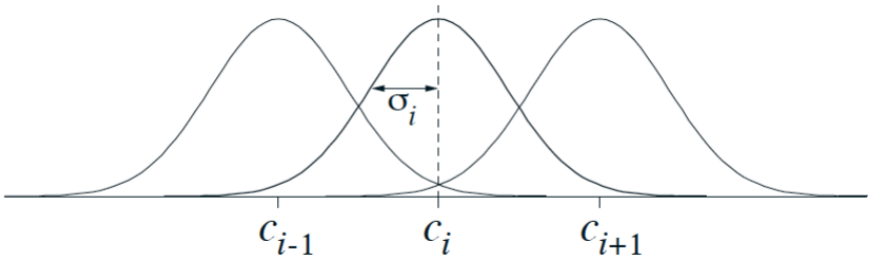


Figure 4.3: Radial basis functions in one dimension [**Sutton2020**].

$$c = \begin{bmatrix} 0 & 7.78 & 15.56 & 23.33 & 31.11 & 38.89 & 46.67 & 54.44 & 62.22 & 70 \end{bmatrix}$$

An even radial basis function generalisation is achieved by choosing $\sigma = 4$ such that the sum of radial basis functions equals one over the entire range of elevated water reservoir heights. The individual radial basis function is plotted in Fig. 4.4 along with the sum of all radial basis functions.

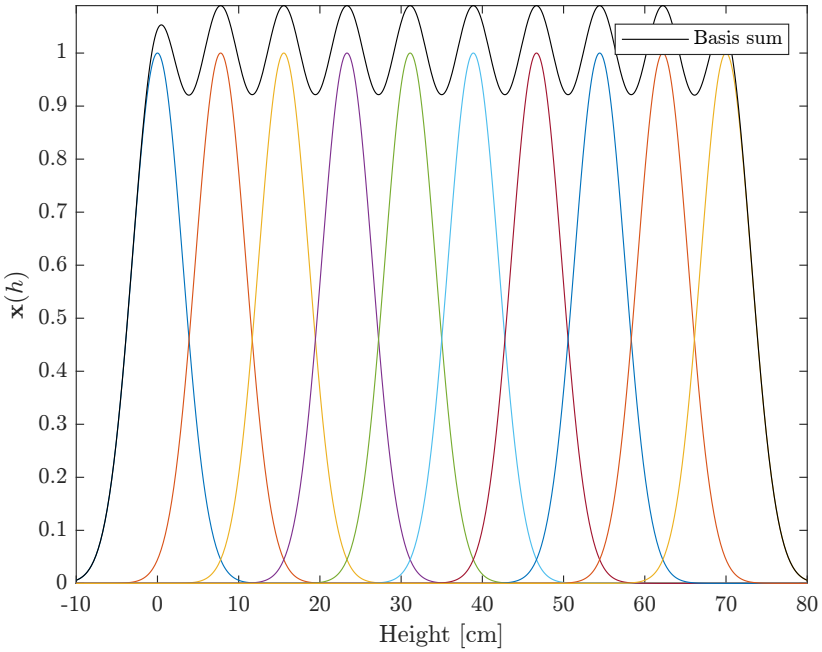


Figure 4.4: Radial basis functions and their sum.

The weight vectors update rule will in the continuous height, discrete time and action case be:

$$\mathbf{w}_{t(k),a(k)} \leftarrow \mathbf{w}_{t(k),a(k)} + \alpha \Bigg[J(k+1) + \gamma \min_{a'} \left(\mathbf{w}_{t(k+1),a'}^{\mathsf{T}} \mathbf{x}\Big(h(k+1) \Big) \right) - \mathbf{w}_{t(k),a(k)}^{\mathsf{T}} \mathbf{x}(h\Big(k \Big) \Bigg] \mathbf{x}\Big(h(k) \Big) \tag{4.11}$$

Where $t(k)$ and $a(k)$ is taken from finite sets. Algorithm 2 presents the function approximation Q-Learning algorithm applied to the water distribution network shown in Section 3.1. with continuous height state and discrete time state and discrete actions.

Algorithm 2 Continuous height discrete time and action Function approximation Q-Learning algorithm

1: **Input:** $\alpha \in [0,\ 1]$, small $\epsilon > 0$, and $0 \leq \gamma < 1$, number of Radial basis functions, center placements and width
2: **Initialise:** weight vectors $\mathbf{w}_{t,a}$ for all discrete times and actions (e.g. $\mathbf{w}_{t,a} = 0$), and states $t(1)$ and $h(1)$
3: **for** k=1,2,... **do**
4: Take action according to a ϵ-greedy policy:

$$a(k) = \begin{cases} \underset{a' \in \mathcal{A}}{\arg\min}\ \hat{Q}\Big(h(k), \mathbf{w}_{t(k),a'} \Big) & \text{exploitation with probability } 1-\epsilon \\ \text{Random action from } \mathcal{A} & \text{exploration with probability } \epsilon \end{cases}$$

5: Observe next states $h(k+1)$, $t(k+1)$ and cost $J(k+1)$
6: Update $\mathbf{w}_{t(k),a(k)}$ according to:

$$\mathbf{w}_{t(k),a(k)} \leftarrow \mathbf{w}_{t(k),a(k)} + \alpha \Bigg[J(k+1) + \gamma \min_{a'} \left(\mathbf{w}_{t(k+1),a'}^{\mathsf{T}} \mathbf{x}\Big(h(k+1) \Big) \right) - \mathbf{w}_{t(k),a(k)}^{\mathsf{T}} \mathbf{x}\Big(h(k) \Big) \Bigg] \mathbf{x}\Big(h(k) \Big)$$

7: **end for**

4.3.2 Continuous Height and Time, and Discrete Action

In the continuous water level and time, and discrete action case, it is necessary to create a weight-vector for every discrete action.

$$\hat{Q}\Big(h(k), t(k), \mathbf{w}_a\Big) = \mathbf{w}_{a(k)}^{\mathsf{T}} \mathbf{x}\Big(h(k), t(k)\Big) \tag{4.12}$$

This reduces the number of weight vectors from 96 as in Section 4.3.1 to only 3. As in Section 4.3.1 radial basis functions are used as basis function for the height part of the feature vector. Another linear function approximation method is based on Fourier series, which are useful for approximating periodic functions such as water consumption:

$$f(t) = a_0 + \sum_{n=1}^{n=N} (a_n \cos(2\pi n f_0 t) + b_n \sin(2\pi n f_0 t)) \tag{4.13}$$

Where a_0, a_n and $b_n \in \mathbb{R}$ are the Fourier coefficients and f_0 is the first harmonic of the signal. Eq. (4.13) can be rewritten as a linear combination of weights $\mathbf{w}$ and a feature vector $\mathbf{x}$ containing sin and cos basis functions.

$$f(t) = \mathbf{w}^{\mathsf{T}} \mathbf{x}(t) \tag{4.14}$$

With weight vector and basis functions defined as:

$$\mathbf{w} = \begin{bmatrix} a_0 \\ a_1 \\ b_1 \\ a_2 \\ \vdots \end{bmatrix} \quad \wedge \quad \mathbf{x}(t) = \begin{bmatrix} 1 \\ \sin(2\pi f_0 t) \\ \cos(2\pi f_0 t) \\ \sin(4\pi f_0 t) \\ \vdots \end{bmatrix} \tag{4.15}$$

The harmonic frequencies found is based on a fast Fourier transformation of the consumer data described in Section 3.2. From the fast Fourier transform, shown in Fig. 4.5, the demand profile is found to have a major dc-component and frequency-components corresponding to periods of 24 hours, 12 hours, 8 hours, 6 hours, and 4 hours corresponding to the first, second, third, fourth and sixth harmonic.

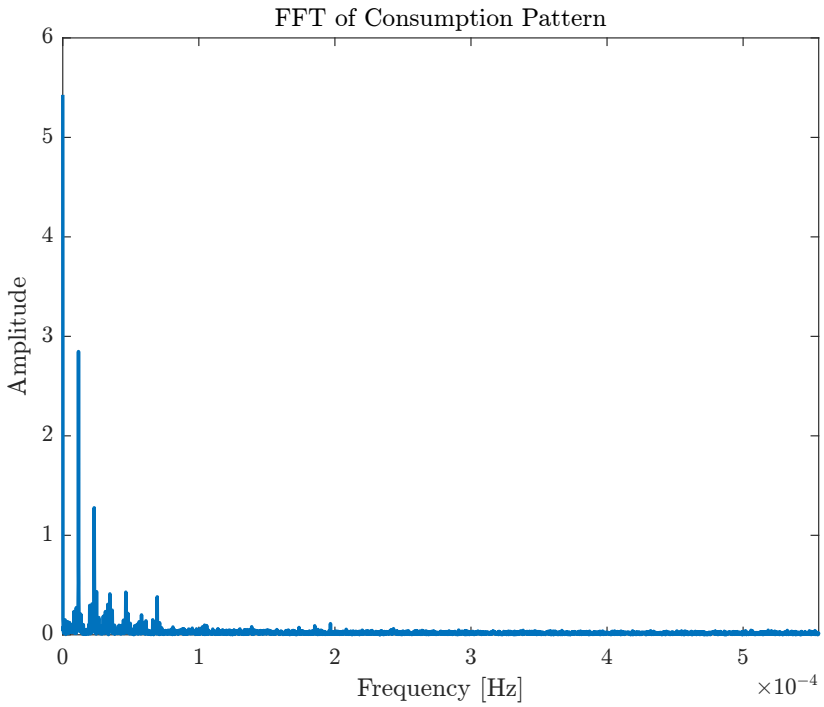


Figure 4.5: Fast fourier transformation for consumer data.

The action value function is then approximated as a linear combination of a weight vector and a radial basis function feature vector, and a linear combination of a weight vector containing Fourier coefficients and feature vector according to $\mathbf{x}(t)$ in Eq. (4.15):

$$\hat{Q}\Big(h(k), t(k)\mathbf{w}_{a(k)}\Big) = \mathbf{w}_{a(k)}^{\mathsf{T}}\mathbf{x}\Big(h(k), t(k)\Big) = \begin{bmatrix}\mathbf{w}_{a_{RBF}} \\ \mathbf{w}_{a_{FS}}\end{bmatrix}^{\mathsf{T}} \begin{bmatrix}\mathbf{x}_{RBF}(h) \\ \mathbf{x}_{FS}(t)\end{bmatrix} \tag{4.16}$$

The weight vectors update rule is:

$$\begin{aligned}\mathbf{w}_{a(k)} \leftarrow \mathbf{w}_{a(k)} + \alpha \Bigg[J(k+1) + \gamma \min_{a'} \Big(\mathbf{w}_{a'}^{\mathsf{T}} \mathbf{x}(h(k+1), t(k+1)) \Big) \\ - \mathbf{w}_{a(k)}^{\mathsf{T}} \mathbf{x}\Big(h(k), t(k)\Big) \Bigg] \mathbf{x}\Big(h(k), t(k)\Big)\end{aligned} \tag{4.17}$$

Algorithm 3 presents function approximation Q-Learning algorithm applied to the water distribution network shown in Section 3.1 with continuous height and time, and discrete actions.

Algorithm 3 Continuous height and time, discrete action Function Approximation Q-Learning algorithm

1: **Input:** $\alpha \in [0,\ 1]$, small $\epsilon > 0$, and $0 \leq \gamma < 1$, number of Radial basis functions, center placements, width and number of Fourier Series basis functions
2: **Initialise:** weight vectors $\mathbf{w}_a$ for all discrete actions (e.g. $\mathbf{w}_a = 0$), and states $t(1)$ and $h(1)$
3: **for** k=1,2,... **do**
4: Take action according to a ϵ-greedy policy:

$$a(k) = \begin{cases} \underset{a' \in \mathcal{A}}{\arg\min}\ \hat{Q}\Big(h(k), t(k), \mathbf{w}_{a'}\Big) & \text{exploitation with probability } 1-\epsilon \\ \text{Random action from } \mathcal{A} & \text{exploration with probability } \epsilon \end{cases}$$

5: Observe next states $h(k+1)$, $t(k+1)$ and cost $J(k+1)$
6: Update $\mathbf{w}_{a(k)}$ according to:

$$\mathbf{w}_{a(k)} \leftarrow \mathbf{w}_{a(k)} + \alpha \Bigg[J(k+1) + \gamma \min_{a'} \Big(\mathbf{w}_{a'}^{\mathsf{T}} \mathbf{x}(h(k+1), t(k+1)) \Big) - \mathbf{w}_{a(k)}^{\mathsf{T}} \mathbf{x}\Big(h(k), t(k)\Big) \Bigg] \mathbf{x}\Big(h(k), t(k)\Big)$$

7: **end for**

Chapter 5

Simulation

This chapter presents simulation results obtained on the system presented in Chapter 3 and will cover both Tabular Q-learning and function approximation Q-learning. Each section begins with a hyperparameter search using a sweep, followed by the simulation results using those parameters. Approximation is first done only on height, then also on time.

All simulation results obtained use the barrier function presented in Section 4.1 with left and right side zeros at 20 and 40 respectively. The height discretisation is done according to Eq. (4.5), with $h_{lower} = 15$ and $h_{upper} = 55$. Time is discretised into 24 states, one for each hour in a day. Q and $\mathbf{w}$ are initialised to 0.

Most of simulations are done where the last 10% of iterations are without learning or randomness, we will refer to this area as the cutoff. This gives a more explicit indication of performance[1].

Note we have setup the simulation to match laboratory values, but time scales are not similar.

[1] E.g. a large epsilon can result in higher cost while learning, but lower cost *after learning*, this behaviour is hidden without the *cutoff*

5.1 Tabular Q-learning

The following section presents results obtained when applying tabular Q-learning to the water distribution nentwork presented in Chapter 3. Action values are updated according to Algorithm 1.

5.1.1 Hyperparameter Sweep

Sweeps are performed over singular parameters, ideally combinations of all parameters should be swept, but we deem this too resource demanding. Performance is evaluated by looking at a moving average of the cost with a large window size, a large window smooths out randomness and clears up tendencies. The moving average is calculated by finding the mean of neighbouring elements from a point, a window size of 5760 is used determining how many neighbouring points are included. In the beginning and end where there are not enough neighbouring points to fill the window, it is truncated to include as many neighbouring points as possible.

Sequential sweeps over single parameters are bound to have some iterative nature going back and forth. Initial values are found by doing a fast and coarse sweep of all parameters to find suitable values, these initial sweeps are not shown here as they receive no further thought than to get started.

Initial values are shown below in the order they are swept:

$$\alpha = 0.5 \qquad \gamma = 0.9 \qquad \epsilon = 0.1$$

Fig. 5.1 shows how the moving average at different α values converge, see Fig. 5.2 for zoomed plots of beginning and end of the sweep. The long middle section for which it looks like all values have settled, is included to see if some values ultimately converge to a lower cost if it is allowed to see many random actions. The only clear tendency is that higher values converge faster. In the final 10%, no values appear better within the margin of randomness. Higher values are more susceptible to the random actions performed leading up to the cutoff, meaning there is a slightly larger spread of their final convergence cost.

This tendency has been further explored by simulating values of $\alpha = 0,1$ and $\alpha = 0.9$ 50 times for 50000 iterations, see Fig. 5.3. Higher values clearly show a higher variance. We also think the wider spread is due to higher α values *bouncing* around the optimal minimum of the cost function. Similar to finding the minimum of a convex problem, a large step size will sometimes hit the minimum, but usually *bounce* around in its vicinity. Looking at the lowest and highest values, both seem to fluctuate more than the more moderate values in the middle part of Fig. 5.1, we suspect this is because the higher values over react, and the lower values are too slow to correct the influence of bad random actions.

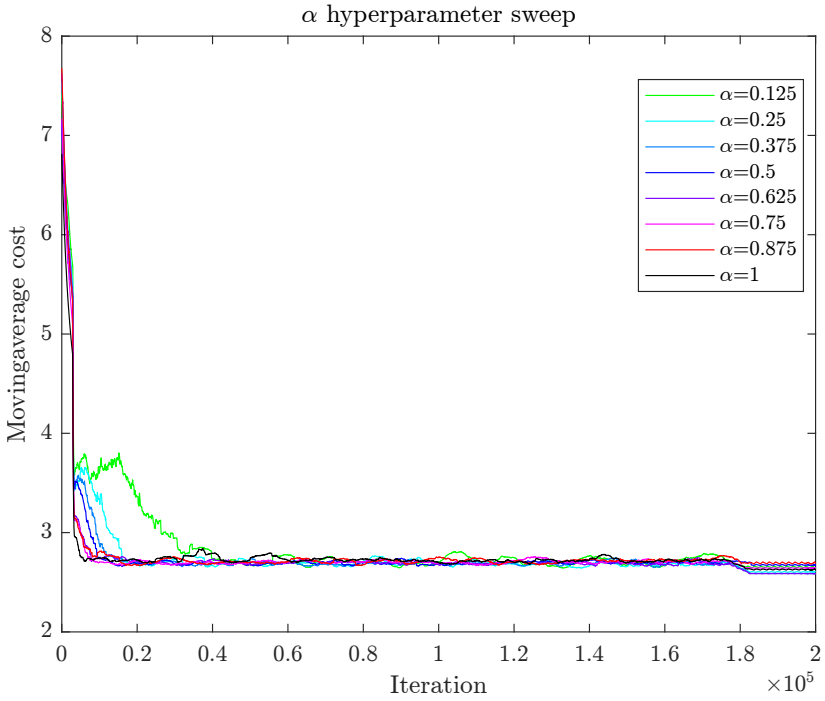
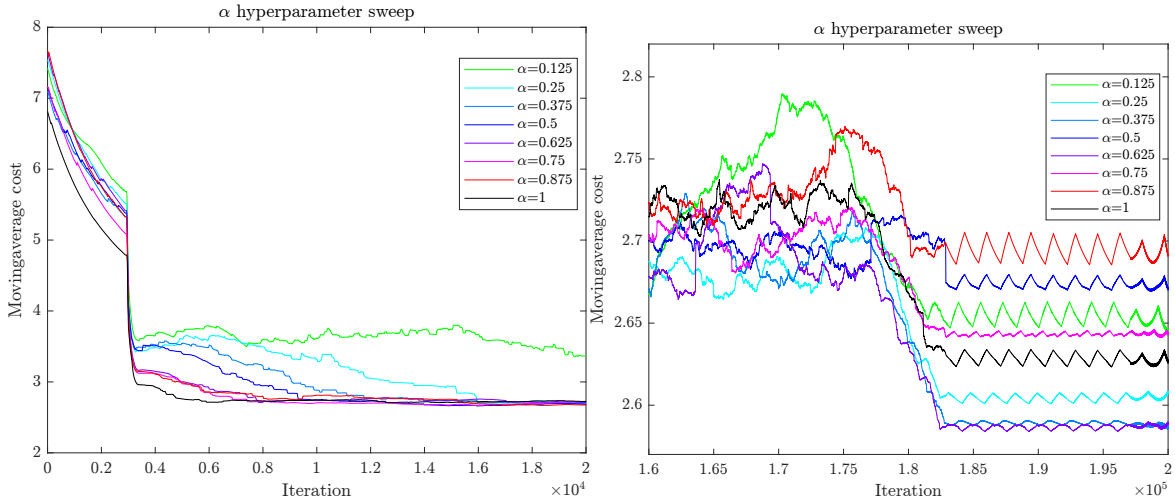


Figure 5.1: Alpha sweep using 200000 iterations.

We will select a value of $\alpha = 0.9$, to gain fast initial convergence, and the downside of a wide spread is not big enough to merit a smaller value.

A hypothesis on why such a large α is optimal is that there are no stochastic behaviour in this problem, any action at a certain state will always lead to a specific next state. As such, less filtering is needed, as the increment in action value is always in the direction toward the *true* value.

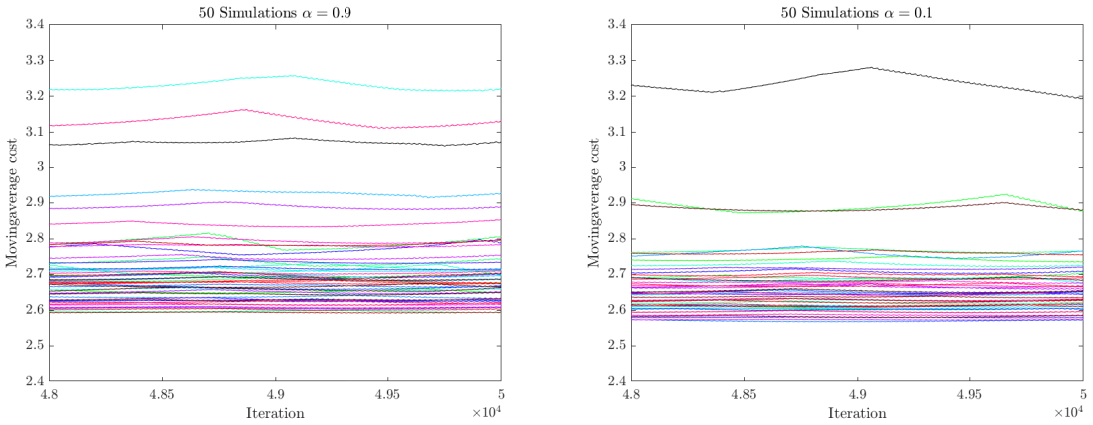
The cutoff is presented as an aid in evaluation here, but could be implemented, while also gaining the benefit of a narrow spread, by reducing α to a much smaller value, either gradually or at some threshold e.g. after initial convergence.



(a) Initial convergence.

(b) Cutoff convergence.

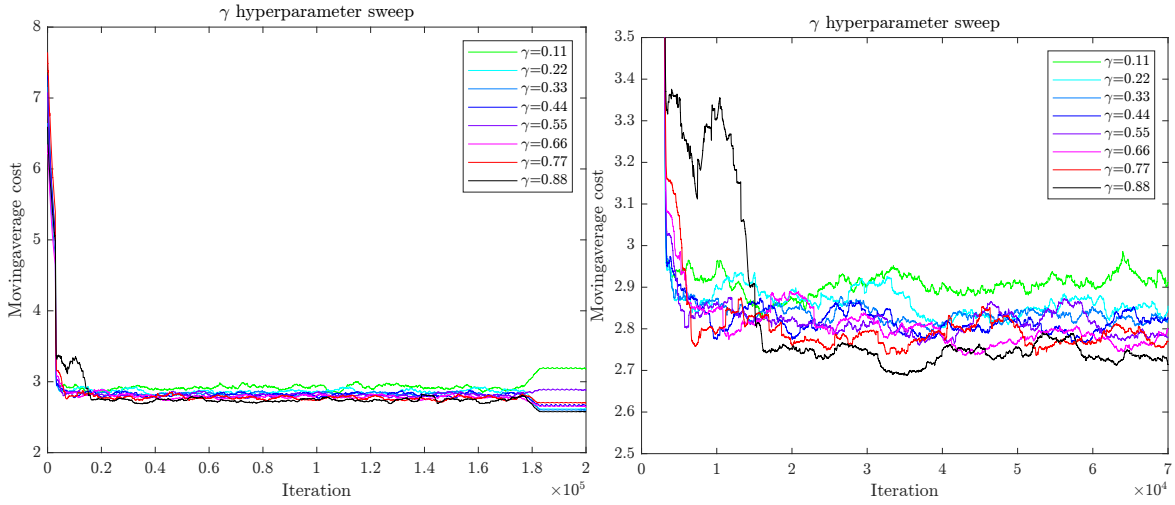
Figure 5.2: Zoomed images of left and right side of Fig. 5.1 focusing on initial convergence and cutoff convergence.



(a) Cutoff convergence of 50 simulations using $\alpha = 0.9$.

(b) Cutoff convergence of 50 simulations using $\alpha = 0.1$.

Figure 5.3: Comparison of cutoff convergence of 50 simulations using $\alpha = 0.9$ and $\alpha = 0.1$. Notice colours do not carry meaning, as Matlab cannot distinguish 50 plots with different colours.



(a) Gamma sweep full plot.

(b) Initial convergence.

Figure 5.4: Gamma sweep showing the full plot (a), and a zoom focused on initial convergence (b).

Fig. 5.4 shows gamma sweep. There is also one clear tendency here, higher values converge slower, but settles lower once they have converged. The final moving average convergence value after the cutoff of all γ values fall within the margin of randomness, and therefore this area receives no further consideration.

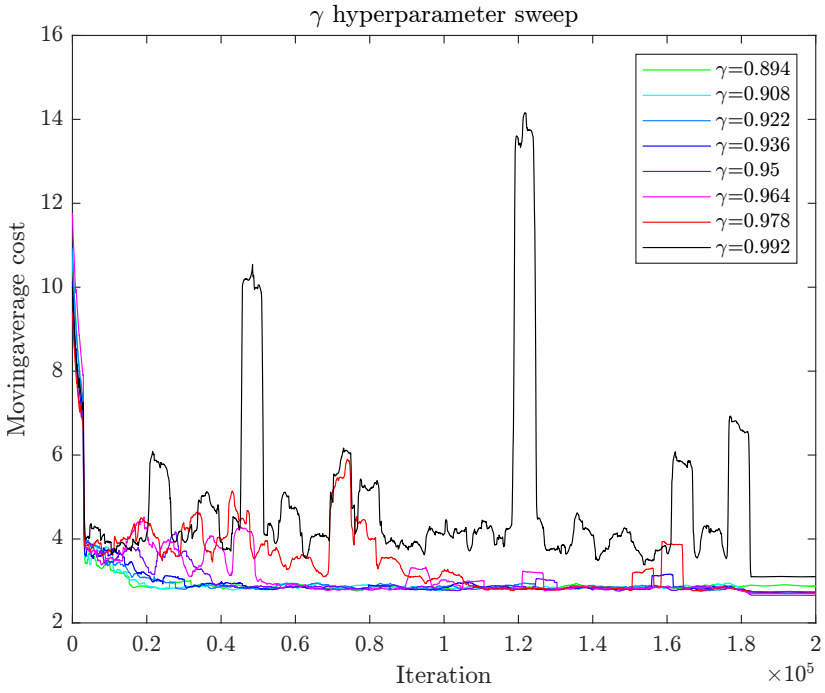


Figure 5.5: Additional narrow gamma sweep, including a finer sweep of high gamma values.

Fig. 5.5 shows a narrow sweep of the very high values. A narrow sweep is performed for high values because the best value was expected to be in this range. Sweeps on previous iterations of the simulations scripts showed good performance with high γ, however, this is no longer valid. Higher values clearly slow convergence even further, to the point of not converging at all. Final convergence does not improve. We choose a value of 0.77, since it is desired to converge as low as possible as fast as possible. Therefore we choose the highest values that does not increase convergence time a lot.

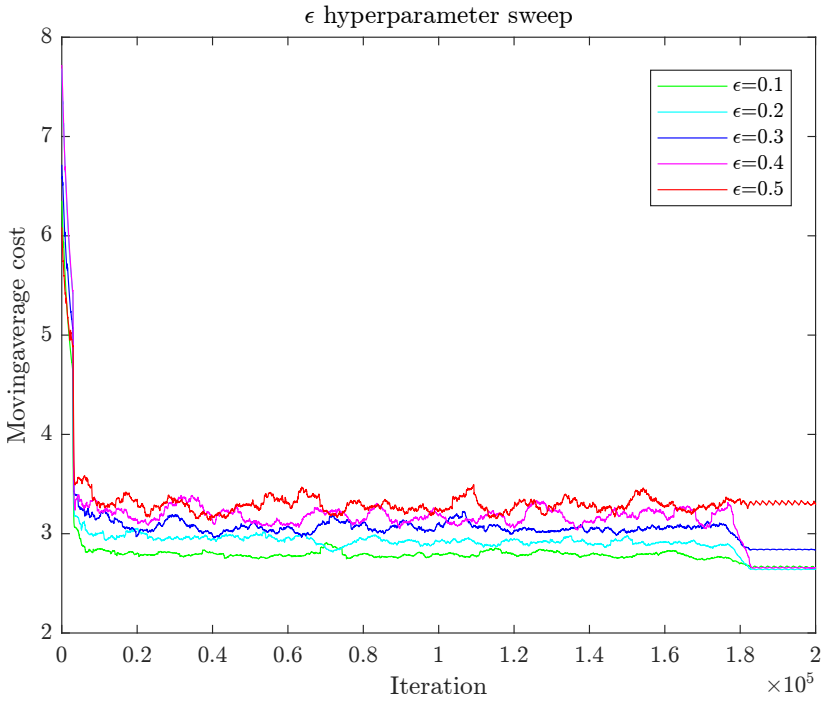


Figure 5.6: Epsilon sweep of relevant values between 0.1 and 0.5.

Fig. 5.6 shows epsilon sweep for relevant values of ϵ ranging from 0.1 to 0.5, higher values means we are more random than not. A small ϵ standout as both yielding the smallest training cost, and lowest final convergence. This is expected from previous sweeps. Higher epsilon means the random influence leading up to the cutoff is greater[2], resulting in a greater spread of final convergence. Note $\epsilon = 0.1, 0.4$ converge to similar final cost, showing that the high epsilon only increases the spread, not necessarily resulting in worse performance. The higher training cost is an obvious result of more random actions. We choose $\epsilon = 0.1$.

Final parameters are:

$$\alpha = 0.9 \qquad \gamma = 0.77 \qquad \epsilon = 0.1$$

[2]Remember we choose a quite high $\alpha = 0.9$.

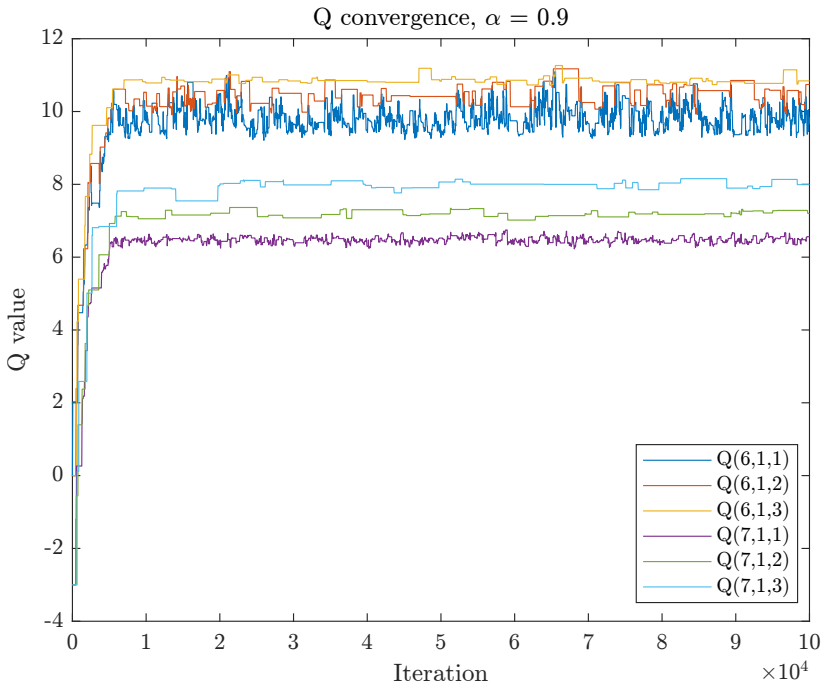


Figure 5.7: Q convergence. Values for two different heights are shown, the lower height $h = 7$, has an offset to add visual clarity to the plot.

As a few final remarks, we consider the convergence of the Q values, and with it an extra parameter evaluation on $\alpha = 0.2$ and $\alpha = 0.9$. The extra evaluation is included to present different water level behaviour, as seen in this paragraph. Q value convergence is a candidate for an alternative evaluation scheme instead of moving average cost for sweeps. Fig. 5.7 and Fig. 5.8 show Q convergence for $\alpha = 0,9$ and $\alpha = 0.2$ respectively. Notice that the *chaoticness* of each signal is an indication of how often that specific state-action combination appears. Obviously the higher α value results in more oscillations, a lower value was included for comparison. Additionally, Fig. 5.9 shows 5 simulations of reservoir water level plotted in the same plot. Here we see the impact of oscillations from high α are minor, the variance of the convergence of $\alpha = 0.2$ is smaller, but with more and greater outliers. Both share the same mean.

These results are more related to specific systems, and *good* behaviour for that system. Sometimes less water level variance could be desired over fast convergence, therefore choosing a lower α.

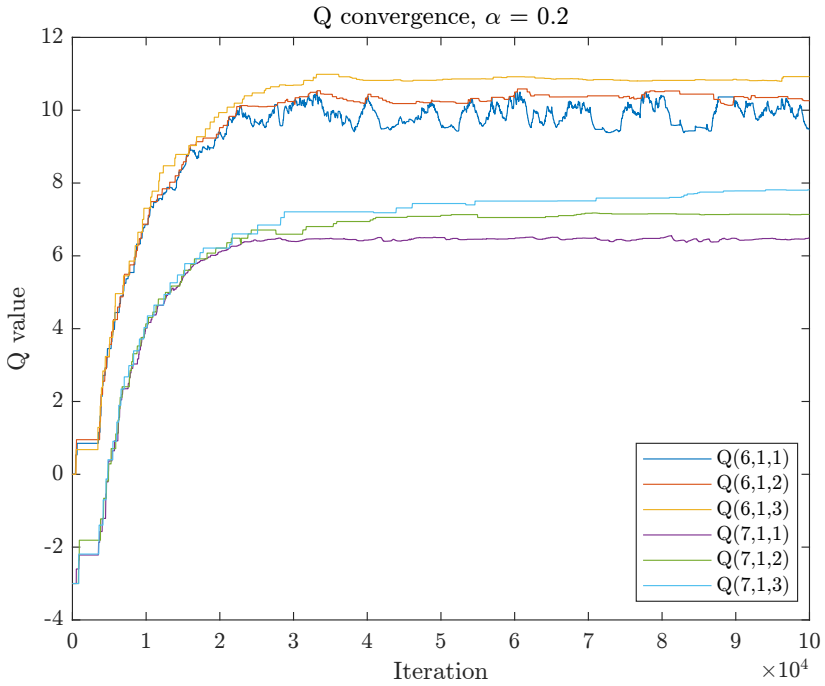
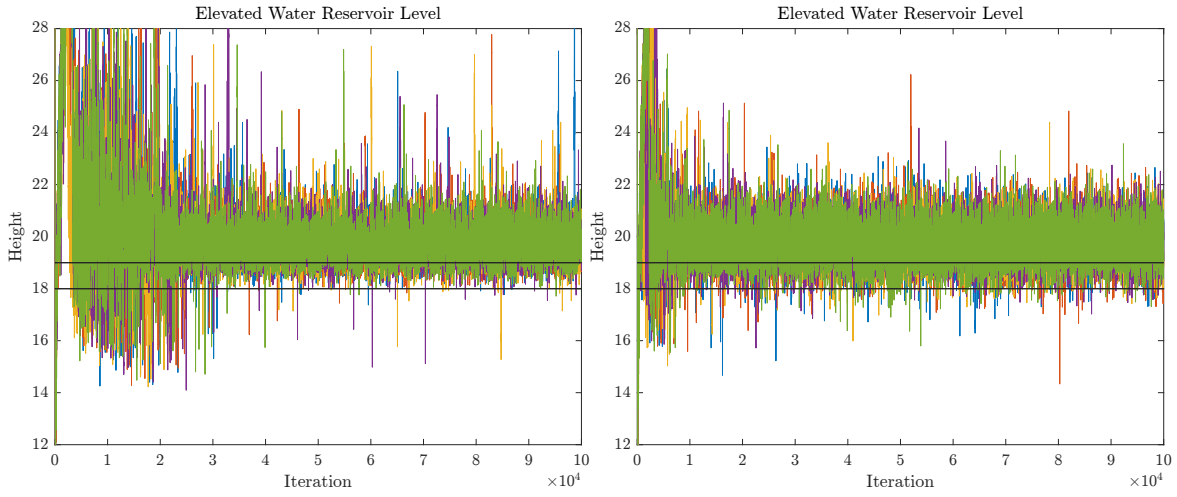


Figure 5.8: Q convergence for reduced α. Offset for visual clarity included again.



(a) 5 simulations using $\alpha = 0.2$

(b) 5 simulations using $\alpha = 0.9$

Figure 5.9: 5 simulations in each plot used for comparison of high and low alpha, and the resulting behaviour of reservoir water level. Black lines are inserted at heights 18 and 19, to make differences more visible. Each colour is 1 of the 5 simulations.

5.1.2 Tabular Results

Here we show simulation results using the swept parameters.

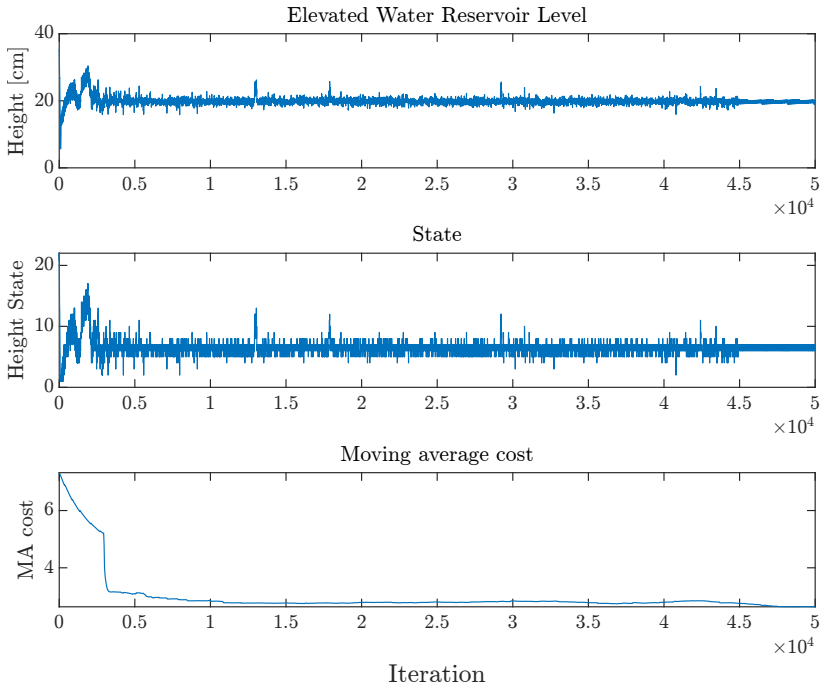


Figure 5.10: Tabular Q learning results, showing elevated water reservoir level, state and moving average cost.

Fig. 5.10 show reservoir water level, height state and the moving average cost. Water level converge nicely around the barrier at 20cm. The states are clearly explored from the bottom first, due to Matlab always choosing the first option when results of the *minimum function* are equal, resulting in descending behaviour when initialising at 0. Let the term *turnaround* denote to the ability to stop ascending-exploration of the height state, and begin to descend toward optimal values, before exploring the entire height space. This happens at iteration 1800 in Fig. 5.12(a). The *turnaround* is unexpected, but results in much faster convergence as many states are unexplored. Some theories of why this happens is discussed in Chapter 7.

Fig. 5.11 shows a 48 hour period after the cutoff. This clearly shows the dynamics between demand flow and pump flow (actions).

Fig. 5.12 shows a zoomed image of the reservoir water level in the beginning and end of the simulation. In Fig. 5.12(a), notice the fast descend in the beginning (iteration 100), this is because action 1 is always chosen. Q is initialised to 0 for all state action pairs, and Matlab's min function always selects the first entry when min evaluations are equal. The rapid ascend back to 15, is because the state never changes, all states below 15 are set to state 1 due to height discretisation. Above the height of 15, dynamics slow down as new

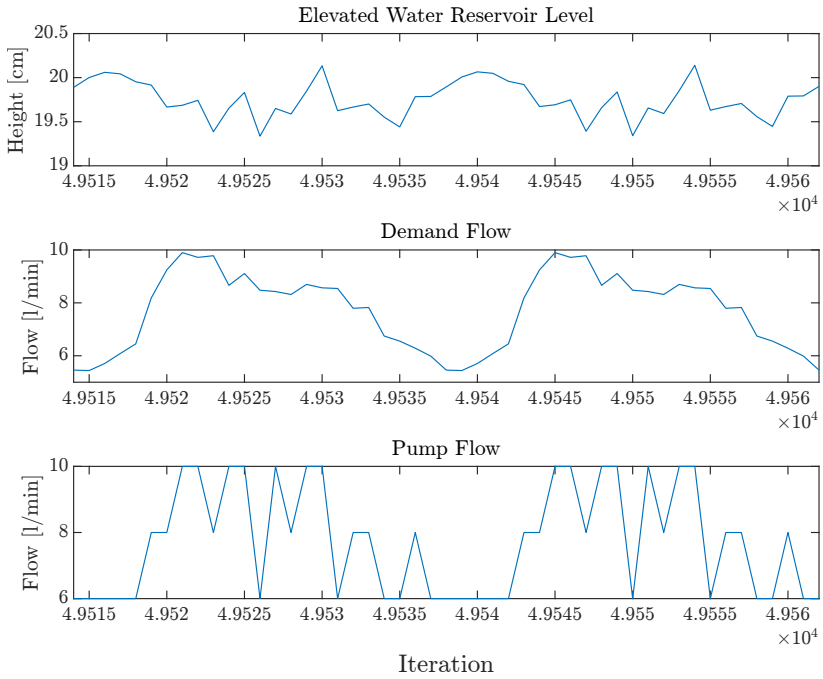
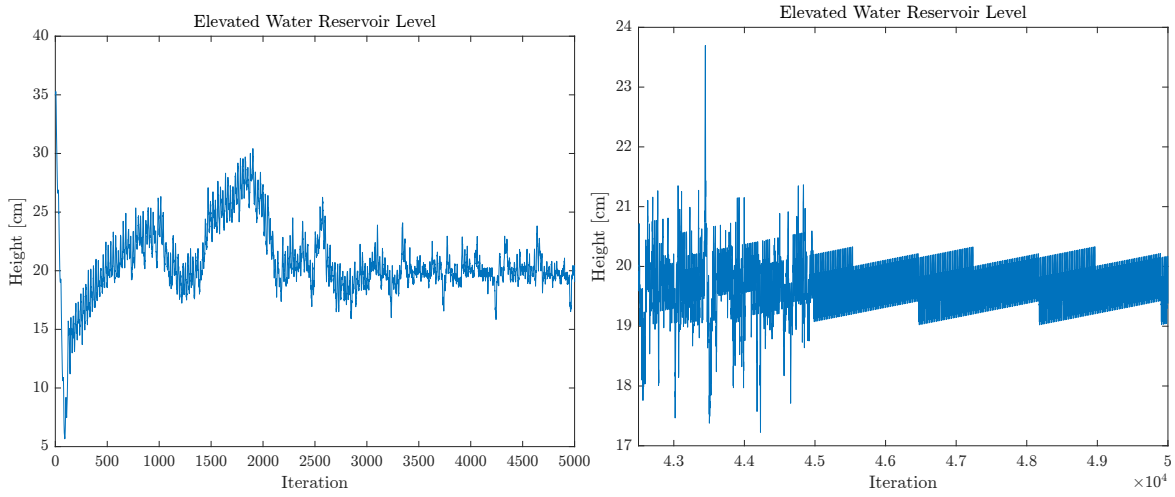


Figure 5.11: 48 hours tabular results,taken after the cutoff. Shows relationship between height, demand- and pump flow.

states are encountered at every height.

The zigzag pattern in Fig. 5.12(b) is a result of slightly scewed demand balancing. The optimal policy simply results in a slight amount of excess water for every 24 hour period. If pumps or demand is altered slightly a smooth non-zigzag result can be obtained.



(a) Initial convergence.

(b) Cutoff convergence.

Figure 5.12: Elevated water reservoir level results focused on initial convergence (a), and cutoff convergence (b).

5.2 Continuous Height, and Discrete Time and Action Q-Learning

As explained in Section 4.3.1 the action-value function can be approximated as:

$$\hat{Q}\Big(h(k), \mathbf{w}_{t(k),a(k)}\Big) = \mathbf{w}_{t(k),a(k)}\mathbf{x}\Big(h(k)\Big) \tag{5.1}$$

This section present simulation results obtained when applying continuous height, and discrete time and action Q-learning, to the water distribution in Chapter 3.

Weight vector is initialised to zero and updated according to Algorithm 2 and radial basis functions are placed as explained in Section 4.3. Hyper-parameter sweep is done similarly as in Section 5.1.1, but plots are not shown, because tendencies cannot be seen in a single plot. Instead many simulations are done with varying combinations of hyper parameters, to build an intuition about system behaviour. The chosen parameters are:

$$\alpha = 0.4 \qquad \gamma = 0.95 \qquad \epsilon = 0.1$$

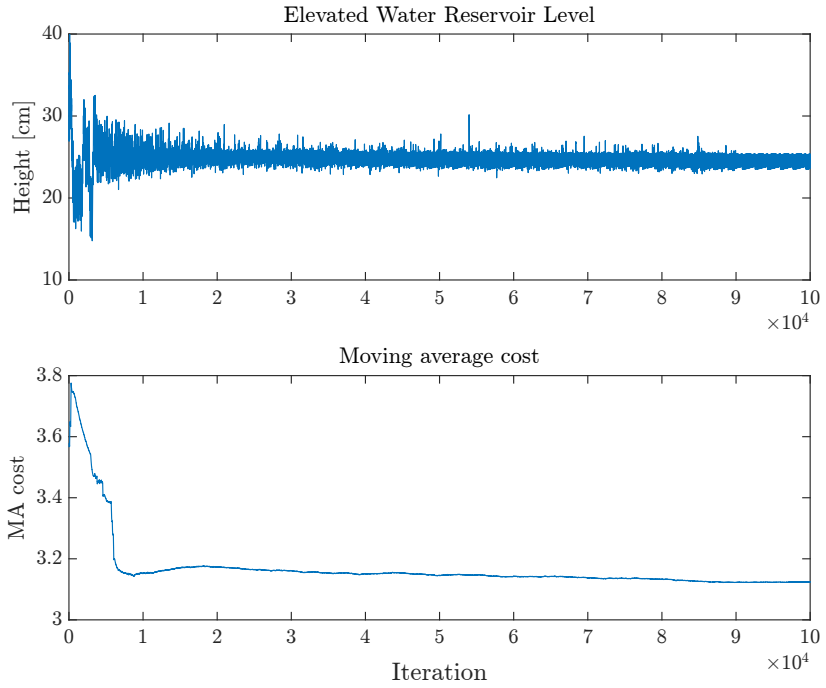


Figure 5.13: Continuous height approximation Q learning results. Showing elevated water reservoir level and moving average cost.

Fig. 5.13 presents water level in the elevated water reservoir and moving average of the cost function. Most noticeable is the tendency to settle much higher than the lower barrier,

this is discussed further in Chapter 7. The benefit of generalisation is very apparent as the algorithm does not have to explore as aggressively to find the optimal policy.

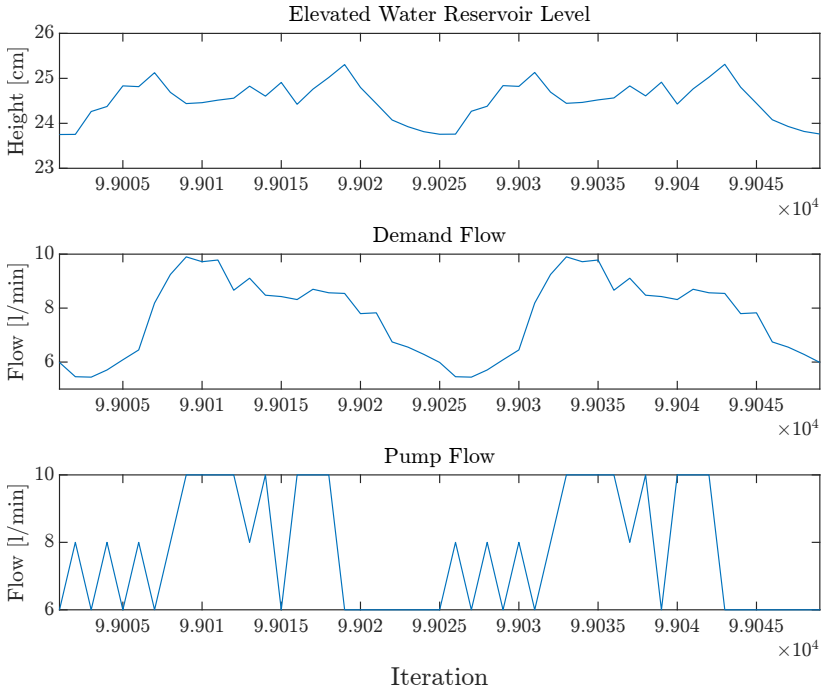


Figure 5.14: 48 hours continuous height results, taken after cutoff. Shows relationship between height, demand- and pump flow.

Fig. 5.14 shows a 48 hour period after the cutoff. It shows water levels similar to tabular method, pump flow (actions) are also similar, but clearly not the same. Although action value functions are obviously not exactly the same for both methods, this shows some proof of the concept that there exists several optimal policies (pump flows) which yield the same value function.

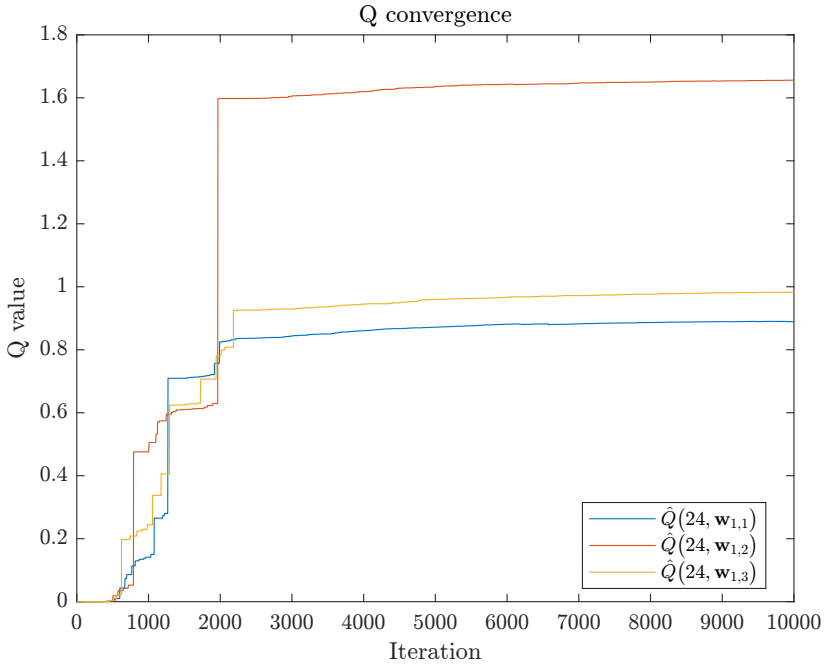


Figure 5.15: Q convergence for continuous height, at $h = 24$, $t = 1$.

Fig. 5.15 presents convergence of approximated action value for height $h = 24$cm and time $t = 1$. Notice the x-axis is zoomed on iterations $0 - 10000$ to show the initial convergence. We can see the impact of generalisation by noticing that in the beginning (iteration 0 - 2000) of Fig. 5.13 the height is explored around $h = 20$, however, Fig. 5.15 showing the Q value at $h = 24$ shows changes at these iterations.

5.3 Continuous Level and Time, Discrete Action Q-Learning

As explained in Section 4.3.2 The action value can be estimated using function approximation in both the height and time:

$$\hat{Q}\Big(h(k), t(k), \mathbf{w}_{a(k)}\Big) = \mathbf{w}_{a(k)}\mathbf{x}\Big(h(k), t(k)\Big) \tag{5.2}$$

Height approximation is done using weights and centers as previously. And time is approximated as presented in Section 4.3.2. The weight vector is initialised and updated according to Algorithm 3. As previously, hyper parameter are swept but not shown. The following parameters are used:

$$\alpha = 0.2 \qquad \gamma = 0.95 \qquad \epsilon = 0$$

Notice $\epsilon = 0$, it was found that any randomness causes worse performance, and it is therefore not used. We present thought on *implicit exploration* caused by Q initialisation and oscillating demand in Chapter 7.

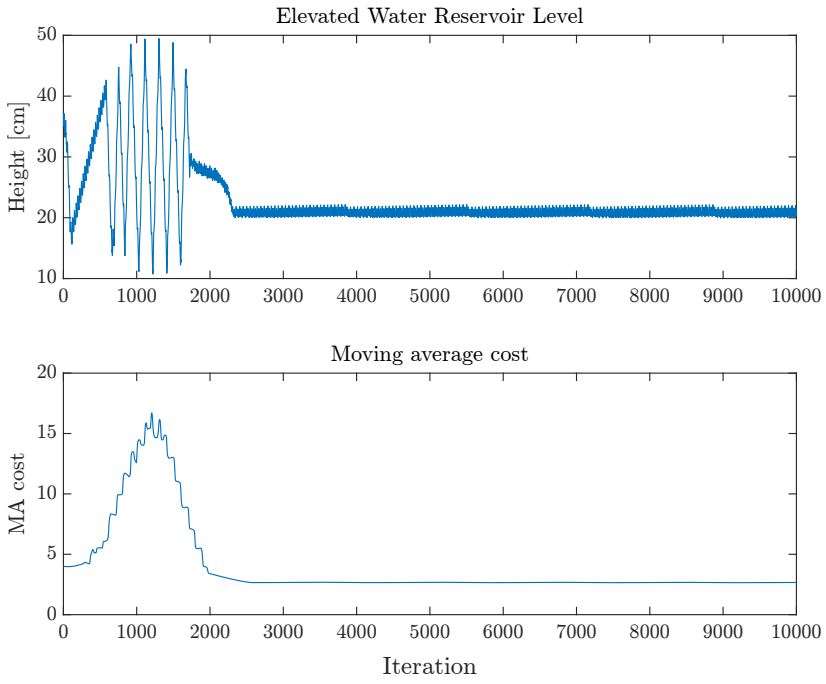


Figure 5.16: Continuous height and time approximation Q learning results. Showing elevated water reservoir level and moving average cost.

Fig. 5.16 presents water level in the elevated water reservoir and moving average of the

cost function. Here we see a very defined initial learning, before converging to a final convergence value almost immediately. Notice that the final convergence is smoother than usual since $\epsilon = 0$. The first *linear* learning explores the same state space as height approximation method, but then explores much broader. A very fast convergence around iteration 2200 is achieved, showing the power of generalisation. The final convergence value is also similar to that of the tabular method (better than height approximation). Notice the tendency to converge higher than the optimal value around $h_\star \approx 20$, which also was visible for height approximation, almost appears around iteration 2000. This tendency is even more apparent for different α values.

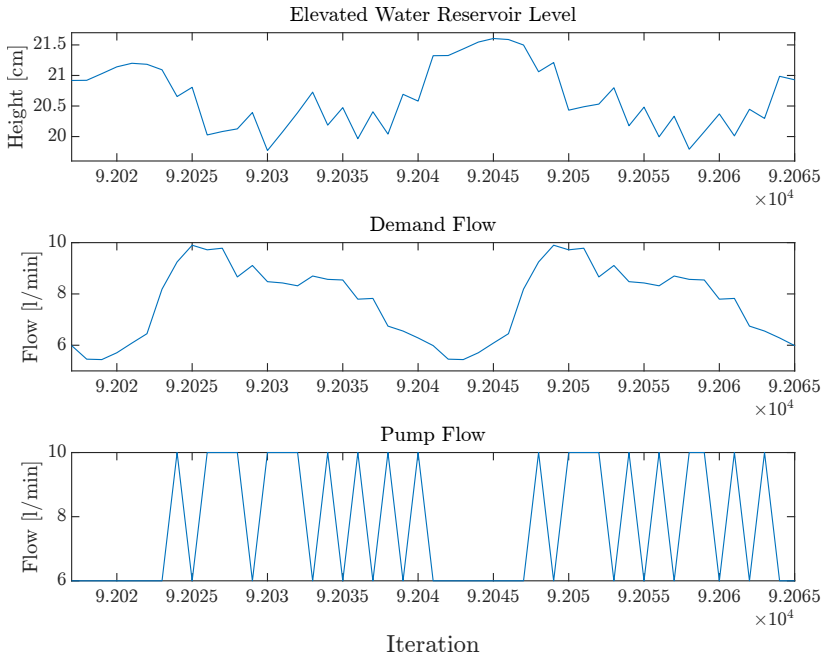
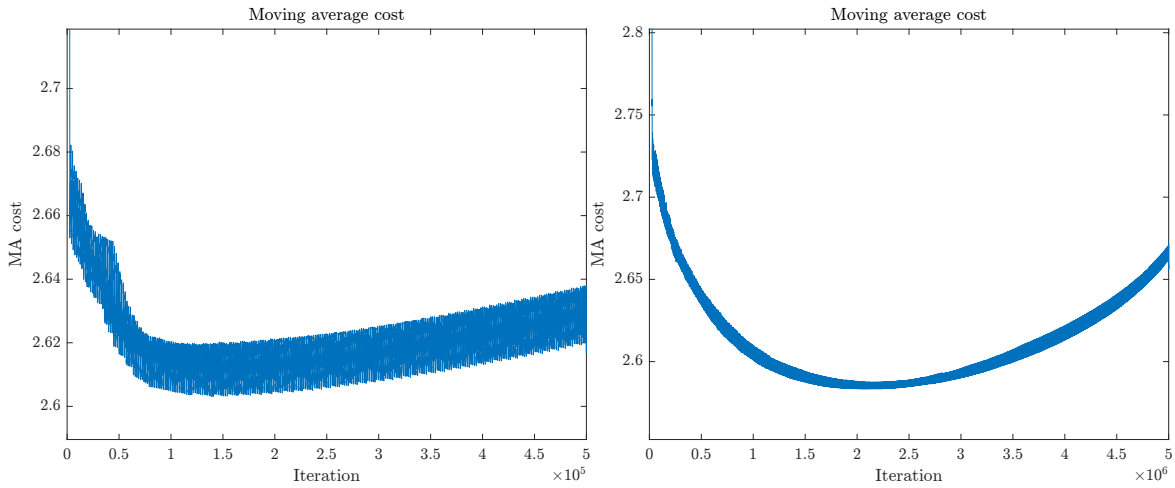


Figure 5.17: 48 hours continuous height and time results, taken after cutoff. Shows relationship between height, demand- and pump flow.

Fig. 5.17 shows the same comparison as for the other methods. Notice there is no cutoff since this simulation does not include randomness. This means the pump profile is different for each 24 hour period, this tendency is also apparent if the cutoff is included, which would usually force equal action profiles for each 24 hour period. It seems generalising on time gives the algorithm more freedom, we do not grasp the full context to this *feature*.

Although this method converges very fast, an issue is found if the simulation is extended. Fig. 5.18 shows moving average costs for long simulations using $\alpha = 0.2$ and $\alpha = 0.01$, both clearly diverge. We suspect this is caused by *deadly triad* [**Sutton2020**]. The deadly triad describes the danger of instability and divergence when combining the three following element: Function approximation, bootstrapping and off-policy training (Q-learning). On policy learning like *SARSA* could be used to circumvent this issue.



(a) Divergence of $\alpha = 0.2$.

(b) Divergence of $\alpha = 0.01$.

Figure 5.18: Comparison of moving average cost beteween $\alpha = 0.2,\ \ 0.01$.

5.4 Method Comparison

Function approximation has shown to be a very powerful tool, but also dangerous. Complexity increases drastically when approximating, intuition and knowledge about the algorithm's behaviour is much harder to obtain. We will make this comparison from the perspective of lab and real world implementation. In this context, double approximation is too parameter sensitive and likely to diverge, to be considered for real world implementation. The tabular method, showing similar convergence as height approximation, and even better final convergence, requires operation dangerously close to the bottom of the tank. Height approximation show good performance and does not need to explore dangerous areas, therefore this method is the most suited for implementation at this stage of development. The performance of this method is further tested as it is applied to our water distribution network test setup in the following chapter.

Chapter 6

Laboratory Results

This chapter presents test results obtained when applying continuous height, discrete time and action function approximation Q-Learning to the Smart Water Infrastructure Laboratory shown in Fig. 6.1, which is used to emulate the small scale water distribution network presented in Section 3.1.

Due to long convergence times the control method implemented is based on the final approximated action value function obtained in Section 5.2. This means control will be based on simulation learning, such that we do not have to learn in the laboratory. Because we have to wait for water level changes to happen, along with long convergence times at thousands of iterations, learning would take weeks in the real world. The performance results when using simulated training, depends on the similarities between test setup and simulations.

There are known differences between test setup and simulation, simulations use a model which is not entirely accurate to simulate water level changes. To reduce experiment length, an hour is made equivalent to one minute in the test setup. This gives behaviour similar to a changed flow or changed cross sectional area of the reservoir, meaning the change of water level will be different.

The test setup does not allow separate pumps to run independently. However, they do have speed control. Therefore rotational speeds will be set equally on two pumps at three different speeds, resembling three different actions. The speeds are chosen to yield combined flows matching simulation flows as well as possible.

Test details:

- **Run time:** 144 minutes - representing 144 hours/6 days in real life. Simulations use time increments of one hour, we use the same concept for laboratory test, however, one hour will be equivalent to 1 minute. Among many things, this means a new

actions is chosen every minute, and water level changes due to that actions, occur over the next 60 seconds.

- **Consumer demand:** Periodic over a 24 minute interval. 24 minutes (96 samples) represent 24 hours of a real day. Demand corresponds to normalised profile showed in Section 3.2. Demand is sampled every 15th second. Flows resembling simulation flows are obtained by controlling a variable valve with a PI-controller with the consumer profile described in Section 3.2 used as a reference.

- **Time state:** As in simulation, time is discretises into 24 states Section 4.2 and Section 4.3.1.
- **Weight vector w**: is obtained from simulation in Section 5.2 and have dimension: $\mathbb{R}^{3\times24\times10}$ corresponding to 10 radial basis functions with centers placement described in Section 4.3.1

- **Actions:** are percentage pump rotational speeds, which are matched to fit simulation flows through trial and error before running results. As such actions are equivalent to rotational speeds of 50%, 65% and 100%.
- **Initial conditions:** initial water level of 38cm at $t = 1$.

Figure 6.1: Picture of the AAU SWIL.

6.1 Laboratory Setup

Fig. 6.2 shows a diagram of the setup of laboratory modules. The pump station supplies water to the system, all water flows through a pipe module, both to add pipe length for a more realistic setup and to utilize the module's sensors. Water flows both to and from the elevated water reservoir. Flow to the consumer is regulated using a controllable valve in the module. Water is circulated using auxiliary pumps in the pump station. For a more detailed diagram of the setup, see Simulink setup in Appendix A.3. This diagram show simulink setup resembling the figure in this chapter, including specific names and numbers of the used sensors and pumps.

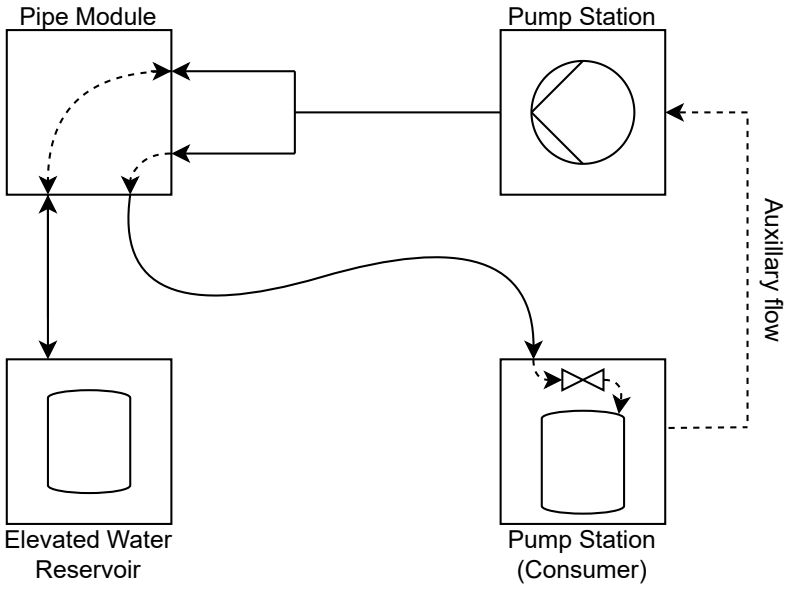


Figure 6.2: Diagram showing simple laboratory module setup.

6.2 Results

Fig. 6.3 present the water level in the elevated water reservoir over a timespan of 144 minutes. Just like in Chapter 5 a periodicity is visible in the water level, which is expected due to the periodicity in the consumer demand shown in Fig. 3.3. Water level is nicely reduced to the optimal value around 25 as expected similarly to simulation results in Section 5.2. The fact that the water level settles so far above the barrier in both simulation and laboratory is discussed in the following chapter.

The simulated mean water level for the last five periods is 24.55 and for laboratory results the mean is 25.25, which we consider to be similar behaviour. The water level variance seems greater for laboratory results, likely because the demand is sampled four times more often, and demands are not as accurate as in simulation, since the demand is controlled via a valve which has a settling time.

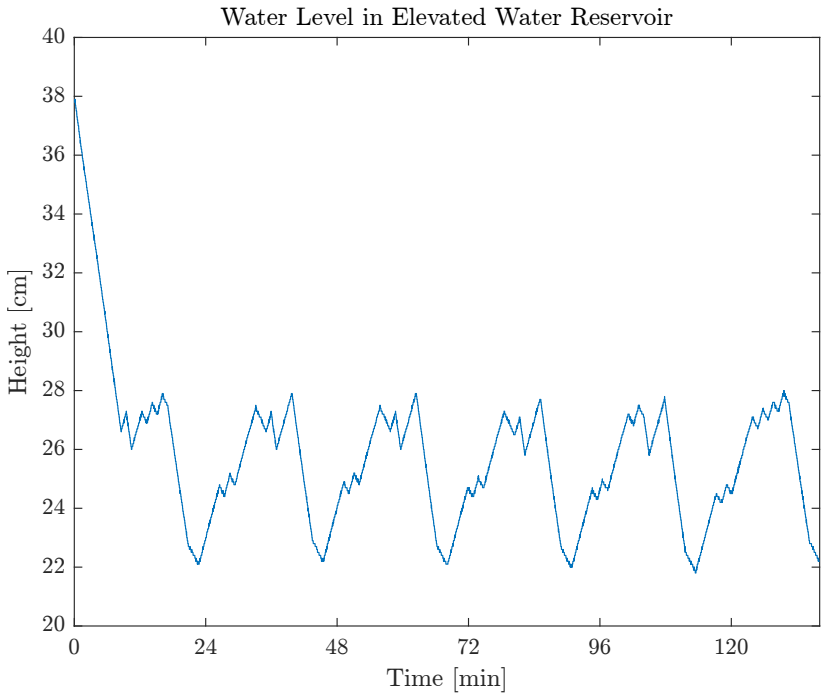


Figure 6.3: Water level in elevated water reservoir, showing a period of 5 days.

The difference to simulation results was expected based on the already known differences between test setup and simulation. One of the advantages to reinforcement learning mentioned in this project is its ability to adapt to changes in the environment. However, this implementation does not include learning, due to Simulink implications. We are certain that a full reinforcement learning method with learning and exploration can be implemented in Simulink, but was not within scope of this project.

Furthermore, for the same reasons, this implementation does not include exploration. The nature of the Simulink issues are from storing previous values while the simulation is running. We do not see any reason that this should not be doable in Simulink if more time was available.

Fig. 6.4 shows actions chosen by the reinforcement learning agent. A periodic pump profile is visible as expected. Notice the lowest rotational speed is used as long as there is too much water in the reservoir.

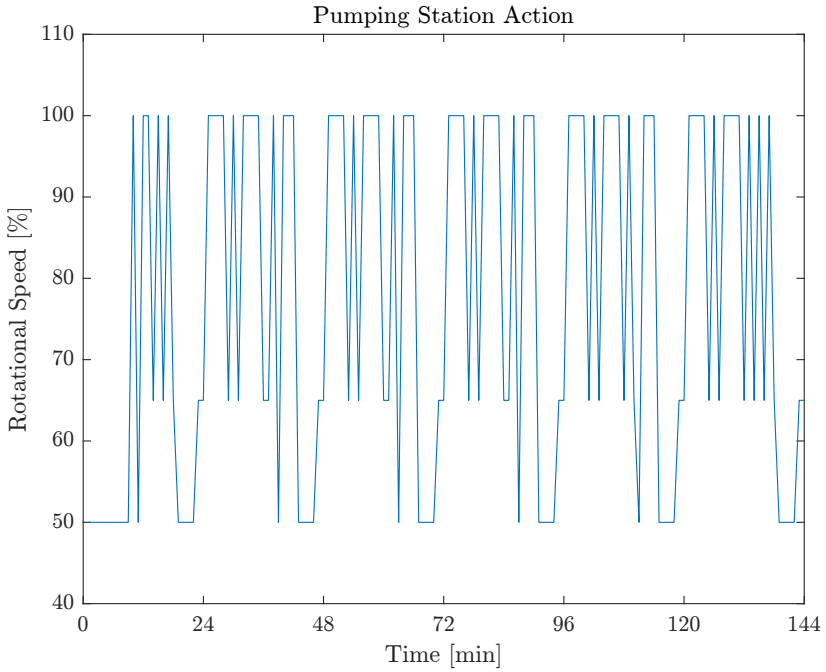


Figure 6.4: Pump profile showing rotational speed (actions) over time.

Chapter 7

Discussion

This section presents a discussion related to results throughout this project. We present hypothesis and thoughts on different aspects of results.

7.1 Demand Balancing

Due to the limited number of action the demand has to be matched to fit the pump flows. If this matching is not done nearly perfect results are diverging. An unbalanced demand, means large periods of the demand are greater or smaller than the highest or lowest pump flow respectively. Without randomness reinforcement learning handles the imbalance, it essentially requires that you always follow the best actions. Without grasping the full context or reasoning to this problem, randomness sometimes results in an action which starts a loop of bad actions, resulting in divergence. From many hours of simulation, heuristic evidence, show balancing demands make the algorithm much more robust toward randomness.

In the results shown in Chapter 5 and Chapter 6 the pump flow are:

$$q(k) = \{6 \;\; 8 \;\; 10\} l/min$$

and the demand flows are:

$$d(k) \in [5.44,\ 9.9]\ l/min$$

A potential solution could be to add more pumps to the pumping station, which would result in more actions and a better coverage of varying demand profiles. This in turn increases the size of the table in tabular Q-learning and adds extra weight vectors in function

approximation Q-learning with discrete actions. With perspective to the real world, where additional pumps come at a cost, and demands cannot be controlled. The option of running 0 pumps should be included and pumps should be slightly too powerful equivalent to demands being balanced slightly too low.

7.2 Hyperparameters

As mentioned previously sweeping hyperparameters sequentially is not ideal. In future work these sweeps should be performed more thoroughly by evaluating combinations of parameters. Any change of one hyperparameter changes the optimal selection of the other parameters. This could not be done within the scope of this project, due to coding- and simulation time required to obtain the results.

The *tendencies* that are often refereed to in sweeping results are subject to randomness, especially for the height approximation method. In reference to the *law of large numbers* tendencies would require a large amount of simulation before being applicable. Results in this project are simulated multiple times, but not enough. This induces some ambiguity in the results of the sweeps. In future work, the ideal sweep incorporates both combinations of parameters and satisfy the law of large numbers.

In a thorough sweep we also suggest considering the convergence of Q values as an evaluation metric. We have not used this method enough to express its advantages, but there seem to be useful information to be achieved in addition to moving average cost evaluation.

Finally a few thoughts on the *cutoff* and its implementation. The cutoff was used in this project as an evaluation tool, but could be implemented by reducing ϵ and α either after a iteration threshold or continuously. This would unlock the optimal behaviour, instead of adding some suboptimal behaviour from learning and exploration. If changes to the environment are expected, these parameters could be increased to detect and learn these changes, and then reduced back to cutoff values, continuing optimal behaviour.

7.3 Premature Convergence

The tabular method shows the ability to converge without exploring the entire state space. This is against expectations, but we present a two part explanation. First we place emphasis on the oscillating demand, which allows the algorithm to see a specific state multiple times without knowing how to *stay* in that state. Secondly each time a state is seen, the algorithm gets one step closer to learn the optimal action in that step. We will consider the scenario where ascending states are explored, heading toward the top of the reservoir. As we explore the *0 cost* area of the barrier function, the only cost assigned to a state is primarily from the action (flow) itself, and the pressure induced from reservoir water

level 3.7. As the tank level rise, the increments of the action cost increases, and these increments increase a lot more when the barrier is hit. When the difference in increments are big enough, the algorithm will choose the lower action increasingly many times in a row before the cost has incremented far enough that its cost is higher than another action's. Eventually, the number of *low-flow* actions in a row, will result in the water level decreasing further than what is expected from the demand at that time. This unexpected decrease results in the algorithm seeing a state that it has already fully learned, and it will deliberately select the optimal action and end the upward exploration. From here on, the algorithm descends down without any reason to go back up.

We assume this theory also applies to the height function approximation method's ability to just barely explore the upper barrier.

7.4 Extending Radial Basis Center Placement

To give the best initial conditions without prior knowledge, the sum of radial basis function should equal 1. If centers are placed at the edges of the range, the sum near the edges will be less than 1, meaning lower emphasis is put on the very edges, this is intuitively not a smart starting point for learning. Edges could be argued to be the most important and interesting areas. Therefore centers where tried extended outside the range, resulting in a similar sum over the entire range.

This was thought to increase performance for all algorithms, although we now believe the increased performance was from more favourable placement of all centers, which happened indirectly when extending the centers on the edge to maintain uniform placement.

The context to extending centers could not be examined sufficiently within the scope of this report, and we would like to research this topic further in future work.

7.5 Barrier Overflow

Height approximation results settle much higher than tabular method, this is because generalisation from radial basis functions extends the cost of the barrier further than its beginning. The entire middle part of the barrier gives 0 cost, but with generalisation the cost of entering the barrier will overflow into the heights that should otherwise belong to the *0 cost area* of the barrier. Fig. 7.1 shows the radial basis function with center nearest the barrier beginning.

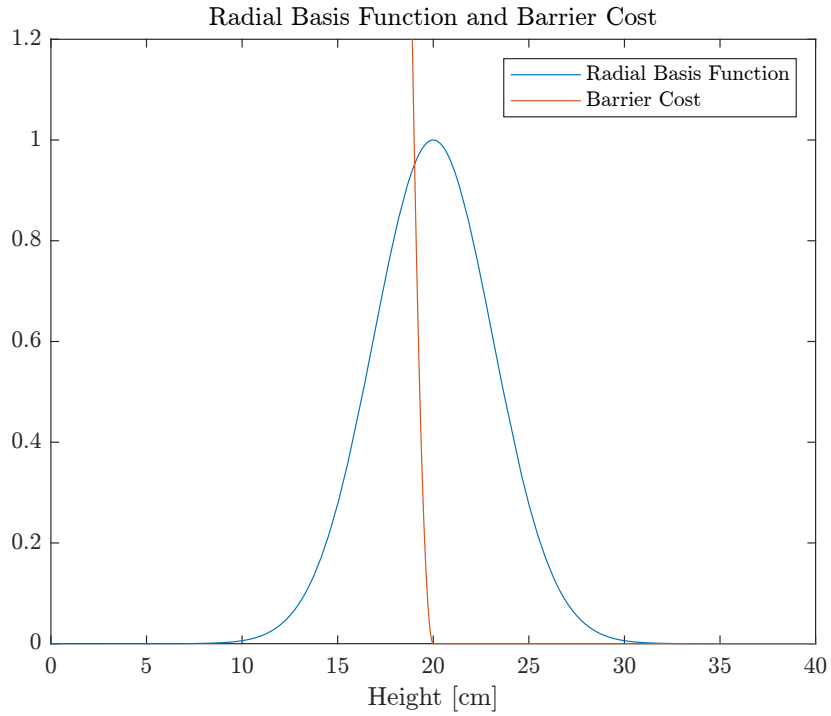


Figure 7.1: Beginning of barrier and radial basis function with center closest to barrier beginning.

Fig. 7.2 shows the results of removing the feature with center near the beginning of the lower barrier. The initial convergence now reaches much closer to the expected optimal height, but convergence in that area is much slower. Because points here are far away from any feature, their impact are weighted very low.

Fig. 7.3 shows the radial basis functions with the center closets to barrier beginning removed. This clearly visualises the reason that overflow is reduced, but at the expense of having very little feature impact in that area causing learning to be very slow.

Another approach to removing the center, is to add more narrow radial basis functions around the barrier beginning. This would cause the generalisation to be more accurate, meaning less overflow, without sacrificing learning.

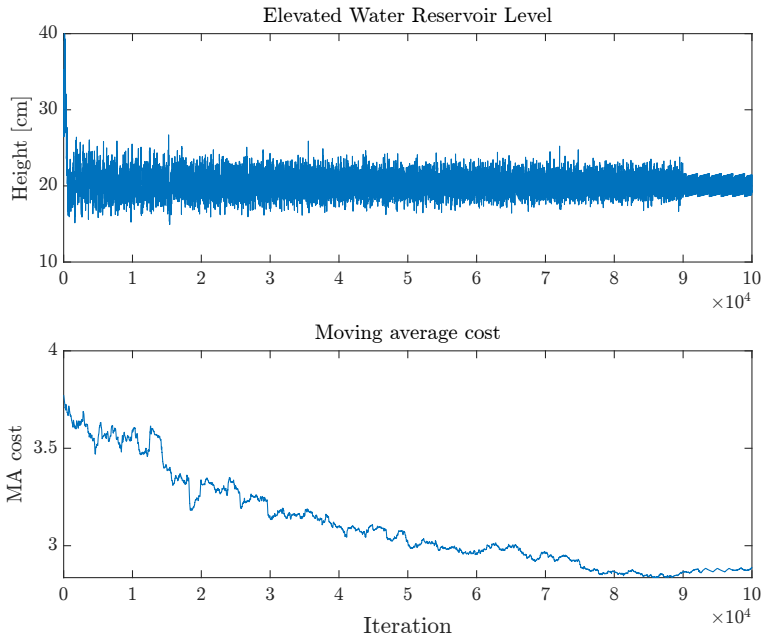


Figure 7.2: Continuous height results with feature near lower barrier removed.

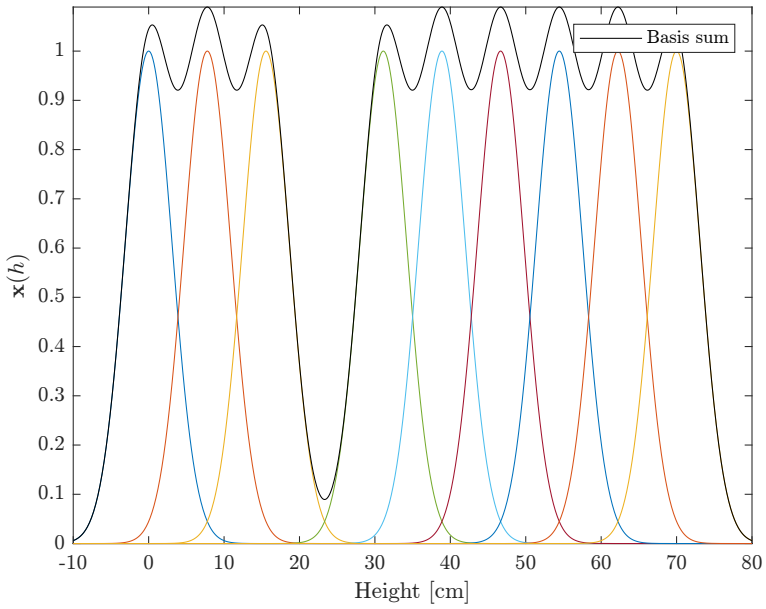


Figure 7.3: Radial basis functions with center near barrier removed.

Essentially, the number of radial basis functions, and placement of their centers can and should be included as a hyperparameter.

7.6 Implicit Exploration

full state function approximation results showed seemingly very nice convergence without randomness, although it ultimately diverges. This initial nice behaviour, even without randomness is possible because of implicit exploration from other elements. The oscillating demand very clearly results in a similar oscillation in state, this is seen in all water level plots. Initialisation of Q values and weights at 0, means whenever a cost is assigned to one action, all other actions are better until they have shown to yield even higher cost. The combination of these is very clear in the full state function approximation results Fig. 5.16, showing a very large range of states being explored before convergence.

7.7 Generalisation to other Water Distribution Networks and Real World Implementation

this section presents thoughts on the real life implementation of reinforcement learning. We have seen all method be quite sensitive to hyperparameters, and performance relies on demand balancing. We therefore doubt the method is easily applicable to all networks, some redesign and tuning will certainly be necessary. However, the laboratory results are performed without a finely tuned algorithm, and shows that decent performance can be achieved with direct implementation. Finally, the tuning of a reinforcement learning algorithm is likely much less tedious than modelling a complete water distribution network, if it is even possible.

We have mentioned the minimisation of energy as a goal of this project. We cannot present evidence of this goal, but believe there is good reasoning to believe reinforcement learning can reduce energy usage. The algorithm clearly settles to a optimal behaviour in regards to the formulated cost function. Whether this is actually optimal would require modelling of the system, and heavily depend on design of cost function and barriers. There is also the element of safe learning, the act of learning, without imposing dangerous states on the system. In this project the only indirect safety feature is imposed by the barrier function. Here we see the trade off between optimality and safety, where the beginning of the lower barrier determines how low the algorithm settles, directly influencing the energy needed to satisfy demands.

Although approximation methods are showing safe learning, due to generalisation. We would not guarantee safe behaviour from these methods, before real implementation, methods to guarantee safe learning needs to be found. Further more, we remark the importance of demand balancing, which in real implementation can only realistically be done by carefully designing the pump setup, as demands cannot be controlled.

Chapter 8

Conclusion

As suggested in Section 1.2 the objective of this project was design and implementation of a optimal and model free controller for a water distribution network. Three reinforcement learning methods have been applied and compared; tabular Q-learning, continuous height, and discrete time and action function approximation Q-learning, and continuous height and time, and discrete actions function approximation Q-learning. When comparing these three methods it was found that the best control method for real life implementation was continuous height, and discrete time and action function approximation Q-learning due to its ability to generalise, giving better behaviour than the tabular method. The method is also superior to continuous height and time, discrete actions function approximation Q-learning, since this method shows unpredictable and diverging results, and also seems so be extremely hyperparameter sensitive compared to the two other methods. Even though we observe diverging result the method is an interesting candidate for further work since simulations show superior initial performance when comparing moving averages of the cost function.

Although the examined methods are not compared to existing water distribution network control methods, relative to pumping station energy consumption, we feel confident in saying that the formulated cost function ultimately guarantees minimal energy consumption.

Laboratory testing indicates that continuous height, and discrete time and action function approximation Q-learning is a viable control method that works well, when training is done in a simulated environment. Training in a real life environment was not within the scope of the project, but the implementation lays the foundation for future work and has given us a strong belief that reinforcement learning methods are well suited for real life implementation and can make a difference on both existing and new systems. We note that reinforcement learning control methods developed should include some kind of safety aspect such that safety constraints of systems are not violated.

Appendix A

Continuous height discrete time and action Q-Learning MATLAB Script

A.1 New state script

Listing A.1: New state and action script

```
function [newstate] = nextstate(state,Q,epsilon,h,demand,timediscretisation)

[val,action] = min(Q);          %Choose action based on min action-value
%Epsilon greedu policy
if rand < epsilon;
    action = randi(3);
end
flow = [6 8 10]; %Set pump flows relatve to action index
A = 0.3;
%Update water leevel in elevated water reservoir
actualh = state(1) + 0.06*(flow(action)-(demand(state(2))))/A;
%Increment time
if state(2) < timediscretisation;                    %increment time
    newt = state(2) + 1;
else
    newt = 1;
end
%Update new states
newstate = [actualh, newt, action flow(action)];
end
```

A.2 Weight update MATLAB script

Listing A.2: Weight update script

```
clear all; close all; clc; warning off;
for ep = 1:1
    %Input values of alpha, gamma and epsilon
    alpha = 0.4;
    gamma = 0.95;
    epsilon = 0.1;

    timediscretisation = 24;    %Set number of discrete times
    h=70;                       %Set height of tank [cm]

    Q = [0 0 0];                %Initialise Q

    time = 1;                   %Initalise time at k=1
    actions = 3;                %Input number of actions available
    weights = 10;               %Input number of weights (RBFs)

    %Create a weight vecotr for each action and time of length w
    w = zeros(actions,timediscretisation,weights);

    %Place centers uniformly
    for c = 1:weights
        centers(c) = (c-1)*(h)/(weights-1);
    end

    sigma = 3.2;        %Set width RBFs

    %load demand data
    demand = load('ConsumptionProfile');
    demand = demand.dcon*5.77+5;     %Demand balancing

    S0 = [h/2 1]';          %set initial state S0(h,t)
    for f = 1:weights       %Initialise feature to k=1 based on h0
        feature(f) = exp((-(abs(S0(1)-centers(f)))^2)/(2*sigma^2));
    end
%-------------------------------------------------------------------------
    maxstep = 2000000; %Number of iterations
    for episode = 1:1
        state = S0;
```

```
39          for ii = 1:maxstep
40              %Calculate new h based on action chosen and increment time
41              newstate = nextstatefunctionapprox(state, Q, epsilon,h,demand,
                    timediscretisation);
42              action = newstate(3);
43              actualh = newstate(1);
44              flow = newstate(4);
45              Qold = squeeze(w(action,time,:))'*feature';     %store Q value
46              oldfeature = feature;                           %store features
47              time = newstate(2);                             %update time
48
49              %Cost function
50              P = 1000*9.82*0.1*actualh;
51              Cost = flow*P*1.667*10^-5 + QuadBarrierFuncActualHeight(actualh);
52
53              %update features at new height
54              for f = 1:weights
55                  feature(f) = exp((-(abs(newstate(1)-centers(f)))^2)/(2*sigma
                        ^2));
56              end
57
58              %update Q values with new features
59              for n = 1:3
60                  Q(n) = squeeze(w(n,time,:))'*feature';
61              end
62
63              %update weights, special case for time = 1
64              if time == 1
65                  w(action,timediscretisation,:) = squeeze(w(action,
                        timediscretisation,:))' + alpha*(Cost + gamma*min(Q) -
                        Qold).*oldfeature;
66              else
67                  w(action,time-1,:) = squeeze(w(action,time-1,:))' + alpha*(
                        Cost + gamma*min(Q) - Qold).*oldfeature;
68              end
69
70              %update Q with new weights and new features
71              for n = 1:3
72                  Q(n) = squeeze(w(n,time,:))'*feature';
73              end
74
```

```
75              %Set new states
76              state = newstate(1:2);
77          end
78      end
79  end
```

A.3 Simulink setup

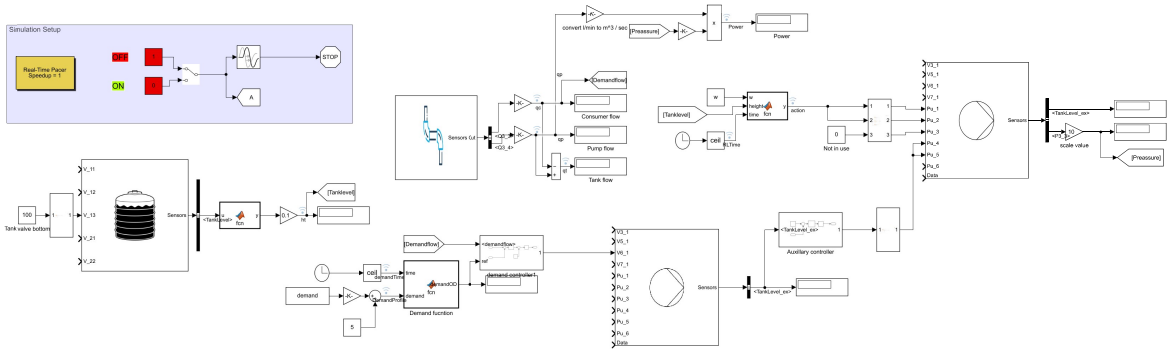


Figure A.1: Simulink setup