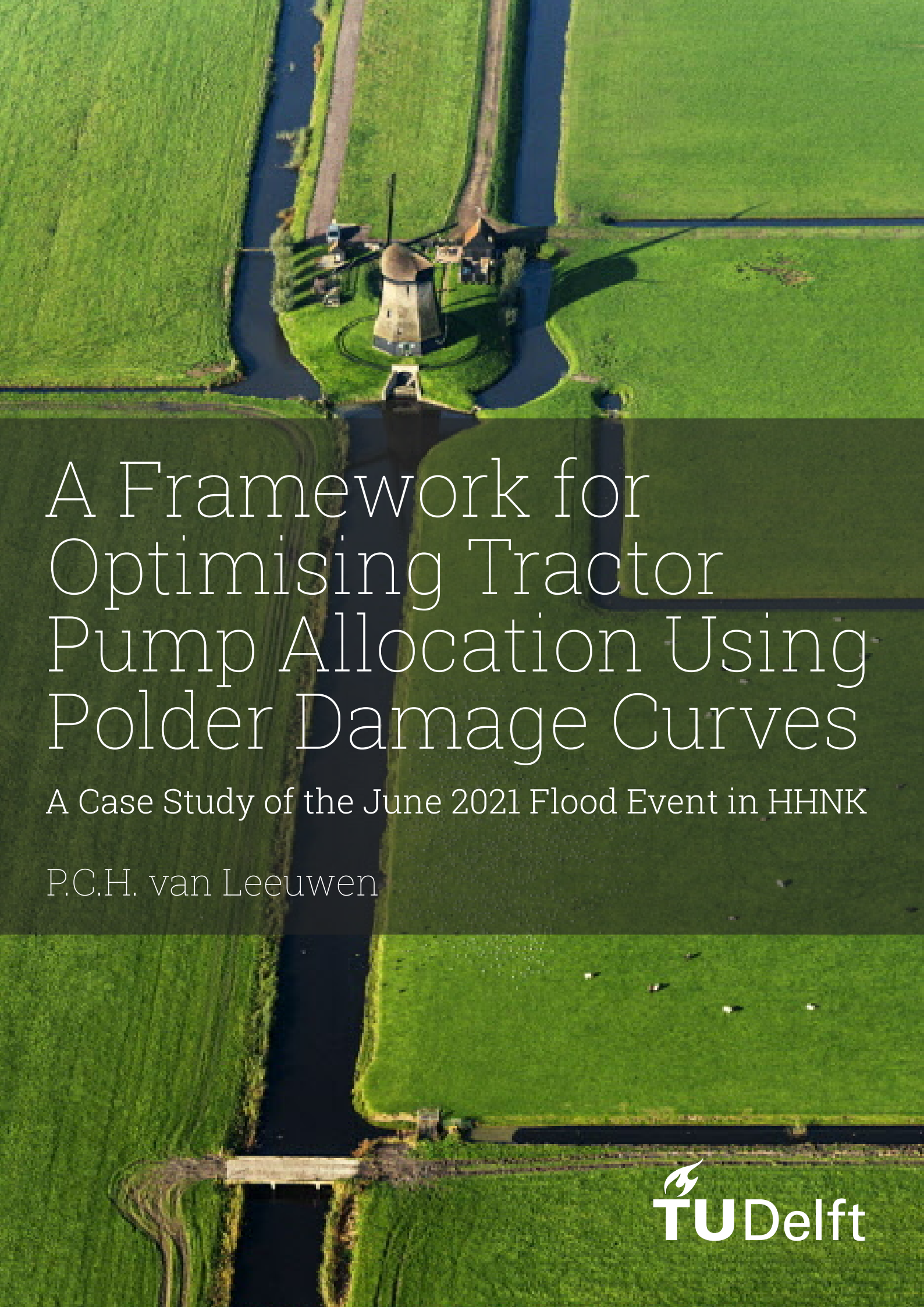


A Framework for Optimising Tractor Pump Allocation Using Polder Damage Curves

A Case Study of the June 2021 Flood Event in HHNK

P.C.H. van Leeuwen

TU Delft

A Framework for Optimising Tractor Pump Allocation Using Polder Damage Curves

A Case Study of the June 2021 Flood Event in HHNK

by

P.C.H. van Leeuwen

to obtain the degree of Master of Science
at the Delft University of Technology,

Student number:	5430593
Project duration:	June 1, 2024 – May, 2025
Thesis committee:	Dr. ir. O.A.C. Hoes
	Dr. Ir. M.W. Ertsen
Supervisors HHNK:	Dr. Ir. P.E.R. van Leeuwen
	A.E.M. van Oostrum

Cover: Nederland, Westbeemster - Mischa Keijser, Beeldunie (2019)
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

With this thesis, I not only conclude my master's degree but also a long journey in which I completed my HBO, bridging program and master's. Throughout this journey, I have continuously pushed myself to learn more and deepen my understanding of water management.

My thesis has been a turbelent period. With some illness, as well as being forced out of my home in Delft, which meant relocating back to my parents in the eastern part of the Netherlands. This made communication with my supervisors a bit more challenging, as dropping by for a quick discussion is not so easy if you are a 100 kilometers away.

I am grateful to my supervisors at TU Delft, Dr. Ir. O.A.C. Hoes and Dr. Ir. M.W. Ertsen, for their guidance, support, and insights. Hoes' approachable attitude, practical perspective, and ability to consider the broader context of the study made this a truly collaborative effort. I am also very thankful to my supervisors at the water board, L. van Oostrum and Dr. Ir. P.E.R. van Leeuwen. In particular, my namesake Van Leeuwen spent countless hours discussing and advising for improvements on the optimization models.

Besides the official support, I would like to thank several persons from my personal life as well. First of all, my girlfriend Lotte, for helping me through this turbulent period. I have been a student too long in her (and my own) eyes, even though she has never complained about this. Secondly, my parents, for their understanding of a student living at home again after 10 years. And lastly my sister, for her help with the discussion and study context.

Furthermore, I would like to thank the CCB team. During my time at the water board, I witnessed the (official) formation of the team and saw how they successfully accomplished this. They always made me feel welcome, even though my presence was limited by distance.
A special thanks also goes out to J. van der Lingen for showing the tractor pumps at the depot in Anna Paulowna, T. Berends for assisting with KNMI data scraping, J. van Ursum for helping me with the (rather complicated) governmental jargon, and everyone else I met during my time at the water board.

P.C.H. van Leeuwen
Puiflijk, June 2025

Summary

Climate change has led to an increased frequency of extreme rainfall events, creating significant challenges for polder regions where water has to be pumped out actively. Tractor pumps stand out as a quick, flexible measure to improve polder discharge capacities. However, in extreme rainfall events, where the need for tractor pumps exceeds available supply, strategic decisions must be made on where to deploy them. This requires a data-driven methodology to support decision making. To achieve this, flood modelling, damage assessment and pump allocation optimization are combined in this study. The study aimed to create a framework that combines these in one integrated model and allocates a limited set of tractor pumps to the polders where they reduce total economic losses most. The study focused on the 18-20 June 2021 flood event, using 20 tractor pumps and 48 selected polders in Hoogheemraadschap Hollands Noorderkwartier. Polder flood damages were quantified through Depth Damage Curves (DDCs), incorporating terrain and land use data from the WaterSchadeSchatter. A two-stage Mixed Integer Linear Program was then formulated to determine optimal placement of tractor pumps, where the First-Stage selected polders and the Second-Stage assinged pumps. The model systematically evaluated all possible allocation options, identifying which placements minimized total damage.

Key findings were that DDCs are not suited for assessing the impact of tractor pumps on polders in linear programming models, as the relation between volume and water levels in a polder are nonlinear. Instead, Volume Damage Curves (VDCs) are more appropriate, as they are able to quantify damage per cubic meter, the variable that pumps directly influence. VDC derivatives were used to classify polders into three types: Type 1 (always relevant), Type 2 (tipping point dependent), and Type 3 (low priority). Of the 48 polders included in the study, 25 were classified as Type 3. Of these, only 2 received pumps, and in both cases the prevented damage was minimal. In contrast, Type 1 and tipping point exceeding Type 2 polders accounted for nearly all significant damage reduction. This suggests that Type 2 polders below their tipping point, and Type 3 polders can be used for polder deselection and as an alternative for the First-Stage model.

A shortcoming was that with the current VDC use, the maximum water volume is the primary driver of damage, as the flood duration is assumed fixed. For agricultural and infrastructural areas, flood duration strongly influences economic losses, suggesting that both the VDC construction and use in the optimization model must be altered to incorporate duration as an influencing variable. VDCs that do so require the direct damage term of every polder to be corrected for the flood duration. This can be done for specific polder increments, where each increment is multiplied with a duration factor. After every model run the accumulated duration for each increment should be stored and passed to the next run, allowing the model to account for ongoing flooding.

Modeling duration in linear programming greatly increases the number of variables and constraints. To keep the enlarged formulation solvable, the pump placement variable should be aggregated by counting pumps only per type and time step instead of individual pump tracking. This change removes the distinction between the First- and Second-Stage, rendering polder subset selection by the First-Stage infeasible. Instead, the polder classification types can be used for subset selection before the solver starts, so the enlarged single stage model still finishes in time for operational use.

To support real-time decisions, HHNK should develop a short horizon model that integrates improved VDCs, forecasted rainfall, current water levels converted to polder volumes, and current pump placements. As new data becomes available, the model should update pump allocation accordingly. The current optimisation model can serve as a starting point for this operational tool.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
AHN	Actueel Hoogtebestand Nederland
CCB	Calamiteiten en Crisisbeheersing
DBR	Design Based Research
DDC	Depth Damage Curve
DTM	Digital Terrain Model
HHNK	Hoogheemraadschap Hollands Noorderkwartier
SDF	Stage Damage Function
SSM	Slachtoffer en Schademodule
VDC	Volume Damage Curve
WAT	Waterboard Action Team
WSS	WaterSchadeSchatter

1

Introduction

1.1. Introduction

In June 2021, the waterboard Hoogheemraadschap Hollands Noorderkwartier (HHNK) was struck by a large precipitation event, resulting in 100-140 mm of rainfall in a single weekend. The intensity and magnitude of the precipitation and subsequent flooding came as a surprise. In many places, water in the waterways rose above ground level and water remained between crops on the land. Consequently, the water board implemented its existing water calamity plan: mobilizing field teams, adjusting water inlets and weirs, maximizing drainage from the polders, and deploying additional tractor pumps. Local farmers and contractors also contributed by installing their tractor pumps for extra discharge capacity. In the evaluation of this event, the water board stated that flood control from a technical perspective was successful. Most of the pumping stations discharged water to the maximum extent. While efforts largely succeeded in limiting widespread damage, tractor pump placement lacked an overarching strategy and was done on an ad hoc basis. As a result, some tractor pumps were deployed in suboptimal locations [37].

As climate change increases the frequency of short, intense precipitation, conventional water infrastructures alone are insufficient to handle extreme situations. Water systems are not designed to handle such extremes. As such, some polder systems can quickly become overwhelmed even though pumping stations operate at maximum capacity. Tractor pumps stand out as a flexible, mobile resource that can be moved and deployed with relative speed to locations where they are needed the most. HHNK owns 20 tractor pumps with capacities ranging from 18 to 45 m3/min and can hire more from contractors if necessary.

The use of tractor pump currently relies on an ad-hoc “first-come-first-serve” approach, where operational field managers pick up a tractor pump at the depot if one is needed. This generally works in a situation when not all 20 pumps are needed, but might fail to result in optimal placement if more than 20 pumps are needed. In an event like June 2021, where more than 20 pumps were needed, that requires making choices on where to place the pumps. Currently, a coordinated procedure or framework for the deployment locations is missing, and placement heavily depends on professional instincts and experience of key individuals. HHNK acknowledges the lack of a coordination procedure [25], and is looking for a methodology for strategic allocation in extreme events like June 2021.

1.2. Research Objective

In situations like the June 2021 flood, the main purpose of deploying tractor pumps is to limit flood damage. Doing so requires quick, well-informed choices about where each pump has the greatest effect. Manually exploring placement options for dozens of polders and twenty pumps is not feasible within the narrow time window of a crisis. This study therefore aims to create a framework that, during an emergency, links flood simulation, damage estimates and pump placement in one integrated model and allocates a limited set of tractor pumps to the polders where they reduce total losses most.

1.3. Study Design

To achieve this aim, the study integrates the three steps of flood simulation, damage estimation and pump allocation within a single optimisation model, as shown in Figure 1.1. The proposed solution is to use precomputed Depth Damage Curves for individual polders, feeding the model with precipitation and initial conditions and letting the optimization algorithm choose between any combination of pump placements. The model is only constrained by the number and capacities of the pumps. This proposed approach reduces computation time, supports rapid scenario evaluation and offers a quantitative basis for crisis communication while the event is still unfolding.

Setting up this framework involves identifying the necessary model inputs, integrating appropriate modeling techniques and evaluating model performance. The June 2021 flood event serves as an experimental environment to build and test the approach. It also serves as a setting for drawing practical insights into polder selection. These insights are gathered through a design-based research (DBR) approach [31]. DBR builds on existing theory by generating practically relevant knowledge through iterative development [32]. This involves designing, testing, refining, and analyzing model results in real-world context [26].

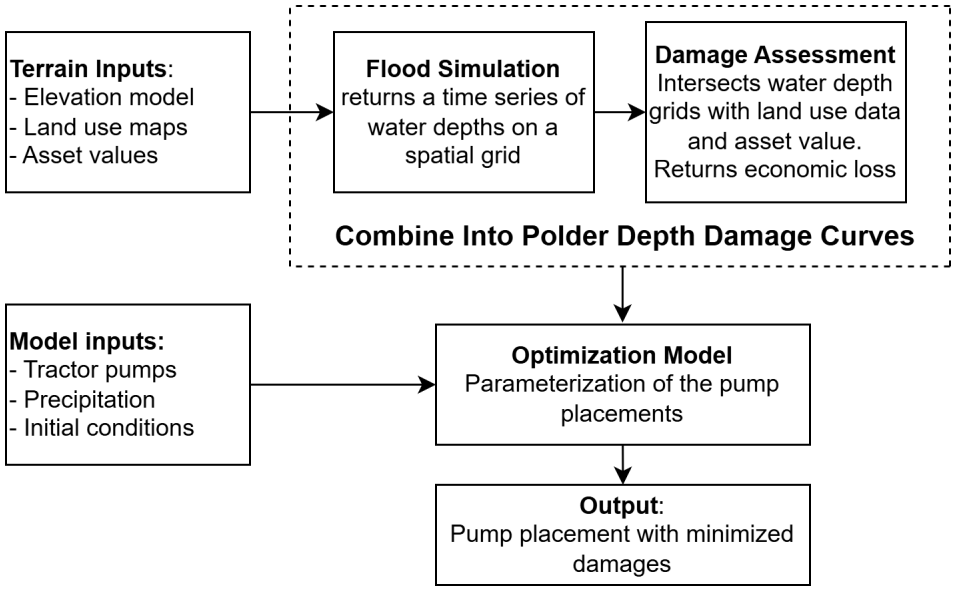


Figure 1.1: Conceptual framework. The dashed box shows the combination of flood maps and damage assessment in polder specific depth damage curves. Instead of providing manual pump placements, the model determines pump placements itself, aiming to minimise total flood damage.

1.4. Research Questions

This thesis aims to explore how a fixed number of tractor pumps can be allocated in a way that minimises flood damage. The primary research question guiding this study is:

- What framework is required to optimize the allocation of tractor pumps to candidate polders?

The main research question will be guided by three sub-questions:

1. What are the main insights from applying the prototype to the 2021 flood event, in terms of model performance and limitations?
2. What improvements are required in the damage assessment method and model structure to enhance scalability and usability?
3. What is needed to make the optimisation model usable as an operational decision-support tool during emergencies?

1.5. Structure of the Report

Chapter 2 describes how flooding happens in HHNK, details crisis response structure, and discusses potential measures the waterboard can use in case of a calamity. Chapter 3 describes the modeling framework, then discusses the operation of linear programming, how it constrains the modeling structure, and finally the coupling of flood modeling and damage assessment in this optimization context. Chapter 4 details the construction of depth damage curves along with the other data and equations that feed the linear model. Chapter 5 presents the optimization model results and discusses the allocation choices of the model, highlighting model functioning. Chapter 6 looks back at model performance, points out current limitations, and suggests ways to refine the approach. Finally, Chapter 7 provides the main conclusions and design principles for tractor pump allocation, as well as recommendations as a result of the study results.

2

Context: Flooding in HHNK

This chapter provides the context for flood response in HHNK, covering day-to-day water management, escalation to crisis operations, insights from the June 2021 event, the role of tractor pumps, and the 48 polders chosen for analysis.

2.1. Water Management in HHNK

Hoogheemraadschap Hollands Noorderkwartier (HHNK) is a waterboard located in the northwest of the Netherlands. It covers the province of North-Holland north of the North Sea Canal, including the island Texel, as shown in Figure 2.1. It is a governmental agency responsible for the regional water management. Its tasks include:

- Flood protection, involving the maintenance and reinforcement of dikes and dunes.
- Maintenance of water systems such as dredging and mowing to keep waterways operational.
- Wastewater treatment, ensuring that (household) effluent is purified.
- Managing and mitigating both excess and shortages of water.
- Emergency management which includes dealing with major pollution incidents, extreme water shortages or flooding and potential dike breaches.

Large parts of the area within HHNK's jurisdiction lie below sea level (referred to as +0 mNAP), and requires active water management to be maintain. HHNK's region is divided into 230 polders, and over the whole area of HHNK 363 polder pumping stations and 5422 weirs are located [36].

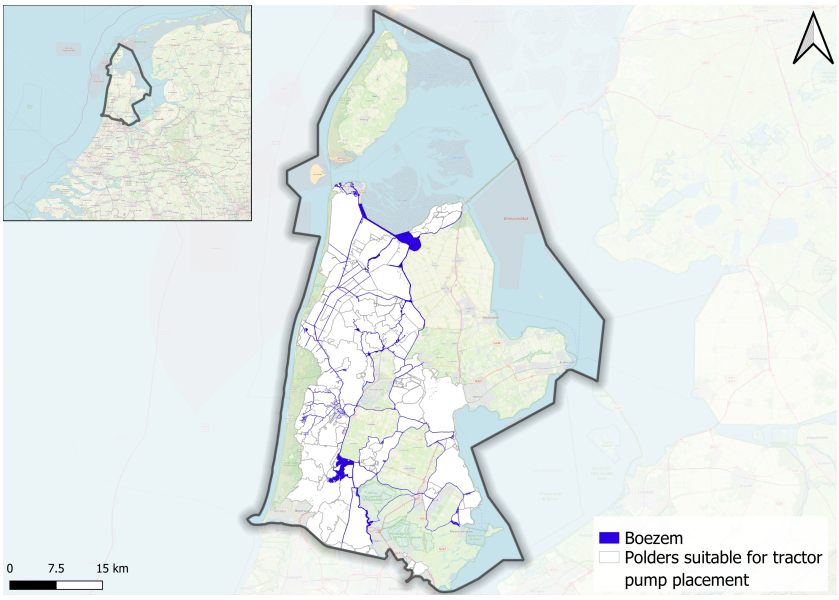


Figure 2.1: Boundary of HHNK, showing the boezem network and the 149 polders that are suitable for tractor pump placement.

Each polder typically functions as a separate unit, often encircled by dikes. Water levels inside a polder are regulated by weirs and pumping stations. Furthermore, polders are subdivided into subareas (known as Peilgebieden), each with its own target water level that can vary within a polder. These target levels are usually tailored to the adjacent land use, with different seasonal water levels (winter and summer). Winter water levels are lower so that water from the saturated soil can flow to the waterways and be discharged out of the polder. In summer, water is let into the polders and water levels are higher so that water infiltrates the soil. Pumping stations are used to pump water out of the polder into large water networks called boezem systems. The boezem system is a network of canals, water bodies and reservoirs that provides storage and transportation of water between polders. It spans about 529 kilometres and sits at a slightly elevated water level, so that water can be let in the polders using gravity and be discharged out of the system into the Markermeer, IJsselmeer, Waddenzee or North Sea Canal.

2.2. Routine Water Management vs. Crisis Response

HHNK has both a political and an administrative branch.

- The General Board is elected every four years and sets policy.
- The Executive Board, chaired by the Dijkgraaf, turns that policy into decisions for daily management.
- The Administrative Organisation carries out those decisions. It consists of nine departments, four for internal support and five for operational tasks: Water Systems, Water Chain, Water Safety, Projects Advice and Research, and Crisis Response [10].

Under normal conditions, the responsibilities of HHNK are carried out from the daily task execution of the departments. When an incident occurs the crisis management structure may be activated. The structure of this activation is detailed in the ”Crisisbeheersingsplan” [36]. The crisis plan distinguishes five alarm phases, numbered zero to four. Phase zero is the alert phase in which routine staff handle the incident while the crisis organisation remains alert. Beyond the Phase 0, there are four escalation phases, each activating multiple teams (see Table 2.1) [36]. Real crisis control starts with Alarm Phase 1, which is often initiated due to a phone call from an in-situ area manager. At this level, the WAT *(Waterbeheer Actie Team)* becomes operational. The WAT is the operational team focusing on source containment. It consists of a WAT leader, information coordinator, communications officer and crisis management advisor, but can also call in specialists from relevant fields such as dike maintenance (*waterkeringen*) or water level management (*peilbeheer*). At the second level the WOT (*Waterschap Operationeel Team*) becomes active, which is a tactical team focusing on impact control. The WOT is a separate team but has the same role holders. At the third level, the WBT (*Waterschap Beleids Team*) becomes active, which is chaired by the Dijkgraaf. In this team strategic-level decisions and public messaging are discussed. At the fourth and highest level, national agencies take the lead.

Table 2.1: Calamity alarm phases and active teams across the escalation phases. During the June 21 event, the alarm phase was escalated to phase 1, with only the WAT team active.

Description	Phase	WAT	WOT	WBT
Incident with heightened vigilance, routine staff	**Alarm phase 0**			
Localized response, source containment	**Alarm phase 1**	✓		
Coordination with municipalities; managing both source and effects	**Alarm phase 2**	✓	✓	
Regional crisis with significant societal and media impact. Broader coordination with external agencies	**Alarm phase 3**	✓	✓	✓
Interregional crisis with wide-ranging impact. Multi agency leadership by DCC-IenW	**Alarm phase 4**	✓	✓	✓

During a crisis, the crisis team (WAT team) should assume a guiding role, emphasizing pragmatism and risk acceptance rather than accuracy. Actions must be taken without full knowledge of the situation. The situation is chaotic, and there is often no time for hydrological calculations to estimate the situation. Ideally, intervention measures should be identified beforehand. In the June 2021 case, a guiding central coordination was missing [25].

2.3. Findings from the June 2021 Evaluation

During the June 2021 flood event, the crisis response was escalated to Alarm Phase 1 due to extreme precipitation exceeding designed discharge capacity, leading to widespread inundation and requiring an emergency response. In theory, upscaling in alarm phase is meant to streamline communication and manage physical challenges [41]. However, the only anticipatory criterion for upscaling remains weather forecasts, which are often unreliable due to large discrepancies between predicted and actual rainfall. In June 2021, rainfall fell in two stages: the first on Friday evening, with local quantities reaching up to 90 mm (with forecasts predicting about 20 mm), and the second throughout the night of Saturday and Sunday, with local amounts of around 50 mm. In total, up to 140 mm of precipitation fell during a single weekend. Such events far surpass design norms, exceeding what the waterboard can effectively control, and resulting in a reactive process of water crisis control.

After the June 2021 flood event, a large evaluation was done on the event. The main finding was that the event was handled effectively. However, centralized leadership was lacking, with the WAT taking a supporting rather than a steering role. The WAT lacked in-situ overview, forcing them to make decisions without an understanding of the situation [37]. Without an information overview, decisions were based on personal expertise, increasing the risk of misallocation of resources and delays in response. Following the evaluation, recommendations were made to improve information availability for impact forecasting, strengthening the robustness of water systems, and formulating strategies to reduce negative impacts during crises.

These findings from the evaluation are in line with a 2019 STOWA report, which reviewed 60 evaluations over the last decade from all waterboards within the Netherlands. The report stated that while waterboards are well-prepared to handle emergencies, they rely too much on the expertise of a few individuals [41]. This was also the case in HHNK during the event. At the time, HHNK lacked decision-support tools, making it dependent on expert judgment, which increases vulnerability if key personnel are unavailable. Following the June 2021 evaluation, significant improvements have been made on monitoring systems, but decision-making supporting tools or models are still absent. Developing structured decision-support systems would reduce reliance on individual expertise and ensure continuity in crisis management, regardless of personnel availability.

Extreme precipitation, like that experienced in June 2021, can develop suddenly, requiring an immediate response [41]. The watersystem infrastructure is not designed to handle extreme weather scenarios, as this would be costly. Consequently, it is impossible to completely avert occurring damage from flooding. Instead, interventions should focus on limiting the impact of flooding. Given the highly localized and intense nature of such rainfall, water boards must have predefined scenarios and a rapid response framework to deploy measures effectively. Because such rainfall bursts develop with little warning the water board must be ready to act quickly, using every measure available to limit impact.

Figure 2.2: A pumping station (gemaal), seen on the left, features a debris screen that can become clogged. On the right, a tilting weir (kantelstuw) is shown. When the water levels rise above the channel, the weir becomes subemerged [1].

2.4. What Happens and What Can The Waterboard Do?

In the waterboard, polder discharge capacities are dimensioned to discharge 10 m^3min^{-1} per 100 ha. This amounts to 14.4 mm/day, irrespective of polder size. This figure varies between polders, but this is the number that most polders at least are able to achieve. This figure is significant for polders, as there are little alternatives for water diversion there [41]. When 100 mm of precipitation occurs, it means that a polder with a removal capacity of 14.4 mm/day needs 7 days to discharge this volume completely. In those cases water accumulates in low-lying subareas (*peilvakken* in Dutch). Instead of draining away dynamically, the water gradually fills the polder until pumps or other drainage systems can lower the volume.

The waterboard is legally required to uphold the specified target levels in the polder (peilbesluit in Dutch). To achieve this, it must actively remove water from the polders using every available measure. This involves operating pumping stations at full capacity, closing inlets, clearing debris (preventing clogging, see left figure 2.2), manually or digitally adjusting weirs, diverting water into designated water storage areas and deploying emergency (tractor) pumps. The water board may also hire additional tractor pumps from contractors.

Even with all measures active, damage and disturbances will occur. How much occurs depends on the amount of precipitation and the elevation differences in combination with the landuse in the polders. Low-lying fields flood more quickly. If these fields contain e.g. flower bulbs, the damage will be higher than if it was potatoes. Since deep polders offer few diversion options, the only backup might be to open designed water storage areas. In short, the crisis plan relies on rapid pumping and using every bit of available storage throughout the polder [36].

2.5. Tractor pumps

Boosting polder discharge capacity above the pumping stations can be done with tractor driven pumps. At present the strategy is simple: field staff collect a pump from the depot on a first come first serve basis and install it where they think it will help most. In total, HHNK possesses 20 tractor pumps and 28 diesel or electromotor pumps, stored at the depots in Anna Paulowna or Zwaagdijk. Because installing diesel and electromotor pumps typically requires a significant time, these are unsuitable for rapid emergency response. Tractor pumps can be deployed more quickly, although their capacity depends on both tractor power and head conditions: higher static plus dynamic head reduces discharge capacity. The available tractor pumps of HHNK are of four types:

- 9 Veneroni AT30-5 with a capacity of 18 m3/min;
- 4 BBA B300 with a capacity of 20 m3/min;
- 6 Veneroni AT400/5 with a capacity of 30 m3/min;
- 1 Veneroni AT500/5 with a capacity of 45 m3/min.

Figure 2.3: The BBA B300 centrifugal pump in operation [3]. The top pipe serves as the discharge side, while the suction side is located at the back. HHNK owns four pumps of this type. The pump is connected to the tractor via the transmission (aandrijfas in Dutch). The discharge capacity also depends on the tractor's power.

The Veneroni pumps are considered 'dumb' pumps, lacking automatic controls or telemetry, and they require full submersion to avoid pumping air (which can damage the pump). They are most suitable for low static heads but can operate with steep embankments [17]. The pump body is lowered directly into the water and the rotor in the pump is operated through a connection via a transmission shaft. BBA B300 pumps are not installed in the water but on land, and are more suitable for higher head applications.

According to the technical service, transportation, placement and installment of the pumps takes around 3-4 hours, but this of course depends on the location, the amount of people working shifts and the capacity to transport the pumps. Installment location of the pumps in the polders is based on the expertise of the people in the field, but is likely at one of the predetermined locations.

2.6. Study Area Selection

Figure 2.1 earlier in the chapter shows all 149 polders suitable for tractor pump placements. This selection of polders was made by HHNK based on several characteristics:

- Land use;
- Total area;
- Location relative to a boezem branch.

From these 149 polders, a selection of 48 polders was made, as shown in Figure 2.4. This selection was made in consultation with the water board, who visited farmers in every candidate polder and asked for cooperation for the practical tractor pump placement locations and strategies. The 48 polders shown in Figure 2.4 were the ones where the local farmers expressed the greatest willingness to cooperate. Eventually, the aim is to extend the model to the full set of 149 polders.

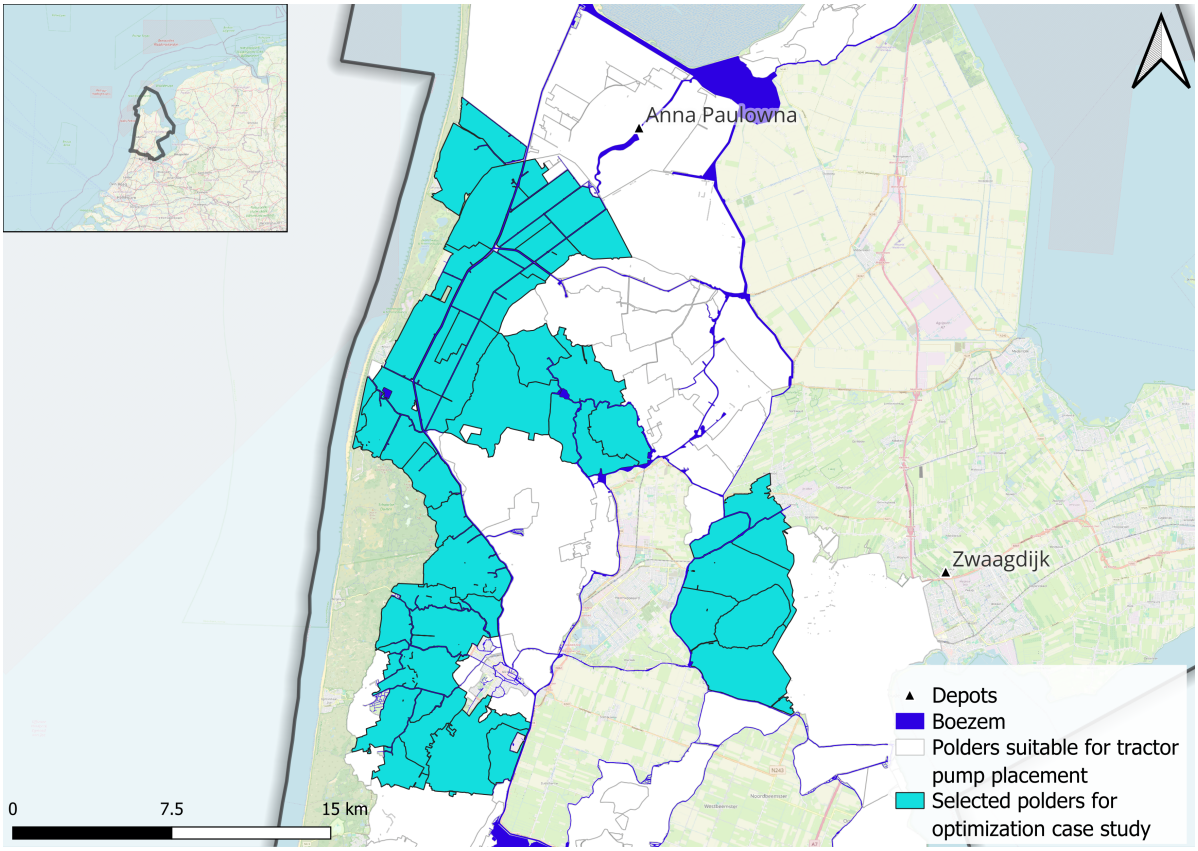


Figure 2.4: A close-up view of Figure 2.1, highlighting the 48 selected polders and the locations of the two HHNK depots where the tractor pumps are stored.

3

Theory Behind a Data-Driven Pump-Allocation Framework

This chapter discusses the framework and the theory for the integration of flood modeling and damage assessment into a single linear programming model.

3.1. Framework Overview

The framework is to combine flood modeling, damage assessment and pump allocation in an optimization model. This is done with linear programming, which can test every possible pump placement in a single run and find the global optimum. However, linear programming cannot call functions or models directly into its process, so the underlying modeling and calculation techniques of flood modeling and damage assessment must be directly incorporated in the linear programming model. The process of incorporation is visualized in Figure 3.1 on the next page.

The model is not to determine exact pump locations within a polder. Instead, it selects between polders to find those best suited for pump deployment. Internal flow dynamics are not included, each polder is essentially represented as a 'bucket'. Based on these simplified representations, the model prioritizes between polders to determine where pump deployment would be most effective.

Chapter 3: Theory

The constraints on the use of linear programming, which limit the options for flood modeling and damage assessment representation in the model structure are discussed in chapter 3. The study uses a representation of Stage Damage Functions as a means of damage assessment. In the Netherlands, the WaterSchadeSchatter is a program that uses these functions. The structure and damage calculation setup of the WaterSchadeSchatter are discussed to construct Depth Damage Curves and Volume Damage Curves for every polder, as these are suitable inputs for the linear programming model.

Chapter 4: Constructing Model Inputs and Formulating the Optimization Model

In this chapter, the focus is on constructing the model and model inputs required for the optimization model and the 2021 Case Study. These include the construction of volume damage curves, the initial volume, precipitation for the June 2021 event, and the polder removal capacities.
The chapter also details how the damage calculation is done in the linear programming model from the volumes and VDCs, and which constraints to pump placement are applied.

Chapter 5: June 2021 Case Study
Linear programming models are known to scale poorly, meaning that for added polders, timesteps or other complexity the solve time grows rapidly. As such, the model choices have been made such as the splitting into two stages. In the First-Stage, the model creates a subset of more promising polders for more detailed calculation in the Second-Stage.

This chapter details the model functioning of both the First- and Second-Stage. This includes plotting resulting variable values to interpret the functioning of the modeled functionalities, and stating the runtime and model sizes. It also includes interpreting pump placements, damage values and identifying influencing factors for polder selection in the First-Stage and the pump placement in the Second-Stage.

Chapter 6: Evaluation and Reflection on Model Functioning
Finally, the model functioning is evaluated. This chapter discusses suggested improvements on the model inputs, structure and a proposed alternative for the subset selection of the First-Stage. The step examines in depth the representation of the VDCs in the model structure, and how these in combination with the damage calculation can be improved upon to better represent the WaterSchadeSchatter damage functionality. The chapter ends with a suggested improved model structure.

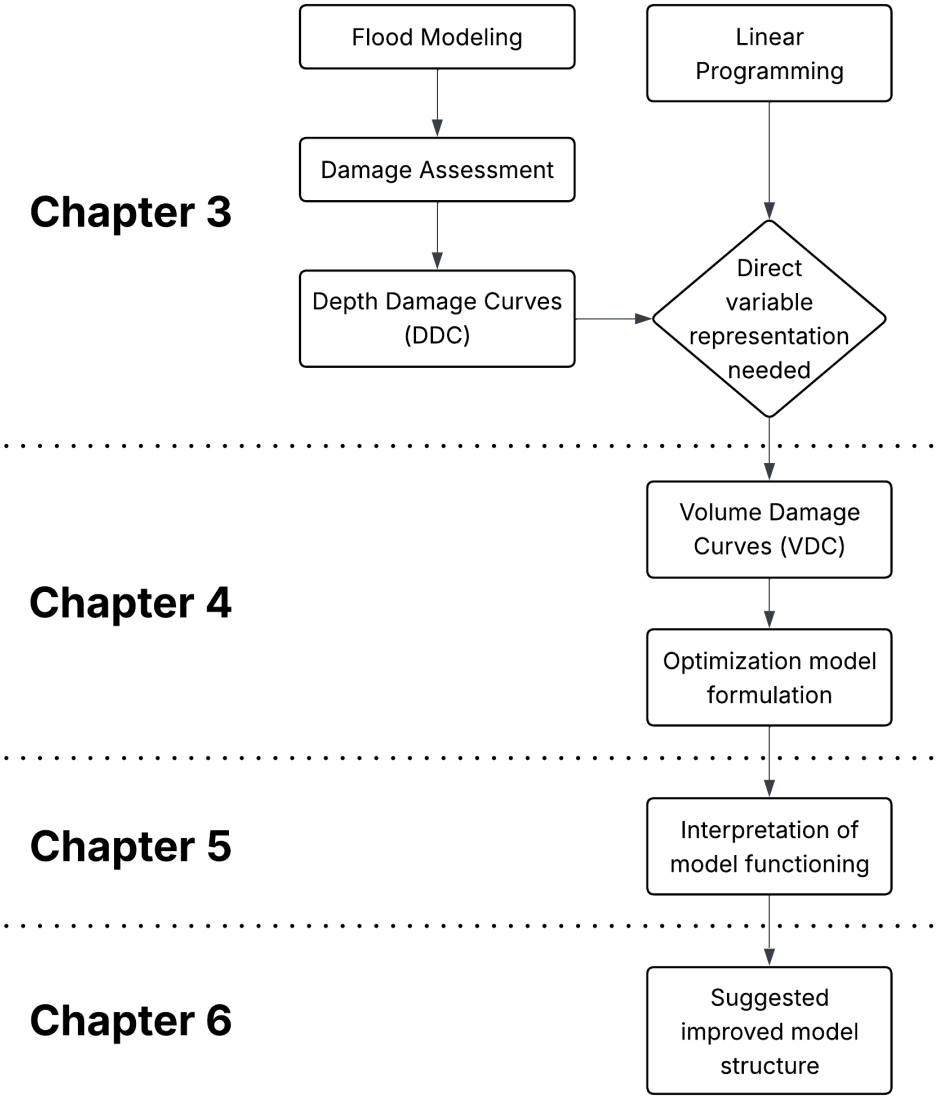


Figure 3.1: Visualization of the proposed pump-allocation framework.

3.2. Need for Fast Quantitative Decision Support

Recent trends in flood management show a shift from purely resistance-based methods to risk- and resilience-based management [38] [7]. Such approaches include non-structural measures that provide great flexibility [28]. In HHNK, tractor pumps are an example of such a flexible measure. During the June 2021 floods, HHNK deployed 18 pumps, and 32 were hired from private contractors. During the event, pumps were regularly moved to more suitable locations, showcasing their flexibility [37].

To demonstrate the effect of any pump placement a quantitative workflow is required that couples flood modelling with economic damage estimation [38]. By converting simulated flood depths into economic losses and comparing scenarios with and without pumps, managers can justify their response in economic terms. Models must therefore deliver clear recommendations within minutes, which is possible when pre analysed data or simple rule sets replace full model reruns [27]. A quantitative basis also supports centralised decisions, promotes shared understanding and mitigates stakeholder conflict [34].

Flood simulation models can predict estimated flood extent. However, flood simulation models are not flexible and modular, and are rarely used in flood disaster management. Studies by Leskens et al. (2014) and others [16][35] show that complex or overly detailed models are discarded in hectic disaster scenarios due to time constraints and information overload. This is due to a discrepancy in what modelers provide and decision-makers want. Decision-makers discard information that increases the complexity they already have to deal with.
Modelers assume that more detailed information improves the analysis and decision-making, whilst users lack the time and resources to perform such analyses. Under time pressure, decision-makers value clear, actionable insights rather than exhaustive technical details and comparisons of different model outcomes. Solving the pump allocation task therefore calls for a mathematical programme that can handle many possible pump placement choices under tight time limits.

3.3. Optimization Approach

3.3.1. From Simulation to Optimisation: Principles and Limits

Simulation models test one pump layout at a time. Meaning that they evaluate a predefined set of conditions and pump placements (e.g. the simulation of pump placement in a hydrodynamic model). Simulation models follow a sequential logic, meaning that decisions are made step-by-step based on the current or past states of the system and user inputs [4]. Simulation models can represent real-world processes better, but seldom find the best overall solution, because every new pump layout needs another model run.

Mathematical optimisation turns the question around. It evaluates all candidate pump placements simultaneously and searches for the combination that minimizes (or maximizes) the objective. By considering all input options at once, optimization models can find globally optimal solutions. The downside is that such models require all parameters to be explicitly defined and mathematically structured with a specific objective (e.g. minimize damage through pump placement). This requires formulating a problem into a declarative, more restricted way [4].

Optimization utilizes a declarative approach. This implies that a problem is formulated in terms of *what needs to be achieved* instead of *how to achieve it*. In this approach a problem is described through decision variables, constraints that define the limits of these variables at given indices (e.g. polder, pump or timestep indices), and an objective function. Declarative modeling is highly structured. It requires the problem to be fully specified before solving [16]. Solvers do not allow non-linear elements, such as variable multiplication, meaning that desired functionalities must be explicitly formulated. This means that the impact pump placement will have on the situation in a polder has to be known beforehand through a predefined relation, or implied directly in the model. Optimization models provide powerful decision-making capabilities, but they come with some limitations in the construction of the model:

- Explicit handling of variables is required; values are not assigned to variables until after the problem is successfully solved.
- Variables cannot be multiplied, as this causes non-linearity.
- Python modules or self constructed functions that calculate water levels cannot be called on optimization variables.
- Variable values cannot be negative.

- Condition logic, such as direct if-else statements, of variables is not possible. Conditional logic has to be directly implement through constraints on indices. The modeler must specify locations and durations for these constraints explicitly.

3.3.2. Type of Optimization Problem

The task is a scheduling problem: twenty tractor pumps must be assigned to forty-eight polders over several time steps so that total damage is minimized [9]. Scheduling models are often used in project management, where scarce resources are scheduled through predefined relations. Scheduling problems involves binary or discrete decision, as half a pump can not be allocated. Tractor pump are either placed or not. They also happen in defined steps (e.g. every few hours). The problem is characterized by:

- Binary or integer pump placement.
- Continuous polder water volume: There is precipitation entering the polder and volume being discharged through existing pumping stations and additional tractor pumps.
- Continuous damage: Damage is calculated in relation with the water level in a polder.
- The objective function: the value that the model seeks to minimize.

From an optimization perspective, this problem is formulated as a Mixed-Integer Linear Program (MILP or MIP) due to the combination of binary pump placement and continuous volume and damage. In Linear Programming, variables can either be free, where the models is allowed to determine the value itself within a set range, or constrained. Constraints in the context of this study include pump availability, pumping capacities and travel times.

3.3.3. Solving a MILP: Techniques and Considerations

An MILP is solved through Branch and bound methods. This combines branching (dividing the problem into smaller sub-problems), relaxation of variables (temporarily ignoring integer constraints by linearizing them) and pruning (discarding branches of which the relaxed theoretical solution is worse than the current best solution) to efficiently find exact solutions. A detailed description of the solver's functionality is included in Appendix D. The branch-and-bound approach systematically searches for the global optimum while eliminating infeasible or suboptimal solutions and branches early in the process. Optimization solvers automatically select the appropriate variation of the Branch and Bound and Cutting Plane techniques based on the model and variable structure.

Throughout the solving, a model is capable of calculation how close it is to optimality based on two values: the bound (best relaxed theoretical solution) and the incumbent (current best feasible solution). The percentage difference between these values is called the gap. When the gap reaches zero, the global optimum is obtained. A global optimum indicates that mathematically proven the best possible solution is reached [18]. Often in optimization, the model is stopped before reaching this optimum, as proving true optimality takes quite long. The most commonly used stopping criterion is a predefined time limit, or stating the desired allowable gap.

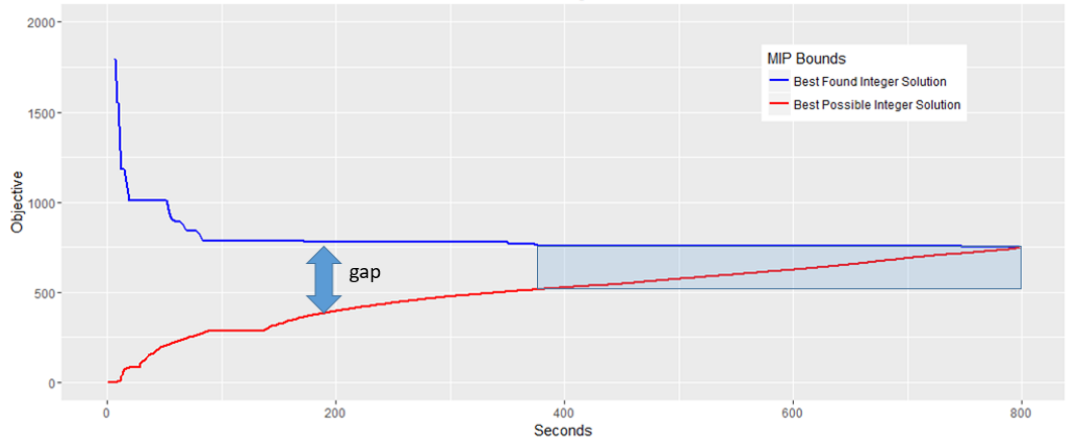


Figure 3.2: Visualization of a fictional Mixed Integer Problem (MIP) gap as an example [13]. The gap represents the percentage difference between the best-found solution (incumbent, shown in blue) and the bound (red). Optimality is proven when the gap reaches 0.

The convergence and runtime of an optimization model is influenced by various factors, including variable bounds, constraint selection, and model complexity. Tightening the bounds on variables reduces the feasible region, leading to faster convergence by bringing the linear relaxation of the integer variables closer to the optimal solution. Model complexity must also be carefully managed, as adding detail does not improve performance per se. The following considerations influence model efficiency:

- Feasible region size: Constraints and bounds, such as a max pumps per polder limit or maximum volume per polder, reduce the number of possible solutions. Removing constraints expands the feasible region, making the solver's job more complex, increasing solve time.
- Problem tightness versus constraint count: Adding constraints can lead to an overly tight model, potentially leading to suboptimal solutions. However, adding constraints can help the solver prune infeasible or non-optimal areas. A well chosen constraint guides the solver towards a better solution more quickly.
- Iterative model development: Optimization models are typically developed in iteratively, gradually adding complexity.

3.4. Flood Modeling and Damage Assessment in Optimization Context

3.4.1. Working Around Optimization Constraints

Flood-risk optimisation tools that couple flood hydraulics and economic damage are still scarce. Often, the scarcity is due to lack of high-quality, localized data [40]. In the Netherlands, luckily, there is enough data on high resolution available. When setting up an optimization model, it is important to have an idea of what kind of model to formulate. A 2024 study on application of optimization methods in water system operations stated that practical models should focus on making use of existing techniques instead of development of new algorithms [4].

As stated in the previous section, it is not possible to incorporate user-defined functions or simulation models directly into the optimization framework. Instead, the relation between the water level and damage has to be formulated beforehand [40]. The workaround is to calculate the water level damage relation beforehand, and to provide the model with these results. One of the techniques in linear programming to do so is through piecewise-linear (PWL) approximation. PWL approximation is a method to linearize 1D equations as a predefined relation.

3.4.2. Flood Modeling Choice for PWL Construction

To evaluate the effectiveness of tractor-pump placement, a flood model is required. Given the practical and computational constraints, simplified conceptual methods are most suitable for integration in optimization. There are several reasons for this. With the main reason being that hydrodynamic models require setup and manual adjustments for each change. Since there are 48 polders and the damage needs to be evaluated for dozens of water levels or flood scenarios, manually setting them up is not feasible. Besides, conceptual models, although they lack flow dynamics, are easy to setup and fast to calculate [29].

There are several low-complexity models, but not all are suitable in the context of optimizing pump placement. For instance, the HAND method provides rapid inundation mapping, but struggles in situations with abrupt terrain changes like levees and areas with minimal gradients and complex drainage systems. Topographic models such as the bathtub method rely solely on Digital Terrain Models (DTM) and are suitable for low-gradient terrains such as polders [2]. The bathtub (or sometimes called planar plane) method assumes that water spreads uniformly in the system, irrespective of flood dynamics. Another method is the inclined plane, which simulates a sloped surface that can account for variations in water level, it however is more computationally expensive and requires measurements to pick up on the slope in field [19].

The bathtub model is most suited for rapid, large-scale screening of scenarios where dynamics are not required. Inundation from heavy precipitation is often treated as static flooding [29], making the bathtub method the most suitable choice given the constraints of optimization. Application of the planer method is also in line with findings from the Evaluation of June 21, that stated that low lying polder regions were most stressed during the event [37].

3.4.3. Damage Assessment

Damage evaluation and economic assessment in the context of flooding is widely done with Stage Damage functions (SDF). SDFs are tools that express the damage as a fraction or percentage of the maximum potential loss with influencing parameters such as depth, duration, recovery time and season. A benefit for using SDFs is that the relation between water level (also called the stage) and damage is relatively straight forward. There are two types of Stage Damage Fucntions: empirical or synthetic. Empirical functions are built with field surveys of historical events based on actual damages, whilst synthetic functions use a what-if scenario, where hypothetical damages are assigned to groups or landuses [14]. In synthetic curves, an estimated monetary value is given to classifications of land use. The advantage of synthetic curves is that a high level of standardization is reached, and that it can be used across diverse regions. The disadvantages are that they require high development effort, robust data inputs and regular recalibration to account for changes in land use or damage costs per object [20].

Both methods of SDFs include direct and sometimes indirect tangible costs. Direct tangible costs are damages by direct contact with the water such as physical destruction of property, crops or infrastructure, whilst indirect tangible damages result from loss of profit, business interruption or costs of relocation [21]. Often, intangible costs are excluded from damage assessments due to their difficulty in quantification [28]. SDFs can be constructed through depth damage relations, or percent damage relations, where factors are multiplied with a maximum damage value of the land use type.

Most studies opt for the empirical method of development due to limited data availability. However, in the Netherlands, data availability is not an issue. In the Netherlands, two methods exist for damage assessment that both use the synthetic approach: the Slachtoffer Schade Module (SSM) and the WaterSchadeSchatter (WSS). The first was developed by the Ministry of Infrastructure and Watermanagement in 2000 and the second by STOWA in 2012. SSM is used for damage assessment of large scale flooding and WSS is used for damage assessment small scale flooding. WSS is better suited for calculating damage to agriculture and is generally used by waterboards as a means of damage assessments for flooding (inundation) scenario's [11].

How WaterSchadeSchatter works

WSS requires water levels (in m NAP) as input and is only a means of damage assessment and not of flood mapping [39]. WSS is a web-based model that, when provided with a raster of m NAP water levels, calculates the water depths at high resolution (0.25 m2) by subtracting the digital terrain model (DTM). Land use for each individual cell is then cross-referenced with the water depths for each cell in a damage functions, predefined in the form of parameters, and maximum damage values, calculating the damage in each cell for a single scenario.

Damage values for cells are calculated as follows: all factors, such as water depth up to a maximum of 30 cm (γ_{depth}), duration ($\gamma_{duration}$), season (γ_{season}) and recovery time ($\gamma_{recovery\ time}$) are converted into scaling factors. These factors are then multiplied with the direct or indirect damage values for each cell, as shown in equation 6.4 below. The factors compound, leading to large variations in damage outcomes between cells.

$$\text{Damage} = \text{Max Direct Damage} \cdot \gamma_{\text{depth}} \cdot \gamma_{\text{duration}} \cdot \gamma_{\text{season}} + \text{Indirect Damage per Day} \cdot \gamma_{\text{recovery time}} \quad (3.1)$$

For each land use, there are unique combinations for the scaling factors. For example, agriculture cells are season dependent whilst buildings are not. In total, there are 154 land use types, with unique combinations of maximum damage values and associated factors. Each land use type has a maximum damage value, given as a minimum, average or maximum estimate, for both direct and indirect damage. These values are then multiplied by a combination of scaling factors to calculate total damage. The factors are obtained via interpolation. Table 3.1 and Figure 3.3 on the next page show how damage values and SDF factors interact for two example land use types: Residential Function and Consumption Potatoes.

For the residential function, damage increases with greater inundation depth, whereas for potatoes it does not matter whether the depth is 5 or 30 cm, as they are grown underground. Potatoes are an agricultural product, meaning the loss is limited to the harvest value without any additional indirect damage. To remove the indirect costs, the $\gamma_{recovery\ time}$ factor is zero. For buildings (such as a residential function), the recovery time must be set manually for all cells and remains constant across them. In the case of residential land use, the duration of flooding has no influence. A one hour flood results in the same γ_{depth} factor as a three day flood. For agricultural products, however, the duration does matter. Twelve hours of flooding is not an issue, but three days can cause significant losses (see the top right plot). Another factor that varies across land use types is the seasonal sensitivity.

Table 3.1: WSS direct and indirect damage values for two land use types as an example. Note the difference in units. The factors for both types are shown in Figure 3.3.

Land Use Type	Damage Type	Unit	Min. Value	Avg. Value	Max. Value
Woonfunctie (Residential function)	Direct Damage	/m^2	163	271	380
	Indirect Damage	/m^2/day	5	11	16
Consumptieaardappelen (Consumption potatoes)	Direct Damage	/ha	2432	8415	2622
	Indirect Damage	/m^2/day	0	0	0

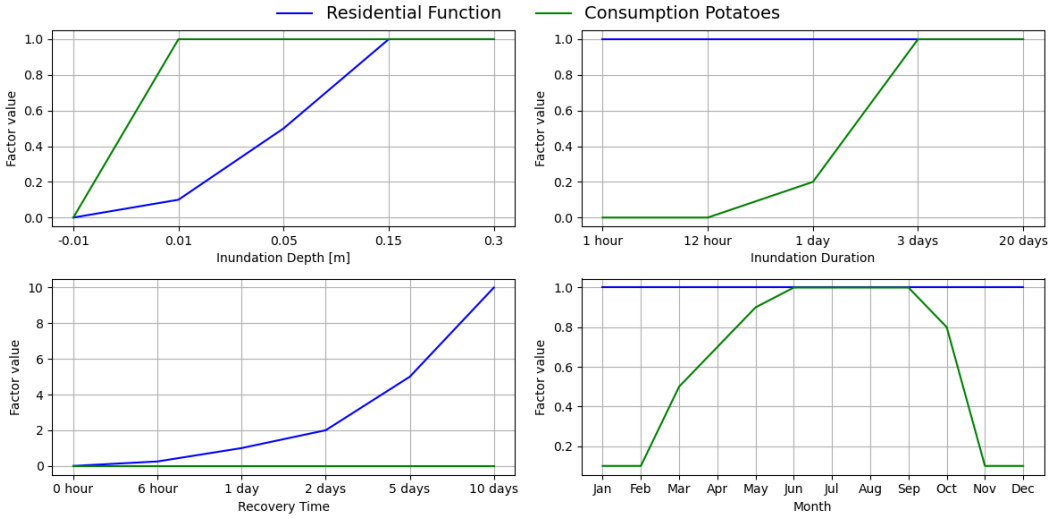


Figure 3.3: WSS factors for Consumption Potatoes (green) and Residential Function (blue). The topleft shows the factor γ_{depth} and the topright the factor $\gamma_{duration}$. The bottomleft and bottomright illustrate the factors γ_{season} and $\gamma_{recovery\ time}$, respectively. Duration and season is of importance for field crops and horticulture, but not for buildings.

Limitations on the use of WaterSchadeSchatter

There are some limitations in the use of WSS as a damage assesment tool. The first being that infrastructure damage is assessed using separate road maps, where indirect economic losses from detours are only calculated if an inundated section exceeds 100 m². A STOWA report found that most users override the indirect damage for this category, as they often think the values are too large [11]. The second is that damage does not increase beyond a water depth of 30 cm: a water depth of 1 meter results in the same SDF value as a water depth of 50 cm [39]. The third is that WSS uses synthetic curves that are deterministic, meaning that they do not account for uncertainty in the following three categories:

- Object data, e.g. spatial errors;
- The maximum damage value of objects;
- The SDF construction and it's associated parameter values [15].

However, since WSS is a tool maintained by STOWA, it benefits from regular updates, addressing concerns about inaccuracy of object data and maximum damage values [20]. That leaves the uncertainty in SDF construction and its associated parameters.

Using the WaterSchadeSchatter data to Construct Depth Damage Curves

By supplying WSS with a series of water level rasters, the program calculates the maximum water depth and duration of the flooding itself. This can be done by manually uploading the rasters to receive either a csv or a raster for the damages in the provided area. However, this requires dozens of input files per polder, which makes the process time-intensive when working with 48 polders.

As an alternative, WSS provides public access to its underlying datasets: land use maps, digital terrain models, and the parameter tables that contain the factor values. Working with those files makes it possible to reproduce the WSS calculation outside the web tool and to generate the same damage numbers much faster. Using this data, it becomes feasible to compute total flood damage for a range of uniform water levels in each polder. The result is a Depth Damage Curve (DDC) that expresses the relationship between water level and total expected damage for a single polder.

3.4.4. Depth Damage Curves

DDCs in polders often have a distinct S-shaped curves, as shown in figure 3.4, which displays the DDCs for polders Afdeling NS and Afdeling E. Most polders display single kinks in the curve, representing (critical) points where damage as a result of inundation accelerates. But some exhibit double kinks or other patterns, like polder Afd. E in figure 3.4, representing more complex relations due to land use features or varying terrain. A flat segment, where damage jumps almost immediately, usually means that one large plot of land or one costly land-use class is inundated as soon as the water level reaches that stage. These variations can provide insights into polder-specific profiles and inform targeted intervention strategies. Observing these curves therefore helps to locate tipping points and to decide where intervention can be most effective. The land-use composition is already directly embedded in the curve, so separate land-use plots add limited extra insight, and it is known that greenhouses and buildings dominate the monetary totals when flooded.

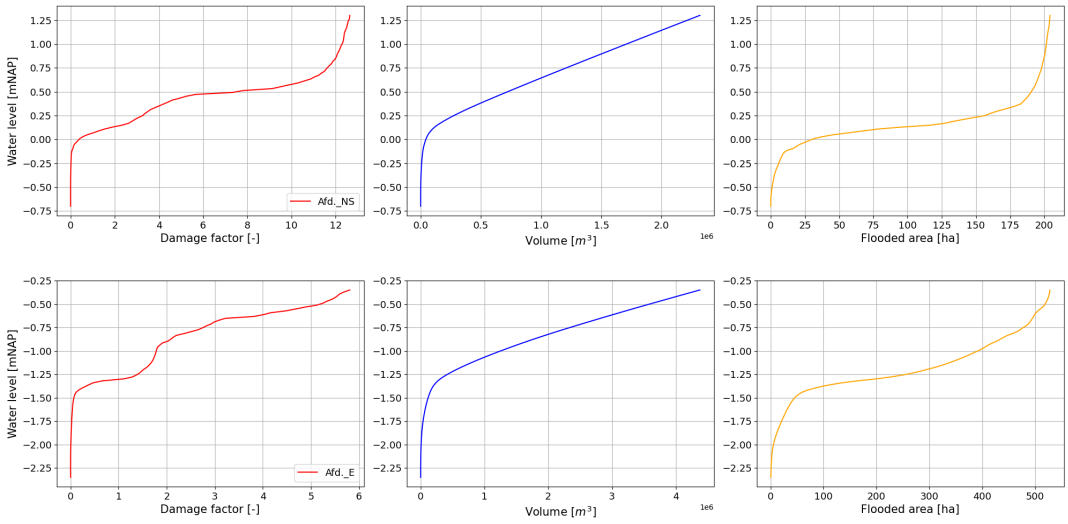


Figure 3.4: Top: Afdeling NS. Bottom: Afdeling E. In all plots, the y-axis represents the water level, ranging from the lowest target level in the polder up to +2 meters.

The relationship between water level and damage is intuitive and easy to understand, as water levels are quite concrete. This is also how decision-makers perceive the situation in the field, where insights stem from water level readings of loggers. Ideally, these readings are used to form (complementary) 2D visualizations of inundated areas. This can be insightful if you want to avoid inundation at specific areas, but still leaves two questions open:

- How will incoming (additional) precipitation translate into a rise of the curve?
- How much impact can an extra pump have on lowering the curve and as a result the damage?

One method to make these questions more insightfull is to add precipitation as a secondary y-axis. Figure 3.6 does so for the same polders Afdeling NS and Afdeling E. In the figures, y-axes are limited up to 200 mm (and the corresponding water level), as flooding beyond this precipitation magnitude is extremely unlikely. The plots highlight a non-linearity between the water level and volume in the polder. Although DDCs are clear for single polders, comparing the plots across polders is harder. Differences

in elevation mean that the same water level can represent different precipitation volumes, and the non-linear relation between water level and volumes hides the actual size of the polder in the curves.

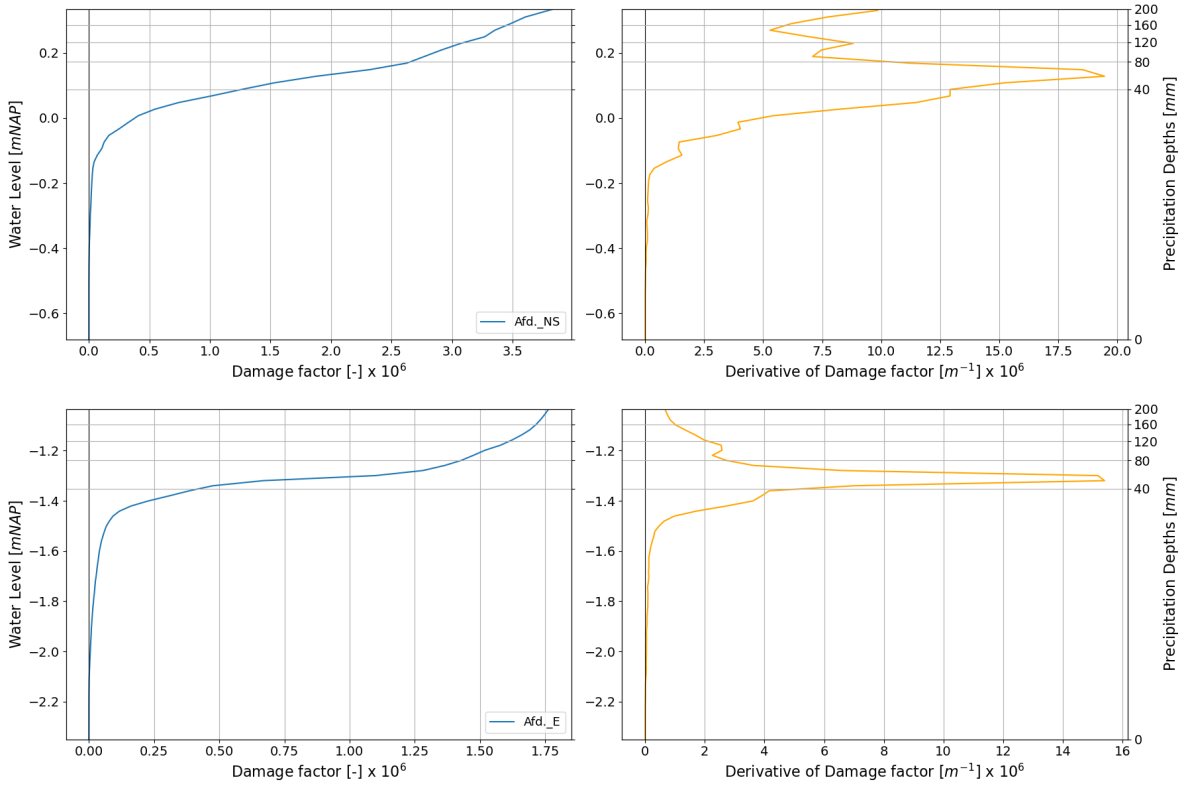


Figure 3.5: Top: Afdeling NS. Bottom: Afdeling E. DDC (left) and DDC derivative (right) zoomed in for a precipitation depth of up to 200 mm. A secondary y-axis has been added to show the corresponding precipitation (volume) relative to the water level.

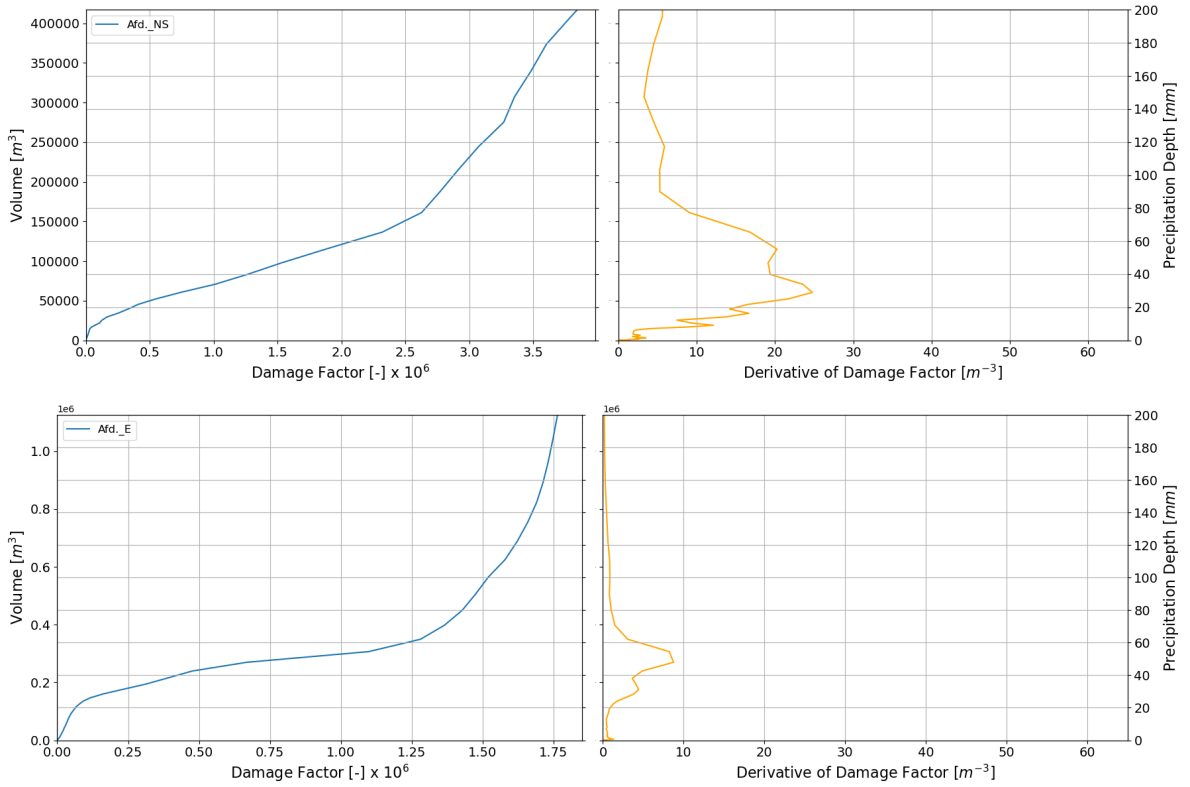


Figure 3.6: Top: Afdeling NS. Bottom: Afdeling E. VDC (left) and VDC derivative (right) zoomed in for a precipitation depth of up to 200 mm. The non-linearity has been removed, meaning that polders can now be compared directly.

3.4.5. The Need for Volume Damage Curves

The non-linear relation between water level and volume leads to three main issues. First, it makes visual comparison between polders and evaluation of pump placement effectiveness more difficult. DDCs do not account for the size of polders, leading to significant variations in the values along the x-axis that represent damage factors between polders. These differences are reflected in the varying logarithmic scaling on the y-axis, which shows volume. Second, non-linearity increases the complexity of the optimization model, removing the guarantee of finding a global optimum and requiring completely different (slower) solution methods. Third, because tractor pumps and precipitation influence volume rather than water level, using water level as the main variable requires an extra conversion step for each optimization node.

Using water levels as the main variable may seem intuitive at first sight, but adds much complexity to the optimization model. Instead, all three issues can be overcome when using Volume Damage Curves (VDCs) instead of DDCs. The relation between volume and damage results in a more interpretable, and easier to model alternative. This resolves the non-linearity and quantifies the increase in damage per cubic meter of water, offering a direct link to the variable that tractor pumps and precipitation influence.

4

Optimizing Tractor Pump Deployment

This chapter explains the preparation of precipitation and pumping data, the construction of the Volume Damage Curves from WSS land use and terrain maps, and the formulation of an optimization model that assigns tractor pumps to polders for the June 2021 Case Study to minimize flood losses.

4.1. Model Input Workflow and Data Preparation

The optimization model relies on several inputs to accurately evalute flood damage and determine optimal tractor pump placement for all timesteps. These include:

- Polder removal capacity.
- Hourly precipitation for the entire June 2021 event.
- The number of tractor pumps and their respective capacities
- Initial volume in a polder.
- The VDCs to relate water levels to the damage factor.

These five inputs directly feed into the final Optimization Model. The following subsections first address the preparation of the precipitation data and the calculation of polder removal capacities. After that, a separate section covers the construction of the VDCs and initial water volumes, as these involve more elaborate processing.

Polder Removal Capacity

Polders in the Netherlands are designed to handle a standard discharge rate known as the *'maat-gevende afvoer'* of 14.4 mm/day. In practice, this removal capacity is determined by the combined discharge capacity of all pumping stations. To calculate the polder removal capacities, the capacities of all pumping stations are summed per poldr. Minor adjustments were made according to a waterboard survey, as detailed in Appendix C. All pumping station's contribution are specified in this appendix, as well as the total polder discharge capacities.

Precipitation

The precipitation data for the June 2021 event was obtained via Meteobase, an online platform providing hydrological data for water management purposes. This data consists of radar-based precipitation measurements calibrated against readings from 216 ground stations [33]. The dataset spans three days: June 18–20, and is provided as hourly .asc raster files. Using an open-source multiband zonal statistics tool in QGIS [6], each raster band representing one hour was processed to calculate the mean precipitation intensity (in mm) for each polder. The 'vector layer containing zones' option of the tool was used to assign precipitation values to specific polders, and the results were exported as a CSV file for integration into the optimization model.

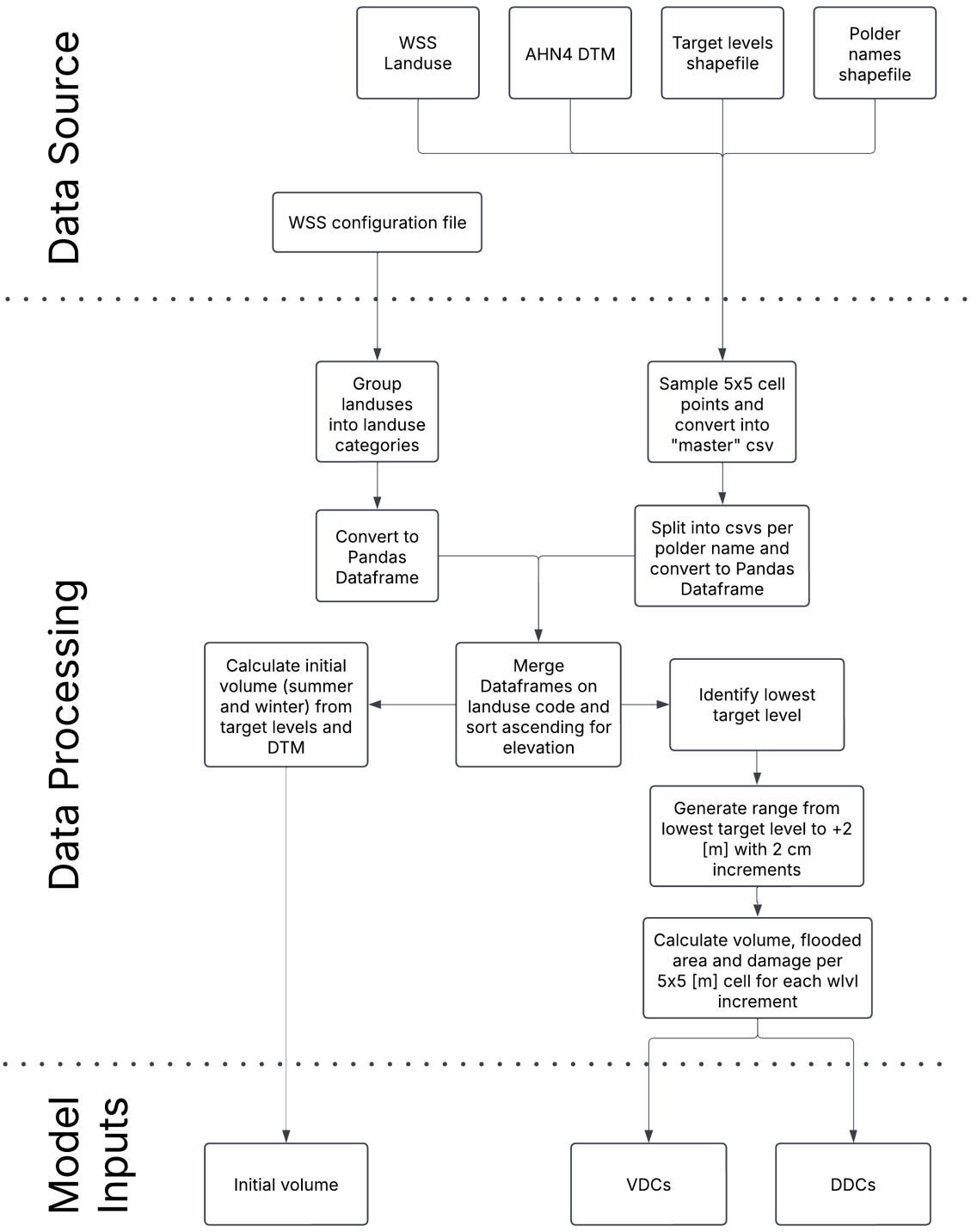


Figure 4.1: Flowchart showing the workflow from the WSS raster input to the calculcation of the intial polder volumes and the VDCs.

4.2. Construction of Damage curves

4.2.1. Data Collection

In total, five data sources are used for the construction of the VDCs and initial volume, as can be seen from Figure 4.1 on the previous page. The data sources used for the construction of the VDCs are the landuse and the AHN3 Digital Terrain Model (DTM) of 2019 from the WSS database [24]. These were the raster data closest available to the June 2021 event. The DTM and land use were downloaded by the waterboard as mosaic tiles from the WSS Lizard Catalogue via an API key [23]. The land use from this catalogue is specially created from a combination of various sources: the BAG register, TOP10NL, BRP gewaspercelen, OSM, and CBS bodemgebruik.

All land use specifics, like the maximum damage values and the factor specifications for the construction of de SDFs are specified in the configuration file [24]. This configuration file contains the factor and asset values for the in total 154 land use types. The prices from the configuration table are based on data from the period 2006–2015 and have not been updated since.

4.2.2. Data Preprocessing

The downloaded DTM mosaics are unedited by the waterboard, but DTM have been constructed from the original AHN3 rasters [39]. Trees, buildings and vehicles have all been filtered out of the AHN3 to remove false height values. These gaps were then filled as following:

- Greenhouses were filled with a floor level based on available data if available. Greenhouses that did not have (enough) available pixel points were filled with a floor level basen on the median pixel value from a 1-meter buffer around the greenhouse.
- Residental buildings were also filled via this buffering and filling technique, with the exception that 15 cm was added to the median value.
- Remaining blanks were filled with inverse distance weighted interpolation.

Minor adjustments were made on the WSS configuration file. The monetary values for three types of potatoes were adjusted as these were found to be inconsistent with previous WSS reports [12]. The adjusted prices are based on the KWIN-AGV number from a 2024 inventory of the Wageningen University [30].

The calculation of indirect damage to infrastructure cells relies on a specialized network map of road segments and intersections. Indirect damage is assigned for a whole segment when more than 100 m2 of a road segment is inundated. However, this raster for road segments and intersections is not downloadable through the Lizard API. Therefore, indirect damage for highways (code 25), regional roads (code 26), local roads (code 28) and railroads (code 31) classified with '/section/day' ('/wegvak/dag' in Dutch) have been excluded.

To format the data required for the generation of the VDCs, multiple steps were performed. First, the configuration file was loaded as a pandas dataframe, with all the different factor values uploaded as columns for each land use code. Then, land use types expressed in hectares (as can be seen in table 3.1) were converted to m2. Lastly, multiple processing steps were performed:

1. All mosaic raster files were merged into single raster files for landuse and the DTM using the python GDAL library. These rasters have a grid size of 0.5x0.5 meter.
2. The merged DTM is resampled to a 5x5 m grid using the average value.
3. A loop was used to process the resampled DTM rasters, extracting 5x5 m point values for the centre of the raster cells.
4. For each point, the corresponding land use, x and y coordinates were sampled. Note that while the DTM was resampled, the land use data remained at its original 0.5x0.5 m resolution.
5. Each point is also assigned a corresponding polder name and target water level (zomerpeil, winterpeil) by sampling shapefiles containing these values. More about this in the sector below.
6. The sampled and processed point data for all polders are saved as one large CSV file.
7. This 'master' CSV is split into smaller csvs for each unique polder. During this step, polder names have also been standardized by removing spaces, apostrophes and dashes to ensure code functionality. Unassigned points, which fell outside defined spatial boundaries, were exported to a separate CSV file for verification.

8. Each polder-specific CSV was then loaded into a Pandas dataframe, containing columns for land use codes, target water level, and DTM height.
9. These dataframes were merged with the configuration file dataframe to include all factors for each polder cell.

Target Level and Initial Water Volume

HHNK distinguishes five categories of water level control: Fixed, Flexible, Dynamic, Seasonal and Dynamic Seasonal. The polder dataframe stores, for every raster cell, both the DTM and the summer target water level. Only the summer target water level was used for the initial state of the model.

Since the control categories define water level regulation in different ways, a value was selected for each category to serve as the summer target level, as specified in Table 4.1. This approach allows each raster cell to be assigned a single target level. For each cell, the initial water depth was calculated as the difference between the summer target level and the terrain elevation from the DTM. Multiplying this depth by the cell area gives the initial water volume per cell. The total initial volume for a polder is then the sum of all cell volumes within a polder.

Table 4.1: HHNK has five types of target water level control. This table shows the corresponding values used for the initial water levels.

Target water level	**Fixed**	**Flexible**	**Dynamic**	**Seasonal**	**Dynamic Seasonal**
Summer level	Fixed	Upper bound	Upper bound	Summer level	Dynamic summer target level

4.2.3. VDC Construction

The VDCs are constructed using a synthetic approach, using data from the WSS database. The synthetic approach ensures that the individual cells reflect realistic economic values, and can be updated when new land use maps, monetary values or factors are available. Unlike the standard WSS workflow, where the maximum water level is uploaded for damage calculation, these curves were constructed directly to enable integration with the optimization model. The bathtub method with uniform water level across the polder was chosen for its simplicity and compatibility, and coded such that the theoretical VDCs simulate the damage ranging from the lowest target water level in the polder to 2 meters above it, with 2 cm increments. Practically, this means that the factor depth is the only varying factor in the individual SDFs of a cell. The other factors $\gamma_{duration}$, γ_{season} and $\gamma_{recovery\ time}$ were fixed on 3 days, June and 5 days respectively. This approach provides a practical solution that avoids manual uploads.

The volumes and damages were calculated incrementally for each polder, with 2 cm steps starting at the lowest target level. For each (uniform) water level, the inundation depth relative to the DTM is calculated. For each individual cell, the γ_{depth} is interpolated for the respective land use. Then, damage is calculated for each individual cell and then summed for all cells. Volumes and damages were aggregated for each water level and stored as outputs arrays to be used in the optimization model as inputs.

To reduce computational load, the raster cells were resampled from 0.25*0.25 m to 5*5 m resolution. At the original resolution, the interpolation of the depth factors proved to be too computationally intensive, regularly leading to crashes due to memory overload. By increasing the cell size, the number of calculations was reduced by a factor of 160. The coarser resolution significantly reduced the processing load, making the method feasible on personal hardware. Additionally, the interpolation of the γ_{depth} factor was done through a custom interpolation, and done through vectorization instead of looping through the cells for increased computation efficiency.

4.3. Iterative Cycles in Model Development

Now that the relation between volume and damage has been established as input for the Piecewise Linear (PWL) constraints in the optimization model, the model structure can be defined. The model is formulated in Gurobipy, which is a module that can be added to python. It is an interface that uses the Gurobi solver. The solver is accessed through an academic license. The gurobi solver uses algorithms to automatically detect what sort of problem it is dealing with, and selecting the appropriate techniques to solve the problem. As such, the user only has to specify the objective, the variables and constraints on the variables, but not the solving techniques [8].

As explained in the introduction, this study follows a design-based research approach, meaning that the optimization model was not constructed in a single step but refined iteratively through cycles of building, testing, and evaluation. This required adapting the model structure and inputs based on insights gained at each stage. Multiple modeling approaches were explored and discarded or improved upon depending on their performance.

Tractor Pump Placement Variable

In a simplified test scenario, tractor pump placement was one dimensional with a single timestep (order of $O(p)$), where pumps were placed for the entire duration of the event, without switching. This was then expanded upon so that pumps are allowed to 'jump' from one polder to another, by changing the variable tractor pump placement to $O(p*t)$. For this configuration, pumps still all have the same capacity. So the next step is to construction varying pumping capacities, which led to the binary variable order of $O(p*t*k)$, where k denotes the tractor pump. In this case, all tractor pumps are individually tracked, with the k index denoting the specific pump. To each individual pump, a capacity can be assigned.

Water Balance, Pumping Stations and Infeasibility

The damage is determined by piecewise linear (PWL) constraints that interpolate damage based on the maximum volume. Because the pumping station discharge is fixed, certain timesteps may lead to negative volume in polders, causing model infeasibility. To adress this, a Slack Volume variable $S_{t,p}$ was introduced. This variable is the second variable of which the model is allowed to determine the value. It is a continuous variable, which can take on any value between 0 and the polders pumping station capacity. Through the use of this variable, the model can add water to a polder when needed to prevent infeasibility. Theoretically, it can also add water when not needed, but adding water leads to higher volumes, and thus damage values in a polder. As the models goal is minimize the damage, it will automatically try to limit the added volume.

Pump Movement and Penalty Volume

In the simplified test scenario, tractor pumps were allowed to 'jump' instantaneously between polders without delay or cost. To address this unrealistic behavior, a method was required to incorporate travel time from depot-to-polder, or from polder-to-polder. To model this, a binary Pump Movement variable $Y_{t,k,p}$ was introduced, which identifies when pump k is assigned to polder p at time step t.

One approach to incorporating movement delay is through a downtime constraint, which explicitly prevents pump operation for a fixed number of timesteps after movement. However, this method introduces additional binary logic and can complicate the correct identification of pump movements. Instead, a simplified and computationally efficient alternative was introduced using a Penalty Volume variable $Q_{t,p}$. With this alternative, pumps are still allowed to move instantaneously, but when movement occurs (i.e., $Y_{t,k,p} = 1$), a penalty volume is added to the receiving polder. This penalty volume offsets the pump capacity for the arriving pump at that timestep, mimicking the delayed availability of the tractor pump with limited increasing model complexity.

Full Scale Testing With a Two Stage Model

Full scale testing with all 48 polders resulted in runtime issues, with the first iterations taking multiple days to reach a 10% GAP. To address these scalability concerns [4], a two-stage approach was adopted. A two stage approach is a manner of *preliminary screening* [18]. The idea is not to necessarly find the best solution, but to reduce a very large set of potential options to a more compact, promising subset.

The first stage is a simplified model: tractor pumps have generic capacities ($O(p*t)$), and no travel penalties are imposed, allowing pumps to instantaneously jump between polders. This stage identifies which polders benefit most from tractor pump placement for damage reduction potential and reduces the number of polders to be considered. The second stage is a more detailed optimization model with subset of polders and pump tracking ($O(p*t*k)$). This model includes real-world constraints such as travel time and individual tractor pump capacities.

4.4. Final Model Iteration

4.4.1. First-Stage

The goal of the first stage optimization is to generate a selection of polders to continue with in the second stage. It should serve as a quick indication that finds where the tractor pumps can have the largest effect as damage reduction without complex constaints. As such, pumps are allowed to instantaneously jump from polder to polder without forced travel time. All pumps are given the same capacity of 24.5 m3/min, which is the average of the 20 available tractor pumps. The optimization serves as an indication and due to its coarse nature the objective value of the best solution has limited usefullness, since this best solution is without the constraints that reflect real world application. The goal of the model is to minimize the total damage over all polders, which if formulated as the following objective function:

$$\text{Minimize } \sum_{p=1}^{P} D_p + \sum_{t=1}^{T} \sum_{p=1}^{P} c \cdot X_{t,p} \tag{4.1}$$

Where D_p is the damage in a polder as a result of the maximum water volume for all timesteps, obtained through the VDC. $X_{t,p}$ is the pump count, an integer variable stating how many pumps are placed in each polder and each timestep and c is an (arbitrary) parameter stating the cost of pump operation per timestep, which is modelled as 1000 euros per timestep. This was added so that the model doesn't place pumps when there is no more damage reduction to be obtained. While the first-stage optimization simplifies real-world constraints, such as instantaneous pump relocation and uniform tractor pump capacities, it is well suited to identify polders with high potential for damage reduction.

Table 4.2: Overview of Variables and Parameters. Decision variables are those the model is free to optimize, while other variables are fixed or constrained. {..., ...} indicates integer or binary variable, [..., ...] indicates continuous range.

Type	Name	Abbreviation	Domain/Value	Nature
Decision Variable	Pump Count	$X_{t,p}$	$\{0, 1, 2, 3, 4\}$	Integer
Decision Variable	Slack Volume	$S_{t,p}$	$[0, \text{PS}_p]$	Continuous
Variable	Maximum Volume	V_p^{max}	$[0, 200\ mm]$	Continuous
Variable	Water Volume	$V_{t,p}$	$[0, 200\ mm]$	Continuous
Variable	Damage	D_p	$[0, VDC(200\ mm)]$	Continuous
Parameter	Average Pump Capacity	X_{cap}	$24.25\,\text{m}^3/\text{min}$	-
Parameter	Operational Cost	c	0.001	-
Parameter	Precipitation	$P_{t,p}$	-	-
Parameter	Pumping Station Capacity	PS_p	-	-

The model, of which the code is provided in Appendix G, is subject to the following constraints:

- The total tractor pumping capacity ($TP_{t,p}$):
  It is an integer variable, determining the count of tractor pumps per polder, per timestep. X_{cap} represents the capacity of pump (24.5 m3/min). It is a free variable, so the model is allowed to determine the value.
  $$TP_{t,p} = X_{t,p} \cdot \text{X}_{\text{cap}} \tag{4.2}$$
- Maximum of 20 tractor pumps for each timestep:
  no more than 20 pumps can be deployed for all polders at any timestep.
  $$sum(X_{t,p}) \leq 20, \quad \forall\, t \tag{4.3}$$
- Maximum of four tractor pumps per polder:
  This constraint is set to prevent excessive deployment of tractor pumps in a single polder at a given timestep.
  $$X_{t,p} \leq 4, \quad \forall\, t, p \tag{4.4}$$

- Water Balance Equation:
  The volume at the initial timestep $V_{init,p}$ is set as the summer target water level.. The volume at subsequent timesteps is determined by the volume of the polder in the previous timestep, the precipitation, pumping station capacity and placed tractor pumps. Additionally, the Slack Volume $S_{t,p}$ is added to prevent infeasibility.

$$V_{t,p} = \begin{cases} V_{\text{init},p}, & t = 0 \\ V_{t-1,p} + P_{t,p} - PS_p - TP_{t,p} + S_{t,p}, & t > 0 \end{cases} \tag{4.5}$$

- Maximum Water Volume:
  This is an auxiliary variable that tracks the maximum water volume V_p^{max}. The maximum water volume is the value used for the interpolation of the damage through the PWL-constraint.

$$V_p^{\text{max}} \geq V_{t,p}, \quad \forall\, t, p \tag{4.6}$$

- Damage Calculation:
  As stated in the previous paragraph, Gurobi is unable to call functions during the optimization. Therefore the Damage (D_p) in each polder for a range of volumes is computed beforehand as a set of points. Between these points linear segments are enforced, so that a piecewise linear function (PWL) is formed. This is an integrated type of function for Gurobi. From this function, the damage can be interpolated for any volume within the range of the PWL function.

$$D_p = \text{PWL}(V_p^{\text{max}}, \text{DDC}_p) \tag{4.7}$$

4.4.2. Second-Stage

The first-stage optimization produces a 2D array representing the number of pumps placed for each timestep and polder. To identify polders where tractor pumps have the most impact, the total number of placements across all timesteps is calculated for each polder. Since each timestep in the first stage represents 3 hours, a polder qualifies for selection if the total pump placements exceed five timesteps, which corresponds to at least 15 hours of pumping. This threshold can be met in various ways, such as a single pump operating for multiple timesteps, multiple pumps operating simultaneously for fewer timesteps, or any combination where the cumulative pumping time reaches 15 hours.

This results in a list of polders where tractor pumps can have the largest impact. These polders serve as the input for the second-stage optimization, which incorporates more detail while focusing on fewer polders. To further improve computational efficiency, the timestep duration in the second stage is increased to 6 hours, reducing the number of timesteps to 12 (instead of 24).

The objective of the second stage remains largely same as the first stage: to minimize damage over all polders by placing 20 tractor pumps, but now of varying capacities.

$$\text{Minimize } Z = \sum_{p=1}^{P} D_p + \sum_{t=1}^{T} \sum_{k=1}^{K} \sum_{p=1}^{P} c \cdot X_{t,k,p} \tag{4.8}$$

In the second stage, several key changes are made to enhance realism:

1. Individual Pump Tracking: the decision variable Pump Count ($X_{t,p}$) is replaced with Pump Assignment ($X_{t,k,p}$), a binary variable where k represents the specific pump. This allows the model to handle pumps with varying capacities.
2. Pump Movement and Travel Time: pumps can no longer instantaneously jump between polders. A Pump Movement variable ($Y_{t,k,p}$) is introduced to track whether a pump moves to a specific polder during a timestep. Travel time is now added as a generic 3-hour (1 timestep) movement penalty using the variable Penalty Volume ($Q_{t,p}$).
3. Fictional initial timestep ($t = 0$): A fictional timestep with zero precipitation is added to account for variables which depend on the state at $t - 1$.

Unlike the first stage, where pumps could move instantaneously between polders, this stage incorporates travel time. To add a penalty, it must first be known when a pump arrived in a new polder. This is done through a binary Pump Movement ($Y_{t,k,p}$) variable. This variable tracks whether a pumps has moved into a polder at a given timestep. If $Y_{t,k,p} = 1$, a movement penalty is enforced.

Instead of modeling individual travel times between each polder, a generic 3-hour penalty (half a timestep) is applied whenever a pump moves. This is done through a Penalty Volume ($Q_{t,p}$), which adds a fictional water volume equivalent to 3 hours of pumping capacity of that specific pump to the destination polder. Since timesteps are 6 hours long, this effectively means that pumps operate at half capacity in the timestep they arrive. For example, if a pump moves to a new polder at t, it would only be able to pump for the equivalent of 3 hours instead of 6. This ensures that while pumps can move instantly, their effectiveness is reduced upon arrival, mimicking real-world travel delays without introducing complex movement constraints. If specific travel times were modeled for each polder, the number of variables and constraints would increase from $O(t * k * p)$ to $O(t * k * p^2)$. Additionally, the Penalty Volume variable ensures that complex intertemporal constraints such as minimum downtime are avoided. Aditionally, a fictional timestep $t = 0$ with zero precipitation is added. This is introduced since the movement variable $Y_{t,k,p}$ depends on the state at $t - 1$.

Table 4.3: Overview of Variables and Parameters in the Second Stage Optimization. Decision variables are those the model is free to optimize, while parameters are fixed inputs. Note that the Pump Count (O(t*p)) has been changed to Pump Assigned (O(t*k*p))

Type	Name	Abbreviation	Domain/Value	Nature
Decision Variable	Pump Assignment	$X_{t,k,p}$	$\{0, 1\}$	Binary
Decision Variable	Slack volume	$S_{t,p}$	$[0, \text{PS}_p]$	Continuous
Variable	Pump Movement	$Y_{t,k,p}$	$\{0, 1\}$	Binary
Variable	Water Volume	$V_{t,p}$	$[0, 200\ mm]$	Continuous
Variable	Penalty Volume	$Q_{t,p}$	$[0, 45 * 60 * t]$	Continuous
Variable	Maximum Water Volume	V_p^{max}	$[0, 200\ mm]$	Continuous
Variable	Damage	D_p	$[0, VDC(200\ mm)]$	Continuous
Parameter	Pumping Station Capacity	PS_p	-	-
Parameter	Tractor Pump Capacity	Cap_k	Varies per pump	-
Parameter	Operational Cost	c	0.001	-
Parameter	Precipitation	$P_{t,p}$	$-$	-
Parameter	Pumping Station Capacity	PS_p	-	-

The second stage model is subject to the following constraints:

- Pump Assignment:
  This constraint ensures that each pump can only be assigned to at most one polder per timestep.

$$\sum_{p=1}^{P} X_{t,k,p} \leq 1, \quad \forall\, t, k \tag{4.9}$$

- Maximum of 20 tractor pumps for each timestep:

$$sum(X_{t,k,p}) \leq 20, \quad \forall\, t \tag{4.10}$$

- Maximum of 3 pumps per polder:

$$\sum_{k=1}^{K} X_{t,k,p} \leq 3, \quad \forall\, t, p \tag{4.11}$$

- Pump Movement Tracking:
  The Pump Movement ($Y_{t,k,p}$) variable identifies when a pump is relocated. If a pump moves to a new polder at timestep t the variable is set to 1, allowing movement penalties to be enforced.

$$Y_{t,k,p} \geq X_{t,k,p} - X_{t-1,k,p}, \quad \forall\, t > 0, k, p \tag{4.12}$$

- Penalty Volume:
  this constraint calculates the cumulative effect of pumps being moved to the polder over the penalty duration of $0.5 * S$. It is calculated as the sum of the capacities of pumps. S represents the number of timesteps, and can be changed to reflect the duration of a timestep.

$$Q_{t,p} = \sum_{k=1}^{K} \sum_{d=0}^{S-1} 0.5 * \text{Cap}_k \cdot Y_{t-d,k,p}, \quad \forall\, t, p \tag{4.13}$$

- Movement Restriction:
  To discourage excess pump relocation, each pump is allowed to move at most once from the depot to a polder and once between polders. This constraint (significantly) reduces the feasible solution space, improving computational efficiency.

$$\sum_{t=1}^{T}\sum_{p=1}^{P} Y_{t,k,p} \leq 1, \quad \forall\, k \tag{4.14}$$

- Initial Pump Assignment:
  This constraint sets the initial condition at the fictional timestep $t = 0$, ensuring that no pumps are assigned at the fictional timestep.

$$\sum_{p=1}^{P} X_{0,k,p} = 0, \quad \forall\, k \tag{4.15}$$

- Water Balance Equation:
  The water balance remains the same, with the exception of that the total trator pumping capacity of formula 4.2 is now changed to reflect the binary nature of the Pump Assignment $X_{t,k,p}$ and varying capacities.

$$V_{t,p} = \begin{cases} V_{\text{init},p}, & t = 0 \\ V_{t-1,p} + P_{t,p} + S_{t,p} - \text{PS}_p - \sum_{k=1}^{K} X_{t,k,p} \cdot \text{Cap}_k + Q_{t,p}, & t > 0 \end{cases} \tag{4.16}$$

- Maximum Water Volume:

$$V_p^{\text{max}} \geq V_{t,p}, \quad \forall\, t, p \tag{4.17}$$

- Damage Calculation: The damage function remains identical to the first stage, using a Piecewise Linear (PWL) function to interpolate damage based on V_p^{max}.

$$D_p = \text{PWL}(V_p^{\text{max}}, \text{VDC}_p) \tag{4.18}$$

5

Case Study

This chapter applies the optimization framework to the June 2021 Case Study, reporting solver performance, pump allocation patterns, and the associated flood-damage reductions. It also sets out the baseline conditions that serve as the reference for all optimized scenarios.

5.1. Model Performance

First-Stage Model: Size and Runtime

The First-Stage model serves as a screening tool for the Second-Stage optimization. It assumes instantaneous pump movement between models, rendering the objective function less realistic. However, the model is not intended to capture detailed dynamics. Instead, its purpose is to identify polders that are more susceptible to flood damages. With the purpose of quick screening, the model was restricted to a maximum runtime of 1 hour. Figure 5.1 shows the solver's progress over this 1 hour period. In this figure, the orange line represents the bound (the optimal value of the relaxed problem) and the blue line represents the incumbent (the best solution found).

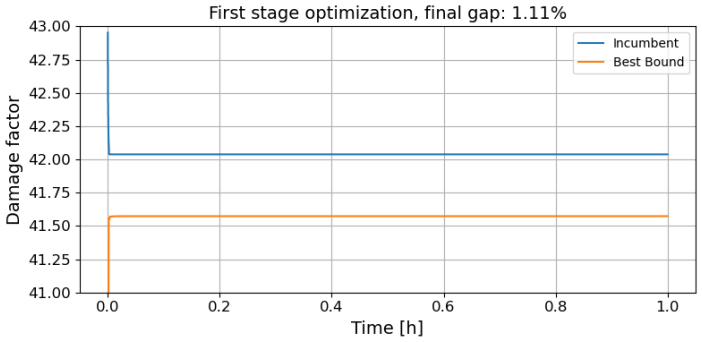


Figure 5.1: First Stage model logfile results visualized. The model quickly identifies the best solution but does not converge.

The model finds the best solution within 10 seconds. After this point, neither the incumbent nor the bound improved. The gap of this final solution was 1.1%. This behavior is as expected, given that there is no penalty for pump relocation. This assumption creates a large solution space with many combinations yielding similar objective values, which makes it difficult for the solver to tighten the bounds and converge. Details on the solvers performance are provided in Appendix J. The appendix shows the logfile, which includes the number of work units used. This metric is independent of the user's hardware, providing a measure of computational demand. For the First-Stage model, 5437 work units were consumed, reflecting the small computational effort needed to identify acceptable solutions within the given constraints.

Preliminary Screening for Second-Stage

To verify the consistency of the First-Stage outcomes, the five best solutions were stored and analyzed. Each solution includes the objective function value, and the complete set of decision variables. Tabel 5.1 shows the cumulative number of pump placements (total pump count over all time steps). All five solutions resulted in identical total pump count per polder. This indicates that while the timing of pump placement may differ, the solver consistently identifies the same polders as priority locations.

A threshold of five or more total pump placements was used to select polders for the Second-Stage model. This threshold was determined through trial and error. Six polders were not selected despite receiving pumps. These include: *Afd. AB* (9 hours), *Afd. I-Zuid* (3 hours), *Obdam* (12 hours), *Polder de Woudmeer* (9 hours), *Speketerspolder* (9 hours) and *Wimmenummerpolder* (6 hours). For each of these polders, pump placement was short and dispersed over the timesteps. This pattern can be observed in Figure 5.8, which shows irregular pump deployment for these six polders. It suggests that these polders were only occasionally beneficial for pump placement and did not provide sustained damage prevention over multiple time steps.

Table 5.1: First-Stage results, showing cumulative pump counts per polder for the five best runs. Each number totals the sum of all pump placements over all timesteps. Bold polders are selected for the Second-Stage. Identical totals per solution across the five runs confirm a stable selection.

Polder Name	Totals Per Solution	Polder Name	Totals Per Solution
Aagtdorperpolder	[6, 6, 6, 6, 6]	**Egmondermeer**	[14, 14, 14, 14, 14]
Afd. AB	[3, 3, 3, 3, 3]	Groeterpolder	[0, 0, 0, 0, 0]
Afd. C	[0, 0, 0, 0, 0]	Grootdammerpolder	[0, 0, 0, 0, 0]
Afd. D	[0, 0, 0, 0, 0]	Hargerpolder	[0, 0, 0, 0, 0]
Afd. E	[0, 0, 0, 0, 0]	Hensbroek	[0, 0, 0, 0, 0]
Afd. F	[0, 0, 0, 0, 0]	Lage Hoek	[0, 0, 0, 0, 0]
Afd. H-ON	[15, 15, 15, 15, 15]	Leipolder	[0, 0, 0, 0, 0]
Afd. I-noord	[8, 8, 8, 8, 8]	Obdam	[4, 4, 4, 4, 4]
Afd. I-zuid	[1, 1, 1, 1, 1]	Oosterzijpolder	[0, 0, 0, 0, 0]
Afd. KP	[0, 0, 0, 0, 0]	Philisteinsepolder	[0, 0, 0, 0, 0]
Afd. LQ	[0, 0, 0, 0, 0]	Polder de Berkmeer	[0, 0, 0, 0, 0]
Afd. NG	[10, 10, 10, 10, 10]	Polder de Woudmeer	[3, 3, 3, 3, 3]
Afd. NMR	[16, 16, 16, 16, 16]	**Polder Schagerwaard**	[26, 27, 27, 27, 27]
Afd. NS	[16, 16, 16, 16, 16]	Polder Valkkoog	[0, 0, 0, 0, 0]
Afd. OT-PV	[39, 39, 39, 39, 39]	**Ringpolder**	[26, 26, 26, 26, 26]
Afd. W	[14, 14, 14, 14, 14]	**Sammerspolder**	[26, 26, 26, 26, 26]
Afd. Z	[15, 15, 15, 15, 15]	Slootgaardpolder	[0, 0, 0, 0, 0]
Afd. ZG-ZM	[30, 30, 30, 30, 30]	Speketerspolder	[3, 3, 3, 3, 3]
Baafjespolder	[7, 7, 7, 7, 7]	**'t Hoekje**	[18, 18, 18, 18, 18]
Bergermeer	[9, 9, 9, 9, 9]	Ursem	[0, 0, 0, 0, 0]
Boekelermeer	[0, 0, 0, 0, 0]	Vennewaterspolder	[0, 0, 0, 0, 0]
Callantsoog	[37, 37, 37, 37, 37]	Verenigde Polders	[0, 0, 0, 0, 0]
Damlanderpolder	[9, 9, 9, 9, 9]	Wimmenummerpolder	[2, 2, 2, 2, 2]
De Kaag	[0, 0, 0, 0, 0]	Wogmeer	[0, 0, 0, 0, 0]

Second-Stage Model: Size and Runtime

The Second-Stage model introduces more realism, and great complexity compared to the First-Stage. It limits each polder's maximum number of tractor pumps to three instead of 4. It uses larger 6-hour time steps instead of 3-hour timesteps to prevent excessive switching and decrease computational demands. Additionally, each pump now has an individual capacity. Specifically, there are 9 pumps with a capacity of 18 m^3/min, 4 with 20 m^3/min, 6 with 30 m^3/min and one with 45 m^3/min. This requires tracking each pump individually. When a pump is assigned to a polder, a penalty volume equivalent to half a timesteps pumping capacity is added to polder. This simulates a generic 3-hour downtime due to travel constraints, forcing pumps to remain in one polder for longer periods. The model only focusses on the 19 selected polders from the First-Stage, which are shown in bold in Table 5.1. By narrowing the scope and number of timesteps, the computational demands are decreased, allowing incorporation of more realistic constraints.

Despite the reduced number of polder (19 out of the 48), and fewer timesteps (12 instead of 24), the Second-Stage model is larger. The explicit tracking of each pump's location introduces a large number of binary variables. This resulted in a computational effort amounted to 45,504 work units over a 12-hour runtime limit. The increased computational demand is largely driven by linearization of the binary variables and piecewise linear interpolation of the VDCs.

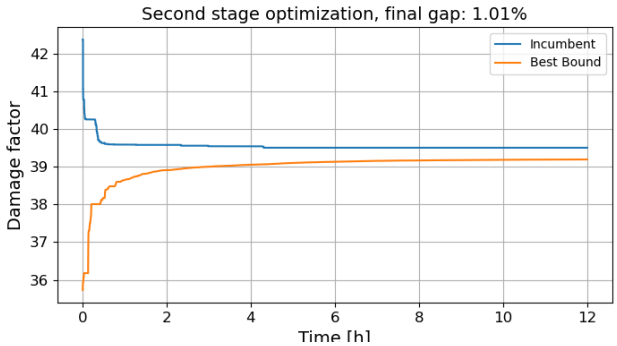


Figure 5.2: Second-Stage model logfile results visualized. The model continues to converge even after 12 hours. It takes around 10 minutes to find a solution within 5% of its best solution after 12 hours. Note that this is not the same as the gap but a comparison of incumbents.

Figure 5.2 shows the solver's progress. The model steadily improved throughout the 12-hour period, ultimately reaching an objective value of 39.51. Of this, 30.6 was attributed to the 19 selected polders and 8.91 to the 29 unselected polders. The final gap was 1.0%, and kept decreasing towards the end of the model run, indicating further convergence. Despite the addition of the penalty volumes, the Second-Stage model achieved a lower objective value than the First-Stage model. There are two causes for this result:

- By concentrating on the subset of 19 polders, this allowed the model to deploy resources more effectively, allowing pumps to stay in high-impact polders longer.
- The varied pump capacities, instead of a generic 24.5 m^3/min enables more nuanced decision-making. Larger pumps are deployed to polders with high damage potential, whilst smaller pumps manage other moderate-risk polders.

5.1.1. Slack Volume Functioning to Prevent Infeasibility

The Slack Volume is a free variable included in both the First-Stage and Second-Stage models to prevent negative water volumes. Negative water volumes would cause the model to become infeasible. The variable is bounded by the total pumping station capacity of the polder. The need for the Slack Volume is necessary as pumping station discharges are treated as constant. In timesteps where the fixed discharge would exceed the maximum available volume, negative values would result.

Conceptually, the variable mimics pumping stations turning off, or operation at a reduced capacity. The variable does not include a direct penalty in the objective function, but it does indirectly affect results. Additional volume leads to greater damage through the interpolation of the VDCs. As such, the model keeps the Slack Volume at a minimum, using it only when necessary.

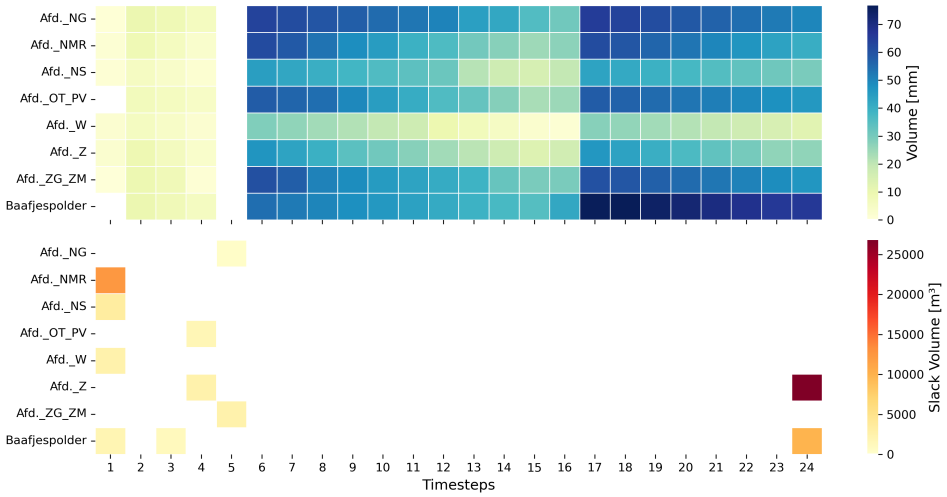


Figure 5.3: Visualization of Slack Volume functioning for 8 selected polders from the First-Stage model. The top heatmap shows water volume [mm]. The bottom heatmap shows the injected Slack Volume [m3].

Figure 5.3 shows how the variable works for a selection of eight polders from the First-Stage model. The upper heatmap shows the water volume in millimeters, which is the volume in the polder independent of the polder size. The figure reveals that all polders reached zero volume at timestep 5. To prevent infeasibility, the model injects volumes with the Slack Volume variable in the timesteps leading up to timestep 5. This shows that the variable functions as intended: it prevents infeasibility by injecting just enough volume when required. However, two notable behaviours can be observed:

1. The injection of volume does not occur at the moment it is needed, but can happen in any timestep prior to it.
2. In some polders, volume is injected in final timesteps of the simulation.

Both of these points are related to how the model calculates the damage in the polder. This is because flood damage is determined based on the maximum volume encountered. As such, the precise timing of the injection does not influence the objective value. Further explanation on the damage calculation from the maximum volume is provided in Section 5.3.2.

5.1.2. Pump Movement Tracking and Penalty Volume in the Second-Stage

As described in Section 3.3, explicit modeling of polder-to-polder travel time would result in too large a model for the evaluation of three days. Instead, the Second-Stage model tracks pump movement, and includes a generic penalty that simulates 3-hour travel time. This penalty is applied every time a pump is newly assigned to a polder, including deployment at the first timestep, as the initial location is the depot. The penalty is equal to the pumps capacity of 3 hours, corresponding to half a timestep. The penalty reflects the capacity of the specific pump being assigned, and is meant to discourage frequent relocation of pumps. The approach works as intended, which can be confirmed through the heatmaps in Figures 5.4 and 5.5. The first figure shows the arrival of the pumps, with their corresponding capacities. The second figure shows the penalty volume added in cubic metres.

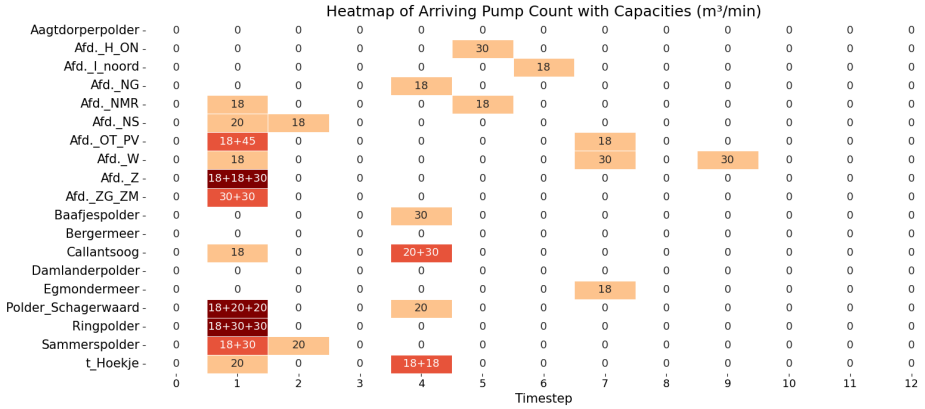


Figure 5.4: Visualization of the Pump Movement ($Y_{t,k,p}$) variable for all polders and timesteps in the Second Stage Optimization. Color coded by the number of pumps that arrived (1, 2 or 3).

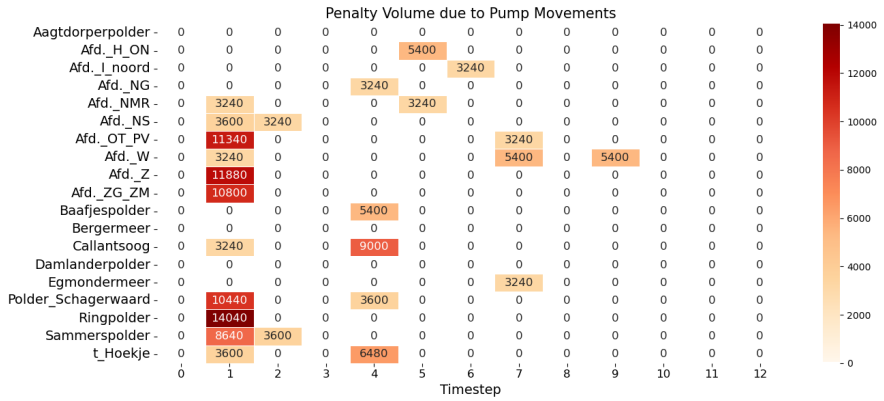


Figure 5.5: Visualization of the Penalty Volume ($Q_{t,p}$) variable for all polders and timesteps in the second stage Optimization. Comparing this figure with figure 5.4 shows that the penalty volume is based on half the cumulative capacity of the arrived pumps.

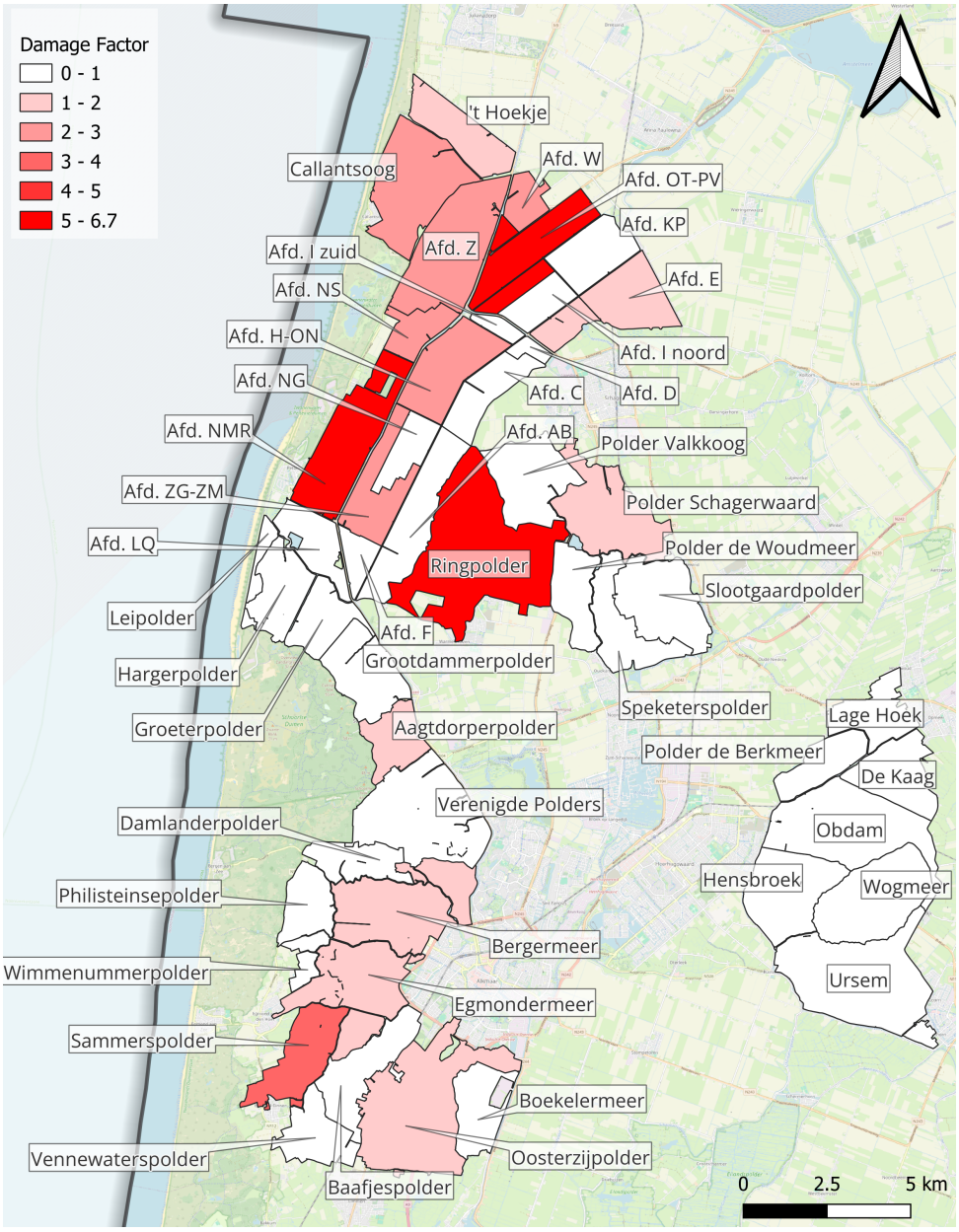


Figure 5.6: Visualization of the baseline damage factor for all polders. Most damage occurs in the regio Zijpe (all polders starting with Afd., as well as Callantsoog and 't Hoekje). Other outliers include the Ringpolder and Sammerspolder.

5.2. Baseline and Optimized Damage Factors

The baseline scenario represents the case where only the fixed pumping stations are active, with no deployment of tractor pumps. It serves as the reference scenario for evaluating the effectiveness of the optimization models. The damage factors are shown for all polders in Figure 5.6, and summarized in Table 5.2. Since these factors represent the total damage for an entire polder, larger polders generally display higher values. Figure 5.6 also includes the names of all 48 polders, enabling easier comparison with other figures.
The total damage factor of the Baseline scenario is 55.59 [-], with the largest contributors being:

- Afd. NMR (6.72)
- Ringpolder (6.23)
- Afd. OT-PV (5.42)
- Sammerspolder (3.68)

Notably, 24 polders had a damage factor below 0.5. The combined damage factors of just Ringpolder and Afd. NMR alone equaled that of these 24 polders combined. This highlights large variation in damage factors between polders.

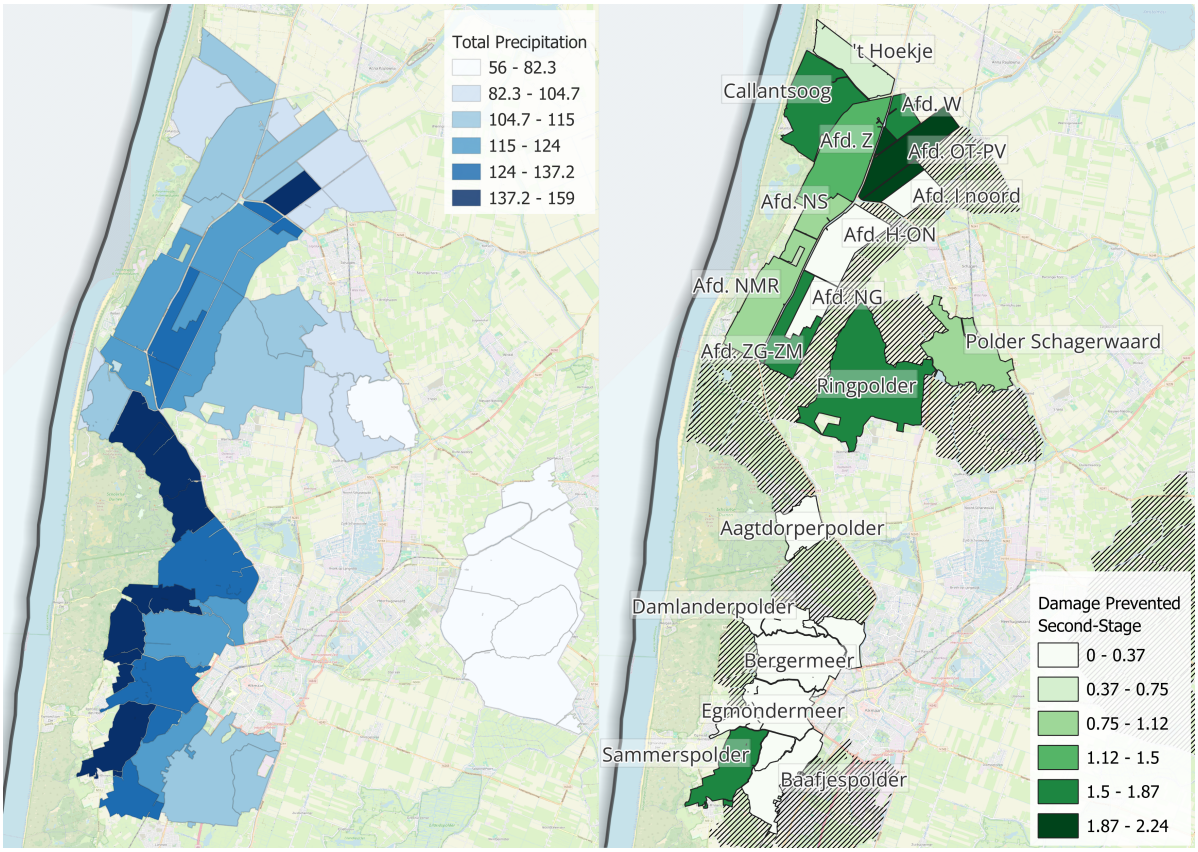


Figure 5.7: The left figure shows the total precipitation for the entire duration of the event. The right shows the deselected polders (shaded) and the prevented damage factors by the Second-Stage model.

The majority of the precipitation was concentrated closer to the coast. With the eastern polders Lage Hoek, Polder de Berkmeer, De Kaag, Obdam, Hensbroek, Wogmeer and Ursem received the lowest total volume of precipitation. Figure 5.7 illustrates this pattern, with the right-hand side showing the polders selected for the Second-Stage model. The eastern, low-precipitation polders were deselected. Southern polders close to the coast experienced the larges precipitation volumes. Notably, some of these polders such as Philisteinsepolder, Wimmenummerpolder, Damlanderpolder and Grootdammerpolder have low removal capacities (10.3, 10.7, 10.0 and 10.1 mm/d, respectively). However, despite having low removal capacities, these polders had damage factors of 0.12, 0.57, 0.19, and 0.05, indicating low vulnerability to flood damages.

Table 5.2: Polder characteristics, baseline damage factors and damage prevented in the First- and Second-Stage optimization models. Highlighted green values represent polder with higher prevented damage in the Second-Stage, red value denote polders which were selected, but where no pumps were allocated to. Gray rows correspond to polders that were not selected for the Second-Stage optimization.

Polder	Removal Capacity [mm/d]	Area [ha]	Total Precipitation [mm]	Initial Volume [mm]	Damage Baseline [-]	Prevented Damage First-Stage [-]	Prevented Damage Second-Stage [-]
Aagtdorperpolder	12.7	284	144	0	1.10	0.08	0.00
Afd._AB	17.2	543	122	1	0.74	0.01	0
Afd._C	14.0	316	124	3	0.43	0	0
Afd._D	48.9	56	134	1	0.01	0	0
Afd._E	13.3	563	103	2	1.39	0	0
Afd._F	19.8	138	128	3	0.19	0	0
Afd._H_ON	20.5	498	124	2	2.11	0.27	0.24
Afd._I_noord	19.2	202	151	4	0.56	0.26	0.19
Afd._I_zuid	16.7	69	137	1	0.23	0.02	0
Afd._KP	15.0	356	101	3	0.14	0	0
Afd._LQ	15.4	299	121	26	0.09	0	0
Afd._NG	18.1	215	123	1	0.97	0.23	0.19
Afd._NMR	25.0	692	123	1	6.72	0.57	0.91
Afd._NS	16.6	208	108	3	2.53	1.10	1.11
Afd._OT_PV	14.5	586	115	3	5.42	2.08	2.24
Afd._W	18.1	159	102	2	2.12	1.88	1.73
Afd._Z	27.1	791	107	5	2.60	0.60	1.16
Afd._ZG_ZM	16.6	381	125	5	2.51	1.64	1.72
Baafjespolder	17.2	461	121	0	0.66	0.13	0.25
Bergermeer	23.1	846	124	3	1.66	0.12	0.00
Boekelermeer	16.4	334	107	0	0.45	0	0
Callantsoog	17.5	739	99	4	2.09	1.75	1.65
Damlanderpolder	10.7	282	152	0	0.57	0.11	0.00
De_Kaag	14.1	409	76	0	0.09	0	0
Egmondermeer	16.1	714	130	1	1.98	0.22	0.06
Groeterpolder	11.5	301	138	0	0.04	0	0
Grootdammerpolder	10.3	461	152	0	0.12	0	0
Hargerpolder	15.3	361	114	0	0.04	0	0
Hensbroek	15.2	567	66	0	0.04	0	0
Lage_Hoek	20.9	327	78	0	0.03	0	0
Leipolder	14.7	94	104	0	0.00	0	0
Obdam	42.9	905	71	1	0.72	0.03	0
Oosterzijpolder	12.5	1127	106	0	1.64	0	0
Philisteinsepolder	10.1	285	159	0	0.05	0	0
Polder_de_Berkmeer	15.1	287	73	0	0.05	0	0
Polder_de_Woudmeer	17.6	327	88	3	0.21	0.04	0
Polder_Schagerwaard	16.7	659	91	0	1.55	0.94	0.98
Polder_Valkkoog	14.1	512	111	0	0.35	0	0
Ringpolder	14.4	1425	115	0	6.23	0.76	1.53
Sammerspolder	18.5	451	142	0	3.68	0.80	1.62
Slootgaardpolder	19.2	570	79	0	0.35	0	0
Speketerspolder	14.2	405	83	0	0.25	0.02	0
t_Hoekje	19.3	388	105	1	1.64	0.43	0.67
Ursem	16.1	1065	57	0	0.07	0	0
Vennewaterspolder	13.6	338	130	0	0.38	0	0
Verenigde_Polders	13.8	916	127	0	0.52	0	0
Wimmenummerpolder	10.0	115	157	1	0.19	0.02	0
Wogmeer	13.5	691	56	0	0.08	0	0

In Table 5.2, the shaded polders were deselected for the Second-Stage. The table shows the polders removal capacties, areas, total precipitation and initial volumes, as well as the damage factors for the baseline, First-Stage and Second-Stage models. It can be obseved that polders seem to be selected irrespective of size, total precipitation or other straightforward metrics. These variables alone do not explain the magnitude of the damage factor, as expected. Instead, the damage patterns are influenced by the interaction between local terrain elevation and land use types in each polder. Because these interactions are unique for each polder, a generalized regression analysis is unsuitable. Instead, each polder requries individual assessment necessary to understand damage accumulation and to analyse the rationale for selection and pump placement in the optimization models.

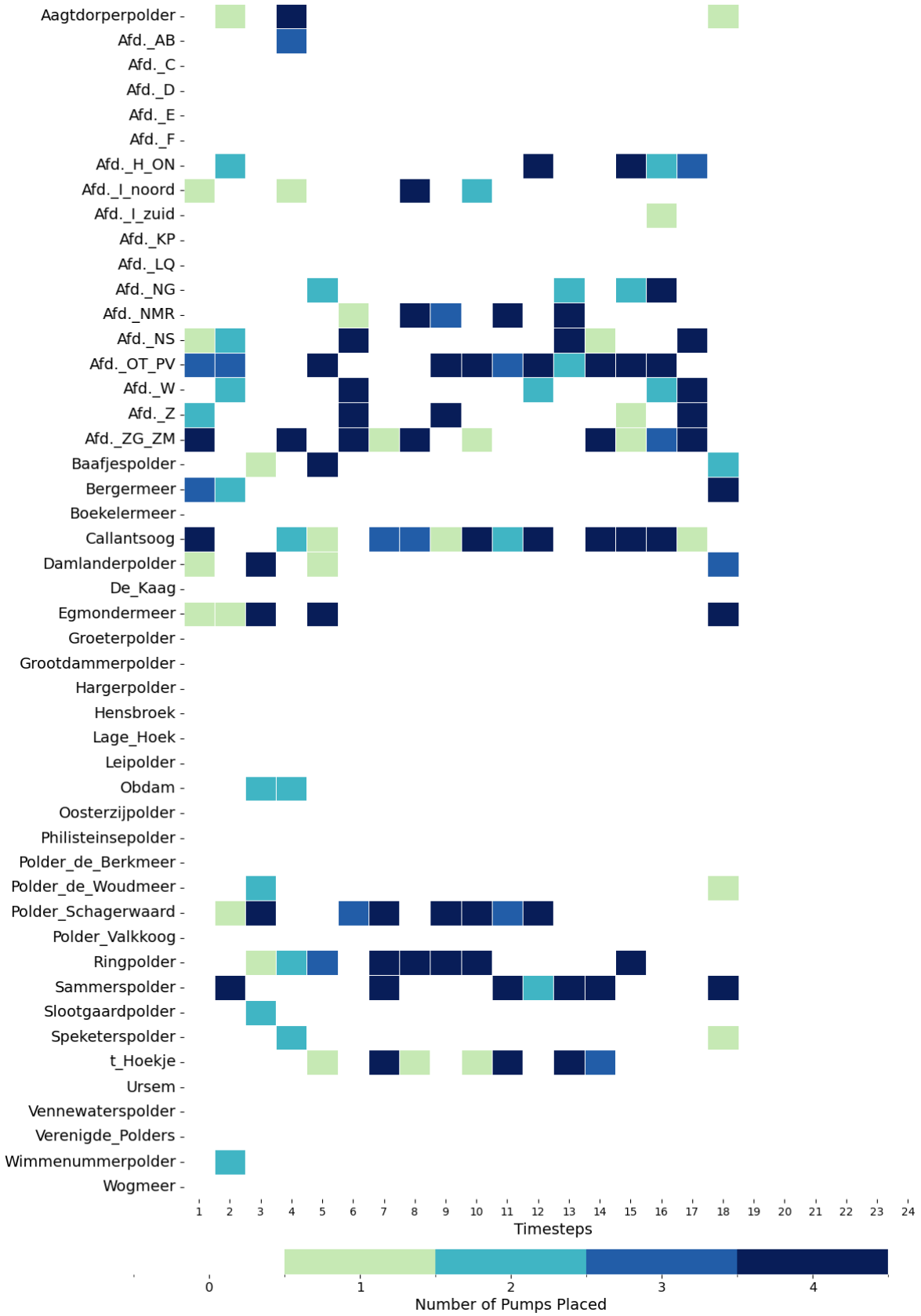


Figure 5.8: Pump count visualization for the five best solutions, showing pump placement for the five best solutions from table 5.1. The x-axis represents the timesteps (three hours per timestep). Instantaneous pump movement between polders is clearly visible. Also note that after timestep 18, no pumps are placed.

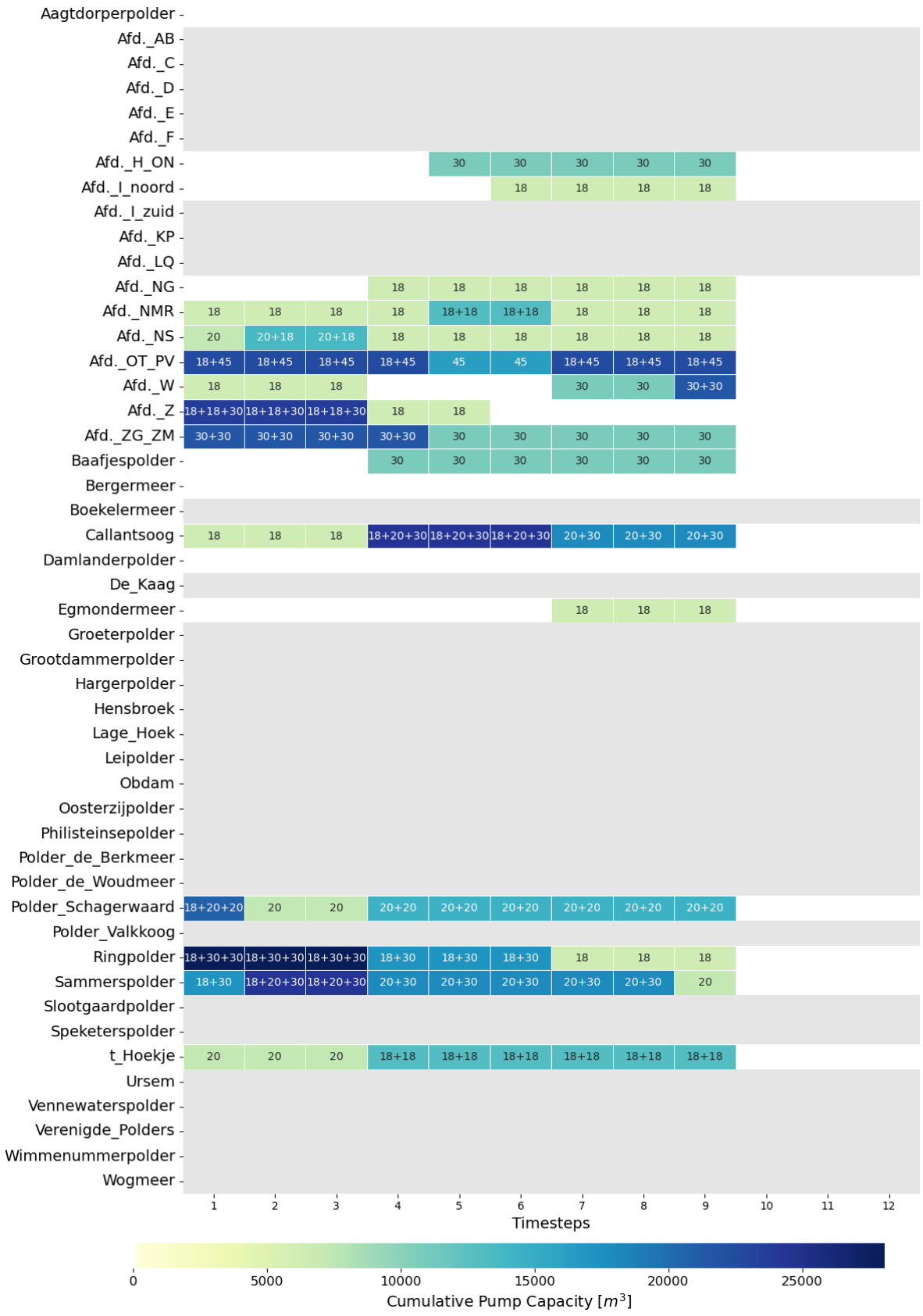


Figure 5.9: Pump placement visualization for the Second-Stage. The x-axis represents the timesteps (six hours per timestep). The gray rows represent polders that are not selected by the First-Stage Model. Also note that after timestep 9 (comparable with timestep 18 in the First-Stage), no pumps are placed.

5.3. Interpreting Pump Placements

5.3.1. Pump Placements in the First- and Second-Stage

Several observations can be made when comparing pump placements from the First- and Second-Stage models. Aagterdorperpolder, Bergermeer and Damlanderpolder received pumps in the First-Stage, but were not selected in the Second-Stage. These three polders are marked red in Table 5.2. In these polders, the First-Stage prevented relatively little damage: 0.08, 0.12 and 0.11, respectively. This is low compared to prevented damage factors in other polders.

In contrast, in polders such as Afd. NMR, Afd. Z, Ringpolder and Sammerspolders significant damage was prevented through pump placement. These polders are marked green in the same table. Notably, these polders are among the largest of the in total 48 polders, measuring 692, 791, 1425 and 451 hectares respectively. This demonstrates that larger polder can still be prioritized, despite the intuitive assumption that tractor pumps would be more effective in smaller polders.

Figure 5.9 shows that certain polders received a high number of pumps (e.g. Afd. ZG-ZM in which a damage factor of 1.72 was prevented), but Afd. W for instance received relatively few pumps but has almost the same damage prevention. This again highlights that tailored pump deployment can significantly impact placement effectiveness. It also suggests that at high-level, there is no clear or consistent relationship between simple metrics and damage prevention outcomes.

5.3.2. Damage Value Calculation and the Link to Maximum Volume

Although Figures 5.8 and 5.9 provide a visual overview of pump placements in the First- and Second-Stage models, interpreting these placements requires indivual attention to the polders. Allocation decisions are not immediately intuitive or directly interpretable. To better understand these placement decisions and identify patterns, this section focuses on how flood damage is computed in the model, and how this relates to the optimization structure.

Pump placement results from the interaction between the (free) decision variable Pump Placement and the objective function. As discussed earlier in Sections 5.1.1 and 5.1.2, the model primarily prevents damage by avoiding increases in the maximum water volume in a polder. This is expected, as the VDCs have one free influencing factor: γ_{depth}. This factor represents the maximum water volume over the entire simulation period.

The use of VDCs that depend solely on γ_{depth} as the influencing factor reveals a limitation: the optimization model focuses exclusively on minimizing the maximum water volume. As a result, no pump placements occur after the maximum volume peak. This behaviour is clearly visible when comparing the precipitation pattern in Figure 5.10 and pump placements of both stages in Figures 5.8 and 5.9. In the latter two figures, no pumps are allocated after timestep 18 in the First-Stage model or after timestep 9 in the Second-Stage model. These timesteps correspond to approximately 06-20 00:00 in the figure below. Notice the black dotted line, that shows the cumulative mean precipitation over all the polders. At this point, the final precipitation occurs, and in all polders, the maximum water volume in the baseline is reached. Consequently, the model allocated pumps so that the maximum water volume of the this point is lowered. However, after this point, the maximum volume has already been reached, so there is no incentive for further pump allocation. This is the direct result of how the objective function in relation to the VDC has been formulated, where the $\gamma_{duration}$ factor is fixed.

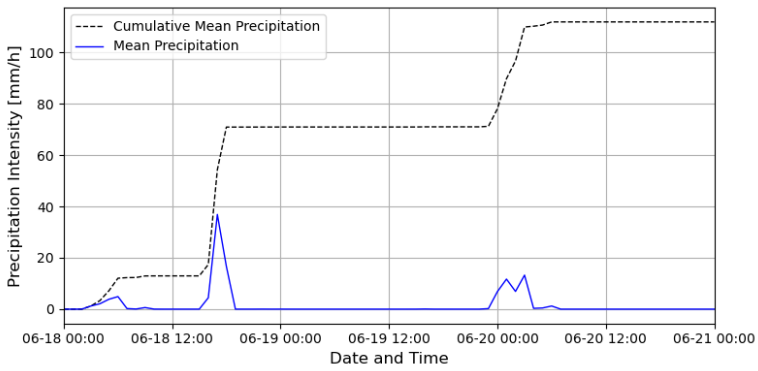


Figure 5.10: Average precipitation pattern of the 48 selected polders, showing both the pattern and the cumulative precipitation. All polders reached their maximum water volume in the early morning (Saturday to Sunday night).

The behaviour that no pumps are placed after reaching the maximum volume can shown by plotting the pump placements with the polder volumes of both the Baseline and Second-Stage model. This is done below for two example polders: Afd. Z and Afd. ZG-ZM in Figures 5.13 and 5.12. The orange lines show Baseline volumes, while the blue lines and shaded areas indicate optimized volumes and periods of pump operation, respectively. In both cases, pump placement was so that the second volume peak at timestep 9 did not exceed the first volume peak.

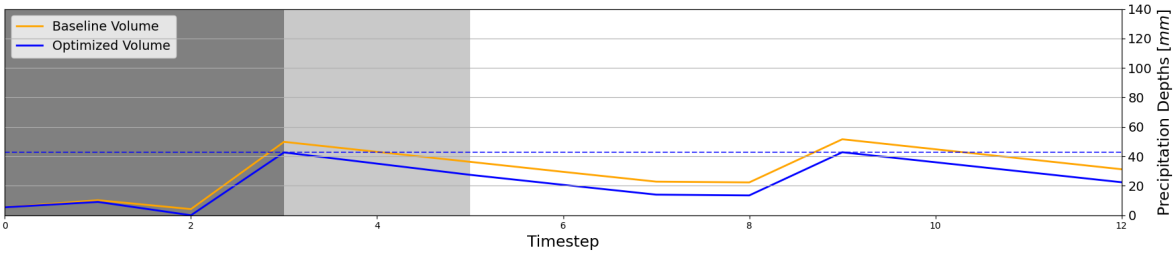


Figure 5.11: Water volume over time in polder Afd. Z. The orange line shows the baseline volume, while the blue line shows the optimized volume. The shaded area indicates the period of pump deployment. The dotted horizontal line represents the maximum water volume reached in the baseline scenario.

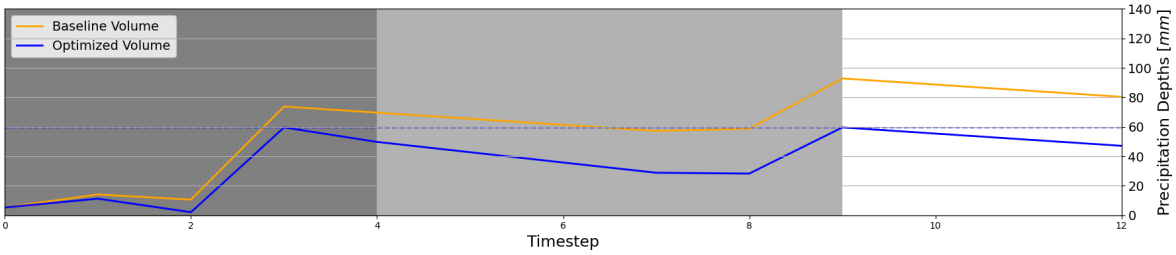


Figure 5.12: Similar plot but for polder Afd. ZG-ZM.

This highlights two issues:

- The curernt use of the VDCs and objective function formulation do not account for pump placement after volume maxima.
- As a result, the model wil not place pumps if no precipitation is given as an input, even if water volumes are high. This becomes problematic in short simulation horizons (e.g. when only two or three timesteps are evaluated). This neglects the effect pump placement can have on post-peak pumping, to reduce the flood duration.

The plotted results reveal a shortcoming of the model in using a fixed $\gamma_{duration}$: pump placement is strictly tied to the maximum volume in this evaluation of this study, but will also be tied to the forecasted precipitation in a short-term decision-support model when using the VDCs in the current format. The approach fails to capture operational value of post-peak interventions, ignoring the effect of flood duration and the potential for pumps to reduce damage even if no precipitation is expected.

5.3.3. Polder Damage Profiles

Despite the limitations of damage calculation discussed in the previous section, it is still possible to extract practical insights from the models inputs and ouputs. One such approach is to derive damage profiles, which are the derivatives of the VDCs. While the solver directly uses the VDCs during the optimization, visually comparing different VDCs is quite challenging. A better interpretable alternative is to plot the VDC slopes of polders and compare those. These damage profiles provide a better interpretable figure of how rapidly damage increases per unit of water volume, and thus how much benefit can be gained from additional pump placement.

In VDCs, the slope indicates the damage sensitivity. A 'flat' section represents increasing damage in the polder. In the damage profile, this is represent as a peak or large value. The derivative value can be used as a proxy for determining the effectiveness of pump placement. A high damage profile value indicates that reductions in volume can yield significant damage reduction. Figures 5.13 and 5.14 show these damage profiles and volumes for polder Afd. Z and the Baafjespolder. These figures now include the damage profile added as a secondary x-axis on top.

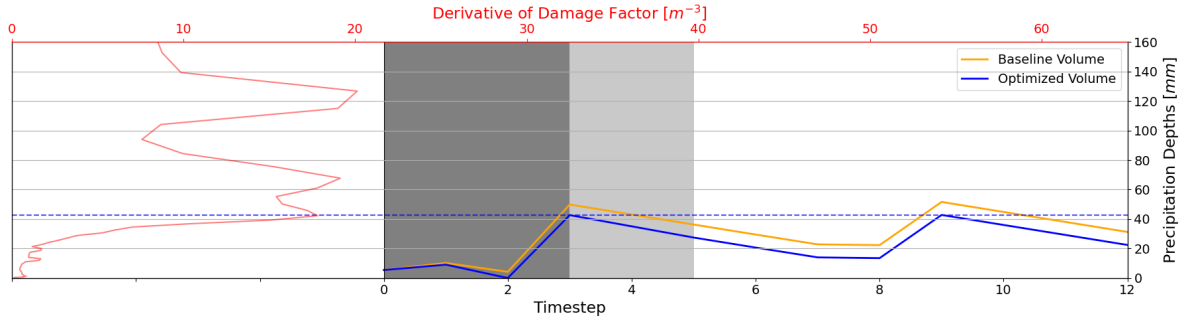


Figure 5.13: Water volumes and damage profile for Afd. Z, showing pump placement to prevent the occurrence of the damage peak.

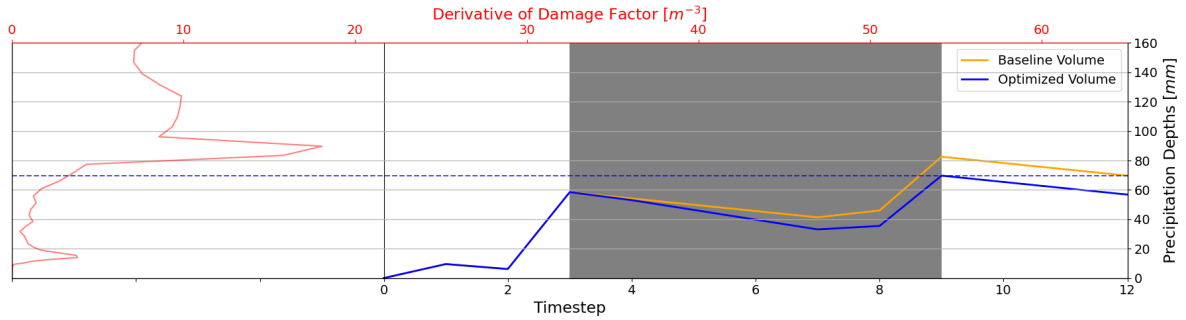


Figure 5.14: Baafjespolder volume and damage profile highlighting a sharp damage increase at around 80 mm.

In Afd. Z (Figure 5.13), three pumps were deployed in early timesteps to keep the second volume at timestep 9 from rising above the volume in timestep 3. This avoided entering a section with large slope values, where volume increases would lead to high damage increases. In Baafjespolder (Figure 5.14), the derivate values are more moderate across the entire volume range. The model can not prevent the volume at timestep 9 from exceeding the volume at timestep 3 and prioritizes other polders where pump deployment yields greater damage reduction per unit volume.
In contrast, the damage profile of the Aagtendorperpolder (Figure 5.15) demonstrates a stable slope, seen as a less fluctuating damage profile. This means that pump placement in this polder is suitable for the entire volume range. However, the damage profile is lower than other polders, making it a less attractive candidate for pump deployment.

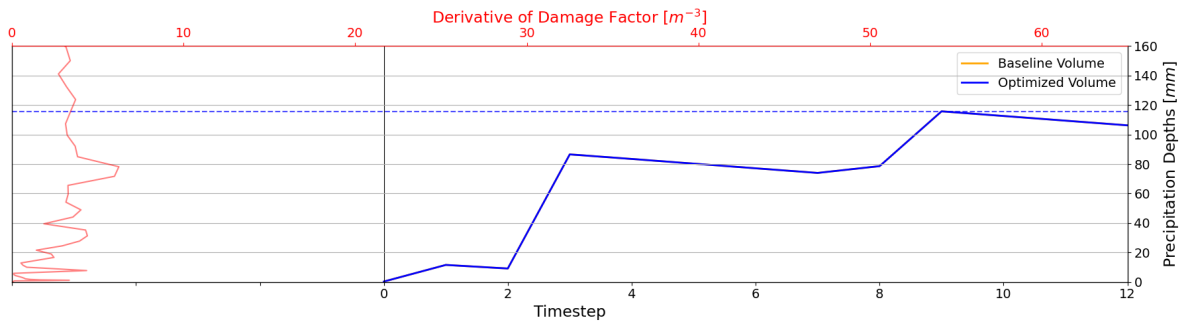


Figure 5.15: Gradual damage increase in Aagtendorperpolder, indicating a relatively steady damage increase per added volume.

5.3.4. Classification System and Damage Tipping Points

From the damage profiles of the polder discussed in the previous section, several patterns can be distinguished. The Aagtendorperpolder, for instance, shows a relatively stable damage profile, where the damage increase remains nearly constant across the entire volume range. In contrast, Afd. Z and Baafjespolder are characterized by minimal damage up to approximately 50 mm and 60 mm of water volume, respectively. However, when the volume increases above these point, damage increases rapidly.

Because the damage profiles closely align with pump placement decisions, it could be used as a basis for polder classification and identification of certain damage tipping points. These are points where a steep increase in the VDC derivative is visible. Below this point, benefit for damage prevention by pump placement is limited. Above it, each unit of pumped water can prevent substantial damage from occuring. The categories presented below are based on visual inspection of the VDCs rather than on a quantitative basis. They serve as a practical initial classification that can later be refined. Based on the visual inspection, three polder types can be distinguished:

- Type 1: Polders that should always be evaluated, as damage occurs at every volume stage.
- Type 2: Polders that should be evaluated only if the water volume exceeds a tipping point.
- Type 3: Polders that do not require evaluation, as the derivative values remain low, indicating minimal benefit from pump placement

Figures 5.16, 5.17 and 5.18 show examples for each type. All plots use the same axes limits for easy comparison. In some cases, Type 1 and Type 3 polders may appear similar. In such cases, the choice was made to classify assign them to Type 1. Type 1 polders should always be taken into consideration. The magnitude of the damage profile varies depending on the volume depth, but some level of damage occurs across the entire range.

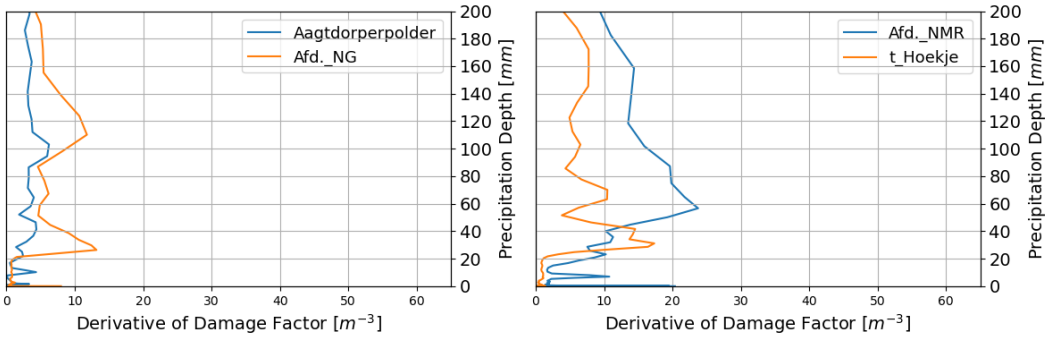


Figure 5.16: Type 1 polders, showing a damage profile in which damage more or less occurs at every volume.

Most Type 2 polders show unique damage profiles, sometimes with pronounced outliers. In Figure 5.17, all four polders show a clear tipping point, a volume beyond which damage begins to accumulate. In Afd. Z, this transition to damage is gradual, but in the other three polders, this is quite abrupt.

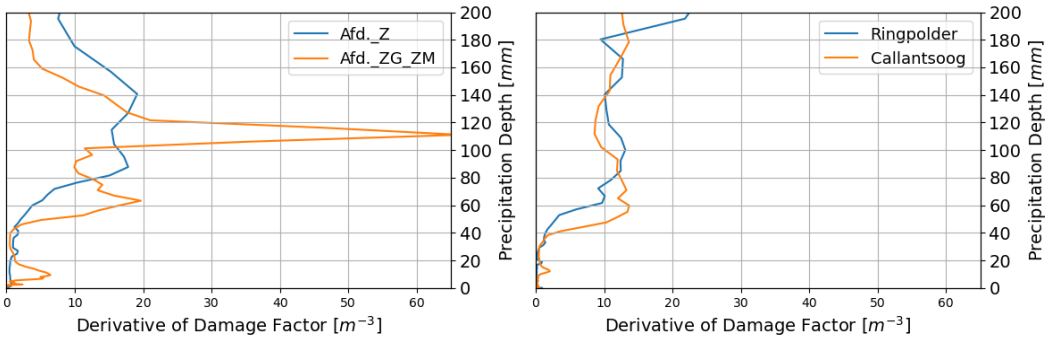


Figure 5.17: Type 2 polders, which often have unique profiles, with tipping points below which little damage occurs.

Type 3 polders are characterized by a comparatively low damage increase across all volume depths. The derivative of the VDC remains small, indicating that additional volumes lead only to marginal increases in damage. Consequently, the benefit of placing a tractor pump in these polders is minimal.

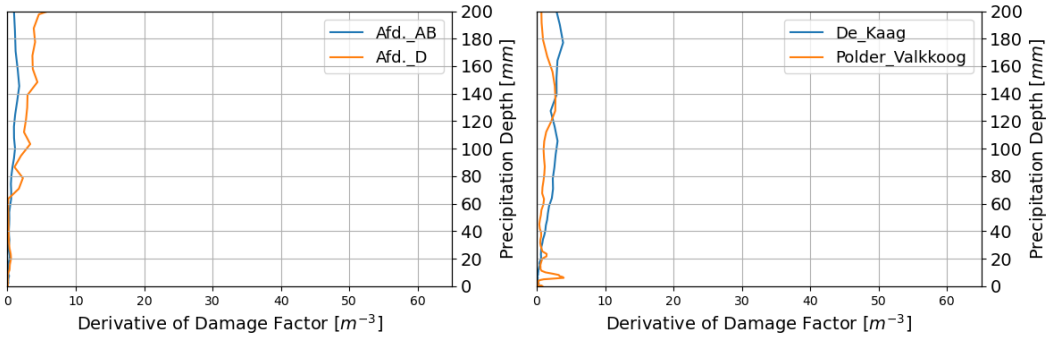


Figure 5.18: Type 3 polders, which do not require evaluation as damage remains negligible.

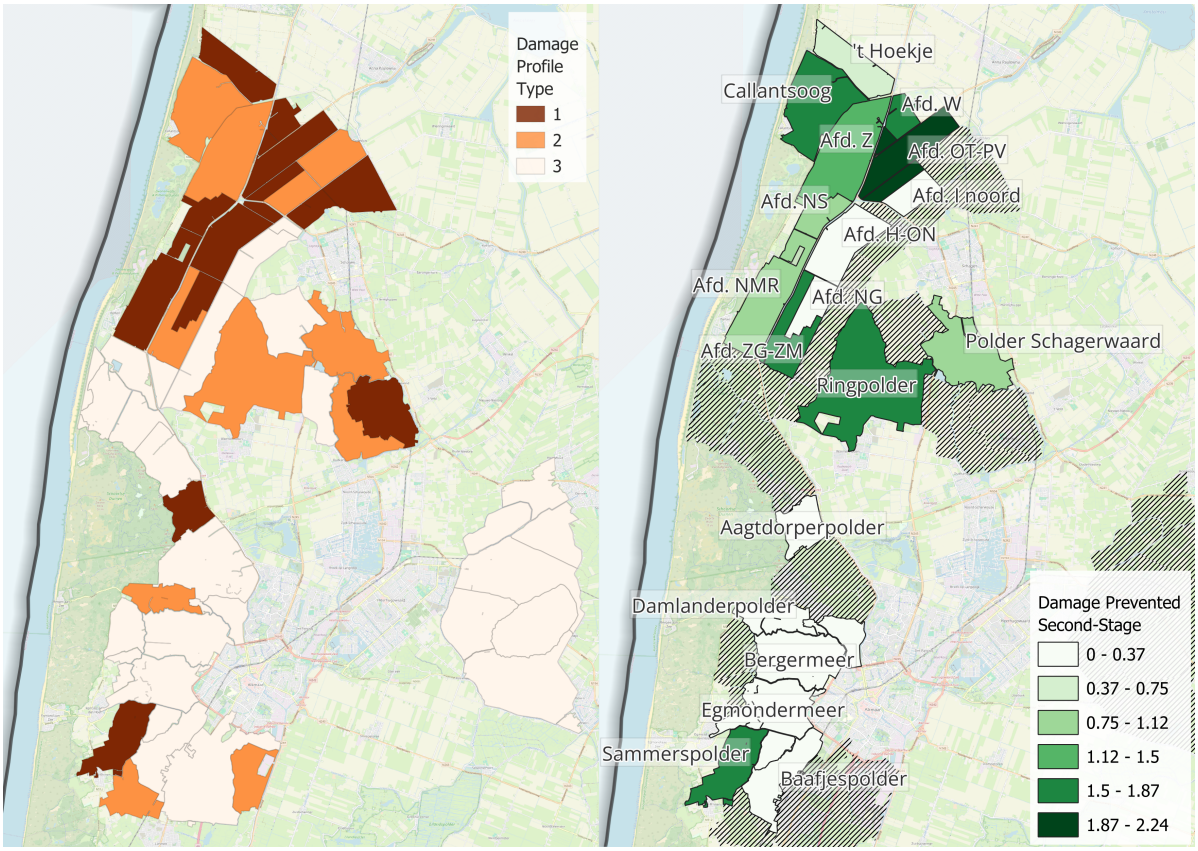


Figure 5.19: All damage profile types as well as the prevented damage from the optimization models. Notice that most polders that were not selected for the second stage are Type 3 polders.

5.4. Revisiting Results: Polder Types as a Method for Placement Deselection

The First-Stage optimisation filters the full set of 48 polders to 19 polders. However, the method of damage calculation is still omitting flood duration effects. Incorporating these will inevitably lead to a larger model. In the Second-Stage model, 19 polders and 12 timesteps produce a runtime of at least 15 minutes to obtain a reasonable gap. Other events may involve all 149 polders suitable for tractor pump placement, or additional pumps from contractors. As forthcoming improvements to the damage calculation will expand the model size, the screening into a subset of more promising polders is even more important. The three polder types provide such a screening method, not for polder selection but for deselection (i.e. creating a promising subset). It offers a fast way to omit polders with limited pump placement benefit and can be used either before the First□Stage model or replace it altogether if future model expansions make the two-stage setup impractical.

Table 5.3: Polder Types, First-Stage selection and pump allocation results.

Type	Total	Selected by First-Stage	Polders with Pumps	Note
Type 1	11	9	8	3 of the 5 not selected did not reach the tipping point
Type 2	12	7	7	
Type 3	25	3	2	

5.4.1. Type 3 Polders: Exclusion

Type 3 polders are characterized as polders with limited damage potential. This is apparent by their consistently low derivative values on the VDCs. This makes them a logical starting point as an exclusion criteria to obtain a smaller subset. Out of the 25 identified Type 3 polders, only three were selected for inclusion in the Second-Stage model: the Baafjespolder, Bergermeer and Egmondermeer. However, in the selection:

- Bergermeer did not receive any pumps in the Second-Stage;
- Egmondermeer received one 18 m^3/min pump for six timesteps, resulting in 0.06 damage factor prevention;
- The Baafjespolder received one 30 m^3/min pump over six timsteps, resulting in 0.25 damage factor prevention.

Compared to damage prevention observed in other polder types, these values are quite small. Looking at the selection of the polders and allocation of pumps more broadly:

- A total of 29 polders were not selected for the First-Stage;
- 32 out of the in total 48 polders were not allocated any pumps;
- Out of these 32, 22 were classified as Type 3;
- There are in total 25 Type 3 polders, of which only two were allocated pumps in the Second-Stage.

These numbers suggest that polders with Type 3 damage profiles were largely excluded by the First-Stage, and when included, only received limited pump allocations and damage prevention.

Table 5.4: Polder characteristics and pump allocation results for polders classified as Type 3.

Polder	Type	Removal Capacity [mm/day]	Area [ha]	Total Precipitation [mm]	Damage Baseline [-]	Prevented Damage Second-Stage [-]
Afd._AB	3	17.2	543	122	0.74	0
Afd._C	3	14.0	316	124	0.43	0
Afd._D	3	48.9	56	134	0.01	0
Afd._F	3	19.8	138	128	0.19	0
Afd._LQ	3	15.4	299	121	0.09	0
Baafjespolder	3	17.2	461	121	0.66	0.25
Bergermeer	3	23.1	846	124	1.66	0
De_Kaag	3	14.1	409	76	0.09	0
Egmondermeer	3	16.1	714	130	1.98	0.06
Groeterpolder	3	11.5	301	138	0.04	0
Grootdammerpolder	3	10.3	461	152	0.12	0
Hargerpolder	3	15.3	361	114	0.04	0
Hensbroek	3	15.2	567	66	0.04	0
Lage_Hoek	3	20.9	327	78	0.03	0
Leipolder	3	14.7	94	104	0.00	0
Obdam	3	42.9	905	71	0.72	0
Oosterzijpolder	3	12.5	1127	106	1.64	0
Philisteinsepolder	3	10.1	285	159	0.05	0
Polder_de_Berkmeer	3	15.1	287	73	0.05	0
Polder_de_Woudmeer	3	17.6	327	88	0.21	0
Polder_Valkkoog	3	14.1	512	111	0.35	0
Ursem	3	16.1	1065	57	0.07	0
Verenigde_Polders	3	13.8	916	127	0.52	0
Wimmenummerpolder	3	10.0	115	157	0.19	0
Wogmeer	3	13.5	691	56	0.08	0

5.4.2. Type 2 Polders: Tipping Points

Type 2 polder are defined by a distinct tipping point in their damage profile. This makes pump placement only beneficial if the water volume surpasses this volume point.
Out of the twelve identified Type 2 polders, five were not selected by the First-Stage model:

- Afd. KP, Speketerspolder and Vennewaterspolder did not exceed their tipping point during the event, and were not selected.
  - Afd. KP has a tipping point at around 100 mm, while the maximum water volume only reached 70 mm.
  - Speketerspolder has a tipping point at around 60 mm, but the maximum volume was 51 mm.
  - Vennewaterspolder has a tipping point of 120 mm and a maximum volume of 99 mm.
- Boekelermeer has a unique damage profile. It shows limited damage sensitivity, except for a single outlier between approximately 35-50 mm, shown in Figure 5.20. Outside this range, the damage increase per unit volume is low. In the baseline scenario, the maximum volume reached 70 mm. As the maximum volume is well beyond the 'damage peak', placement of pumps did not significantly reduce damage, and it was not selected.
- Afd. E has a similar damage profile as Boekelermeer, characterized by a damage peak with a maximum volume well above this peak.

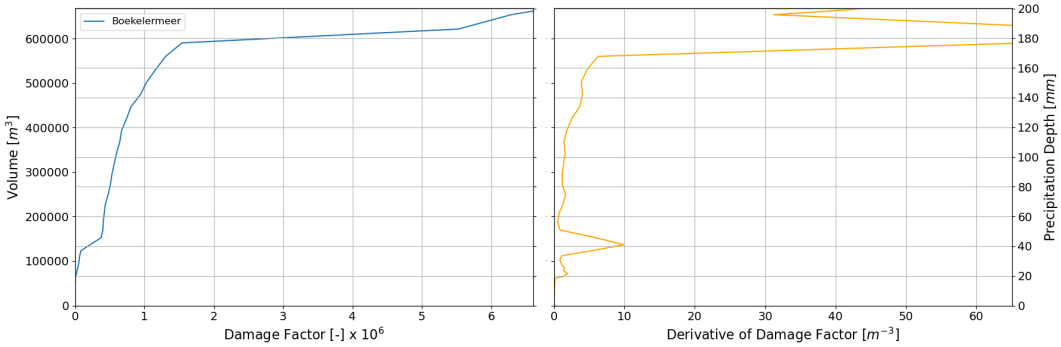


Figure 5.20: VDC and damage profile for Boekelermeer. Most damage accumulates between 35–50 mm of volume. Volumes above this peak result in little additional damage.

Visualizing the damage derivative, water volume and pump placement for Callantsoog in Figure 5.21 clearly shows how the model tries to prevent the tipping point from being exceeded. Callantsoog received 99 mm of total precipitation. In the baseline scenario, the maximum volume reached was 63 mm. With a tipping point at approximately 45 mm, the model placed pumps to ensure the maximum volume remained below this point. This resulted in significant damage prevention (1.65).

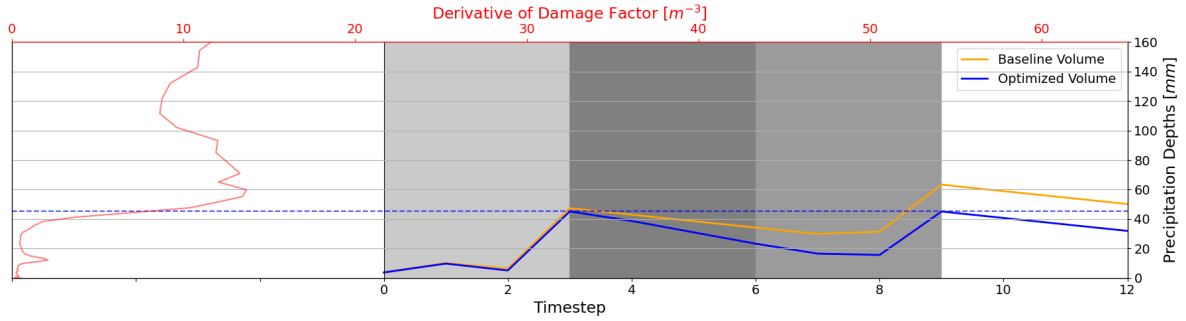


Figure 5.21: Polder volumes for Callantsoog. The dotted line indicates the maximum volume of the optimized scenario, which corresponds to observed tipping point.

Table 5.5: Overview of Type 2 polders. Polder that were not selected for the Second-Stage model are shown in gray. Polders with highlighted green volumes were polders of which the tipping points were not reached, explaining the non-selection. Volumes that are highlighted red are cases when pumps were placed to reduce the volume to below the tipping point.

Polder	Type	Tipping Point [mm]	Removal Capacity [mm/day]	Maximum Volume Baseline [mm]	Maximum Volume Optimized [mm]	Damage Factor Baseline [-]	Prevented Damage Factor [-]
Afd._E	2	40	13.3	75		1.10	0
Afd._I_noord	2	80	19.2	111	100	0.56	0.19
Afd._KP	2	100	15.0	70		0.14	0
Afd._Z	2	35	27.1	52	43	2.60	1.16
Afd._ZG_ZM	2	45	16.6	93	60	2.51	1.72
Boekelermeer	2	40	16.4	70		0.45	0
Callantsoog	2	45	17.5	63	45	2.09	1.65
Damlanderpolder	2	80	10.7	128	128	0.57	0.00
Polder_Schagerwaard	2	35	16.7	53	37	1.55	0.98
Ringpolder	2	35	14.4	65	56	6.23	1.53
Speketerspolder	2	60	14.2	51		0.25	0
Vennewaterspolder	2	120	13.6	99		0.38	0

Beyond volume-based analysis, identifying which land use types contribute to damage calculation can provide additional insights. The WSS database distinguishes 159 individual land use types. For clarity these have been grouped into seven broader categories. Appendix B lists every specific land use type and shows how each one has been assigned to a category. The seven categories are:

- Water, grass and nature;
- Infrastructure;
- Field crops;
- Horticulture;
- Recreation;
- Greenhouses;
- Buildings.

By comparing the categorized VDCs of the different land uses categories with the damage profile and the volumes, it is possible to identify which categories drive damage in a polder. These figures are shown for all polders selected for the Second-Stage in Appendix H. This is shown for all polders in the appendix, but visualized in Figure 5.22 for a single polder. in Afd. ZG-ZM, damage initially stems from field crops up to 40 mm, but transitions to buildings beyond that.

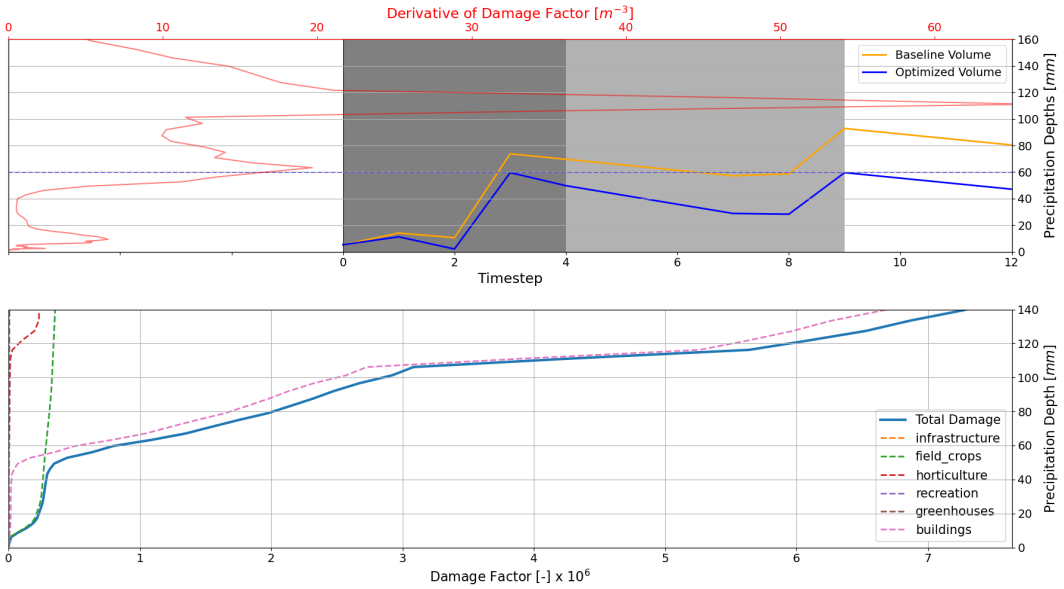


Figure 5.22: Afd. ZG-ZM. The upper plot presents the damage profile alongside the baseline and optimized water volumes, as well as the corresponding pump placements. The lower plot displays the contribution of different land use categories to the total damage. Initially, damage arises primarily from field crops, while beyond 50 mm of water volume, buildings drive damage increases.

5.4.3. Type 1 Polders

Out of the 11 polders classificied as Type 1, two were not selected by the First-Stage model: Afd. I-zuid and the Slootgaardpolder. Both show damage profiles that would normally justify inspection, yet other factors made them less attractive for pump allocation. Afd. I-Zuid is small (69 ha), with a gradually increasing derivative at larger volumes. Due to the small size, a single pump would 'push' the maximum volume down to these lower ranges, making alternative polders more suitable for allocation. Slootgaardpolder, on the other hand, received relatively little precipitation (79 mm). Slootgaardpolder received 79 mm of rain during the event, which is on the lower range of all the polders, and has a discharge capacity of 19.2 mm per day. That combination kept the maximum water volume to 36 mm, so additional pumps were unnecessary.

Aagtdorperpolder (Figure 5.23) was selected, but ultimately did not receive any pumps. Compared to other Type 1 polders, its damage derivative values are lower. This suggests lower damage potential per unit of additional volume. It was classified as a Type 1 polder, but an argument can be made for classifying it as a Type 3 polder.

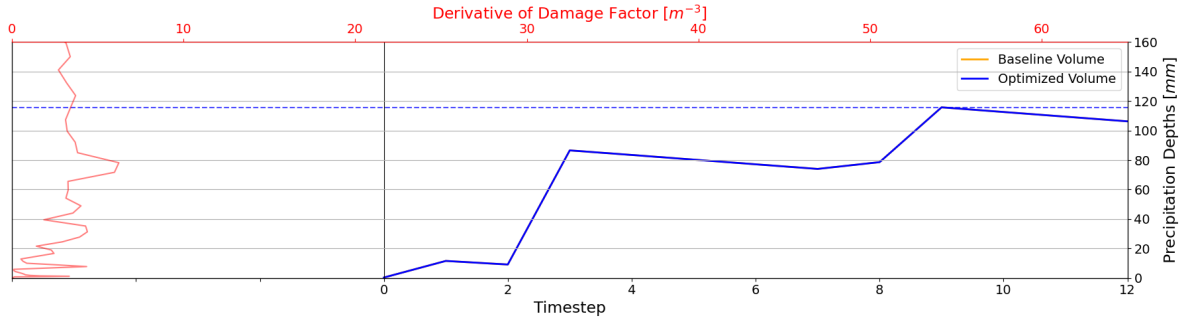


Figure 5.23: Volume and pump allocation (no allocation) for Aagtdorperpolder. Although this polder has a consistent damage profile, the relatively low derivative values may have limited its selection for pump allocation.

Table 5.6: Overview of Type 1 polders. All polders show high damage potential across the full precipitation range.

Polder	Type	Removal Capacity [mm/day]	Area [ha]	Total Precipitation [mm]	Damage Baseline [-]	Prevented Damage Second-Stage [-]
Aagtdorperpolder	1	12.7	284	144	1.10	0.00
Afd._H_ON	1	20.5	498	124	2.11	0.24
Afd._I_zuid	1	16.7	69	137	0.23	0
Afd._NG	1	18.1	215	123	0.97	0.19
Afd._NMR	1	25.0	692	123	6.72	0.91
Afd._NS	1	16.6	208	108	2.53	1.11
Afd._OT_PV	1	14.5	586	115	5.42	2.24
Afd._W	1	18.1	159	102	2.12	1.73
Sammerspolder	1	18.5	451	142	3.68	1.62
Slootgaardpolder	1	19.2	570	79	0.35	0
t_Hoekje	1	19.3	388	105	1.64	0.67

In several Type 1 polders, significant damage was prevented:

- *'t Hoekje* experienced a high baseline damage factor (1.64) due to large areas of land used for flower bulb cultivation, which in classification from the methodology is categorized under horticulture. The small size of the polder increased the effectiveness of pump placement, resulting in significant damage reductions (0.67 prevented).
- Afd. NS, Afd. NMR and Afd. OT-PV where 1.11, 0.91 and 2.24 was prevented. These damages in these polders were primarily linked to extensive horticulture land use.
- In Afd. W, houses in the village 't Zand inundate at around 80 mm of precipitation, leading to large damages (2.12 baseline and 0.38 prevented).

- Sammerspolders had a high baseline damage factor of 3.68, which was reduced by 1.62. Damage in this polders stems from both horticulture and recreation land uses.

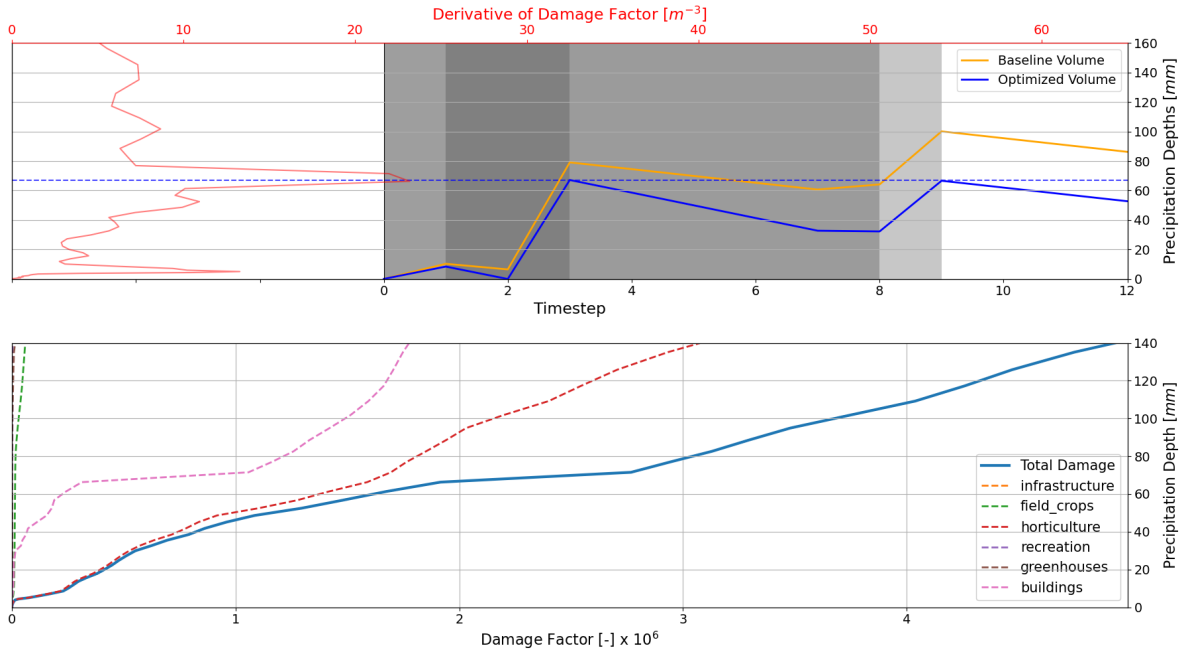


Figure 5.24: Sammerspolder. The top plot shows the baseline volume and optimized volumes, with pump placement indicated by gray shading. The model places pumps to ensure the maximum volume never exceeds the initial, unpreventable volume peak. The bottom plot shows that around 70 mm, recreational land use areas begin flooding.

6

Evaluation and Reflection

This chapter reflects on the functioning of the current model, and describes the transition from a single event case study to a (real time) short horizon model. It discusses how incorporating flood duration, altering the damage calculation, and reducing model size can improve both performance and practical applicability.

6.1. General Points

Study Approach

Using the WSS browser tool provides a method for estimating damage, but it does not help to quickly identify effective measures. However, since WSS provides access to its data, using this data in an optimization setting became the goal of this study. A key step was constructing VDCs and using them to interpolate variable values through PWL constraints. The main focus of this research was investigating how damage calculation through the VDCs could be used in an optimization model to support decision making in emergency flood management. The study is still ongoing, as further work is needed to refine the damage calculation by including flood duration as an influencing parameter and to improve the VDCs themselves.

Findings from this study are based on a case study. This makes results comparable to real-world insight, but can also introduce potential biases that might influence conclusions. To address this, the results chapter maintained a broad and generalizable perspective. Findings should be viewed as foundational rather than definitive. The research provides a proof of concept, but further refinement is necessary to develop a fully operational deployment tool.

Future Work: From Single Event Case Study to Short Horizon Model

The evaluation of the June 2021 event proved sufficient for this project, primarily because no context or prepared inputs were available at the time, and this was the most relevant event available. For a first proof of concept the single event focus kept the scope manageable. Future work, however, should shift toward a short horizon model that can be run quickly for different sets of input data, for example varying numbers of pumps or different precipitation forecasts. Monitoring runtime will be essential to keep the model useful in an operational setting. Upcoming improvements should therefore concentrate on runtime feasibility and on incorporating (real time) field data as model inputs.

From the literature study, it was apparant that models assisting decision makers should be flexible and modular to accomodate changes. While the model is modular, allowing for updates (e.g., new VDCs, additional pumps), it still lacks flexibility. Decision-makers often hesitate to adopt tools that are inflexible or slow [16], so further development should aim for practical performance and the integration of field data.

In its present form, the model functions as a deterministic optimization model: all inputs remain unchanged throughout each run. This is best explained as:

- No uncertainty in model data input (e.g. precipitation).
- Fixed tractor pump capacity and pumping station capacity.
- Fixed number of tractor pumps.

- Fixed amount of timesteps and length of a timestep
- Fixed nature of parameter constraints, such as: a generic travel time, maximum of three pumps per polder and one allowed relocation per pump.

The deterministic nature is not necessarily a limitation, provided that runtimes remain acceptable. In that case, the model can be used to explore multiple input scenarios, such as varying forecasts and number of pumps available. Rapid model runs are important if the model is to support uncertainty in the inputs, even if uncertainty is not directly included in the model structure.

6.2. Setup of VDCs, Notes and Improvements

Updating the VDCs

Setting up the VDCs was labor intensive but straightforward. They were set up using 2019 AHN3 and land use data from WSS, and would need updating to current land use data if applied to scenarios beyond the June 2021 flood event.

Damage Factor

Damage estimates in this study are based on a simplified methodology that assumes a uniform water level, inundation duration of 3 days and a recovery time of 5 days for all cells within a polder. While these assumptions produce absolute damage figures necessary for optimization models, the primary goal is relative prioritization rather than pinpointing exact financial losses. Using absolute damage figures allows for direct comparison between polders, but it does not account for localized variations in flood duration (soil saturation of crops), or drainage efficiency. To this end, the damage estimates are scaled down to a more abstract indication factor.

Indirect Damages Infrastructure

Indirect infrastructure damage is dependent on the type of infrastructure land use: railways and highways result in much larger losses than regional or local roads. According to the WaterSchadeSchatter manual, rerouting losses (known as *omrijdschade* in Dutch) are based on a separate network map containing individual road sections and intersections. For each road section, if more than 100 m^2 is flooded, indirect losses are triggered, irrespective of the inundation size. However, this network map is not publicly available, and the size and definition of the road sections are unknown. As a result, this study could not identify or calculate indirect infrastructure losses and has therefore omitted them entirely.

Flood Modeling Approach for the VDC Construction

An important finding of this study is that optimization models need a direct link between the variable to be controlled (water volume) and the resulting damage. Since the relationship between volume and the water level is nonlinear, an additional step is needed to estimate pumping impact on the water level. In this study, VDCs were used directly, so that the polder's surface area is already accounted for. As the direct VDC relation can be used, this also means that the uniform water level assumption can be replaced. Normally, flood routing depends on elevation gradients, sinks and existing waterways (drainage networks), as well as the impact structures have on local water levels. Since the VDCs are necessary for the optimization model, and not the DDCs, the VDCs could alternatively be constructed from inundation maps of simulation model runs (e.g. specific iteration time of precipitation events, called herhalingstijd in Dutch). Such inundation maps incorporate local flow routes, levees and terrain slopes, showing more accurate flooding patterns across the polder. Using these can help refine the VDCs so they better reflect actual terrain based inundation.

6.3. Reflections on Model Functioning and Improvements

Optimization problems grow rapidly in complexity and seemingly simple additions can be very difficult to formulate mathematically. The only reason the Second-Stage model manages to find a solution in manageable time (5% GAP in 25 minutes) is that the First-Stage screening reduces the polder count from 48 to 19 (found within 13 seconds). The growth in size and complexity is apparent when comparing the number of variables for the first and Second-Stage models: the First-Stage model had 1751 continuous and 1027 integer variable with 3 hour timesteps, whereas the Second-Stage model had 1061 continious and 9445 integer variable with 6 hour timesteps.

The increase in integer variables originates from the change in the formulation of the pump placement parameter from an order of magnitude $O(t \cdot p)$ in the First-Stage to $O(t \cdot k \cdot p)$ in the Second-Stage. Although the number of timesteps is halved, the parameter now has to account for 20 different pumps, increasing the number of integer variables tenfold. In linear programming, integer variables are first relaxed and then tightened using the cutting plane technique. Binary variables are bounded by 0 and 1, but integer or continuous variables require manually specified bounds. The tighter these bounds, the smaller the feasible solution space, which allows the cutting plane method to converge more quickly.

6.3.1. The Importance of Tight Constraints and Bounds

Tightening Variable Bounds

Solver performance is influenced by the size of the solution space. Tight bounds and constraints are therefore essential for model convergence speed. Even variables with fixed values, such as the auxiliary maximum volume variable that is assigned using '==' constraints, benefit from tight bounds. These bounds can be set either before the start of the model, or during the model run. For instance, updating the upper bound of the volume at each timestep based on the precipitation and volume of $t-1$ is effective. This is quite a powerful option, and not fully exploited in the current model. There are still significant potential gains in model performance by further updating these bounds in the framework.

Piecewise Linear Constraints

Solver performance can also be improved by refining the definition of piecewise linear (PWL) constraints, constructed from the VDC points. Currently, these use 2 cm increments over a 2 m range. Duplicate volume points were removed since PWL constraints require unique pairs, and the curves were trimmed to a maximum volume corresponding to 200 mm of precipitation. Reducing the number of points significantly reduces the computation time, as more points slow down interpolation time. Using the 2 cm increments results in a clustering of points at the lower volume range due to the nonlinear relation of water level and volume. A more efficient approach would be to tailor the points for each polder; for instance, using ten-centimetre steps for the initial points followed by smaller increments at higher volumes, perhaps following a logarithmic scale. Overall, selecting suitable PWL inputs and variable bounds enhances model performance as it allows for more efficient exploration of feasible solutions.

6.3.2. Reflection on Damage Assessment with Flood Duration

For most land use categories such as buildings, greenhouses and recreation, flood damage is more influenced by water depth than by flood duration. In contrast agriculture or infrastructure experience damage in proportion to how long they remain flooded. For agriculture the key issue is oxygen depletion in the root zone, which happens over time. Agriculture like field crops or horticulture are highly sensitive to how long crops remain submerged and in which season the flooding occurs [39]. Because flood duration is currently fixed in the VDCs, the model only minimizes the peak (maximum) flood volume. The result of this fixed duration is twofold:

- The model could overvalue the damages on agricultural land uses.
- There is no incentive for the model to allocate pumps after the maximum volume is reached.

How much the model could overvalue the damage is best shown, as done in Figure 6.1. This figure shows the VDC for the polder Afd. H-ON, which has intensive horticulture land use. If flood duration is included, the model might continue allocating tractor pumps after the maximum volume. However, the flood duration factor depends on the land use type. The average values for the seven used land use categories are shown in Table 6.1.

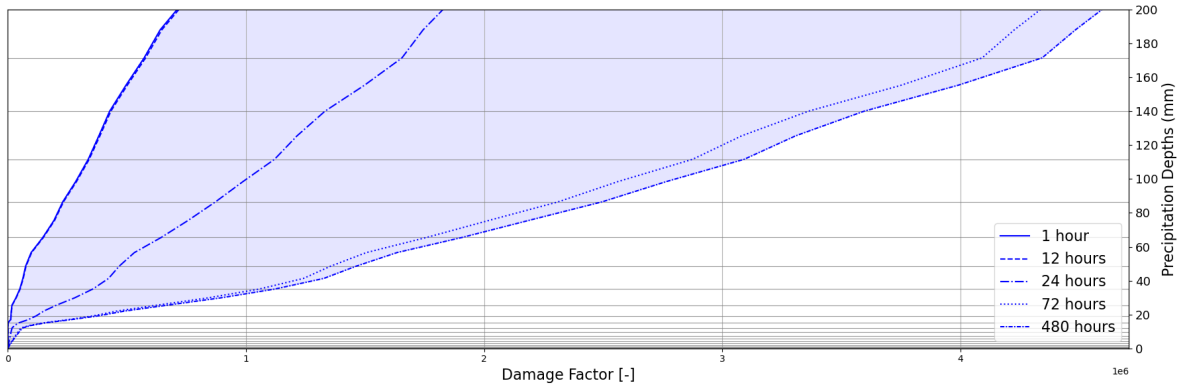


Figure 6.1: VDC of polder Afd. H-ON showing the spread in damage for all flood durations. It highlights the influence of duration to damages in agricultural areas.

Table 6.1: Table shows the average duration factor per land use category, based on the original 154 land use types from the asset values configuration file [24]. Actual values may vary per polder, depending on local land use composition.

Category	1 hour	12 hours	1 day	3 days	20 days
Water, grass and nature	0	0	0	0	0
Infrastructure	0.25	0.70	0.90	1.00	1
Field crops	0	0	0.20	0.45	1
Horticulture	0	0.57	0.67	0.87	1
Recreation	0.20	0.95	1	1	1
Greenhouses	1	1	1	1	1
Buildings	1	1	1	1	1

6.4. Improvements in the Linear Programming Model

Improvements to the model primarily involve incorporating flood duration in the damage estimation and reducing the calculation time. The first improvement requires a change in the VDC construction and objective function calculation, while the second depends on a different time horizon, number of evaluated polder and the pump placement variable modeling.

6.4.1. How to Incorporate Flood Duration in the Damage Calculation

The WaterSchadeSchatter calculates flood damage in a single cell as follows:

$$\text{Damage} = \text{Max Direct Damage} \cdot \gamma_{\text{depth}} \cdot \gamma_{\text{duration}} \cdot \gamma_{\text{season}} + \text{Indirect Damage per Day} \cdot \gamma_{\text{recovery time}} \quad (6.1)$$

Within WSS the flood duration is evaluated for every raster cell. In the current optimization model, however, the VDC is implemented as a one-dimensional function of water depth alone, with damage obtained by interpolating the maximum stored volume. To correctly adjust the damage formule in the VDC format to include flood duration, the formula must be split into direct and indirect terms, since the duration should only be applied to the direct damage term.

$$\text{Damage}_{direct} = \text{Max Direct Damage} \cdot \gamma_{\text{depth}} \cdot \gamma_{\text{season}} \quad (6.2)$$

$$\text{Damage}_{indirect} = \text{Indirect Damage per Day} \cdot \gamma_{\text{recovery time}} \quad (6.3)$$

$$\text{New Singe Polder Objective} = \text{Damage}_{direct} \cdot \gamma_{\text{duration}} + \text{Damage}_{indirect} \quad (6.4)$$

Where the $\gamma_{duration}$ is interpolated from the flood duration in hours. Additionally, the duration factor varies with the land use (as seen in Table 6.1), and the time that different parts of a polder are inundated is not uniform. Because the VDCs have aggregated all raster cells into a single curve, assigning a unique duration to each individual cell is not possible. A practical workaround for modelling this for VDCs is to evaluate duration per water level or volume increment. Within one increment the cells roughly get flooded at the same times, for which an average duration can be calculated. Capturing this requires three steps:

1. Split the VDCs into separate curves for each land use category.

2. Define increments for which the flood duration is tracked.
3. Interpolate $\gamma_{duration}$ for every land use category and increment.

These steps enable the model to represent both depth and duration in the objective function.

The proposed improved methodology for the optimization model is shown in Figure 6.3. It details splitting the VDCs into four land use categories:

1. Infrastructure
2. Field Crops
3. Horticulture
4. Buildings, Greenhouses and Recreation

For the first three categories, the direct and indirect damage is split as in Equations 6.2 and 6.3. The indirect damage for the these three categories can still be calculated as in the current model structure: use the maximum water volume to identify if it is flooded and using a fixed $\gamma_{recovery\ time}$. For the direct damages, the duration factor must be used as a correction factor on the increments.
For the fourth land use category, the damage calculation can remain the same as is currently done: interpolate the damage from the maximum water volume using PWL constraints. This VDC would contain both the direct and indirect damage.

Define Increments for Flood Duration Tracking

The current aggregrated VDC representation prevents us from applying land use specific duration factors, as we only want to correct the $\Delta Damage factor$ of that increment. A proposed solution is presenting the VDCs as seperate increments. This is shown conceptually in figure 6.2. To track flood duration for different increments (sections between the thresholds), we can define thresholds (A, B, C, etc.) and calculate how long the thresholds remain flooded. So that for each increment, the duration can be computed. Coding this is challenging because linear programming does not support logical operators such as if or else statements. A workaround is needed, which can be implemented using the Big M approach.

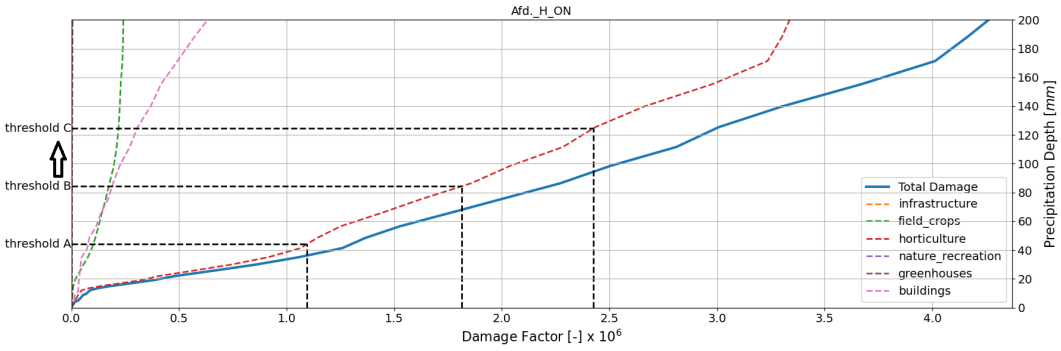


Figure 6.2: VDC of Afd. H-ON, showing how thresholds can be added to include the duration factor approach. The arrow between Threshold B and C indicate an increment.

Interpolating the Duration Factor with Big M Constraints

Conditional statements can be enforced through auxiliary binary variables and using indicator (Big M) constraints. This logic works as follows:

$$
\begin{aligned}
x &\geq y - M(1 - b) \\
x &\leq y + Mb \\
b &\in \{0, 1\}
\end{aligned}
$$

Here, b is a binary variable that indicates when x (volume) exceeds y (threshold). The parameter M should be a large value, for example the corresponding to 200 mm of precipitation. Assume that $b = 1$ indicates whether the volume is above threshold *B*. If $x > y$, then b must be 1 to satisfy both constraints. The other way around, if $x < y$, the large value of M forces b to be 0. Then by summing b across all timesteps, the entire duration for increment *AB* can be calculated. This method can be made to function for as many increments as desired.

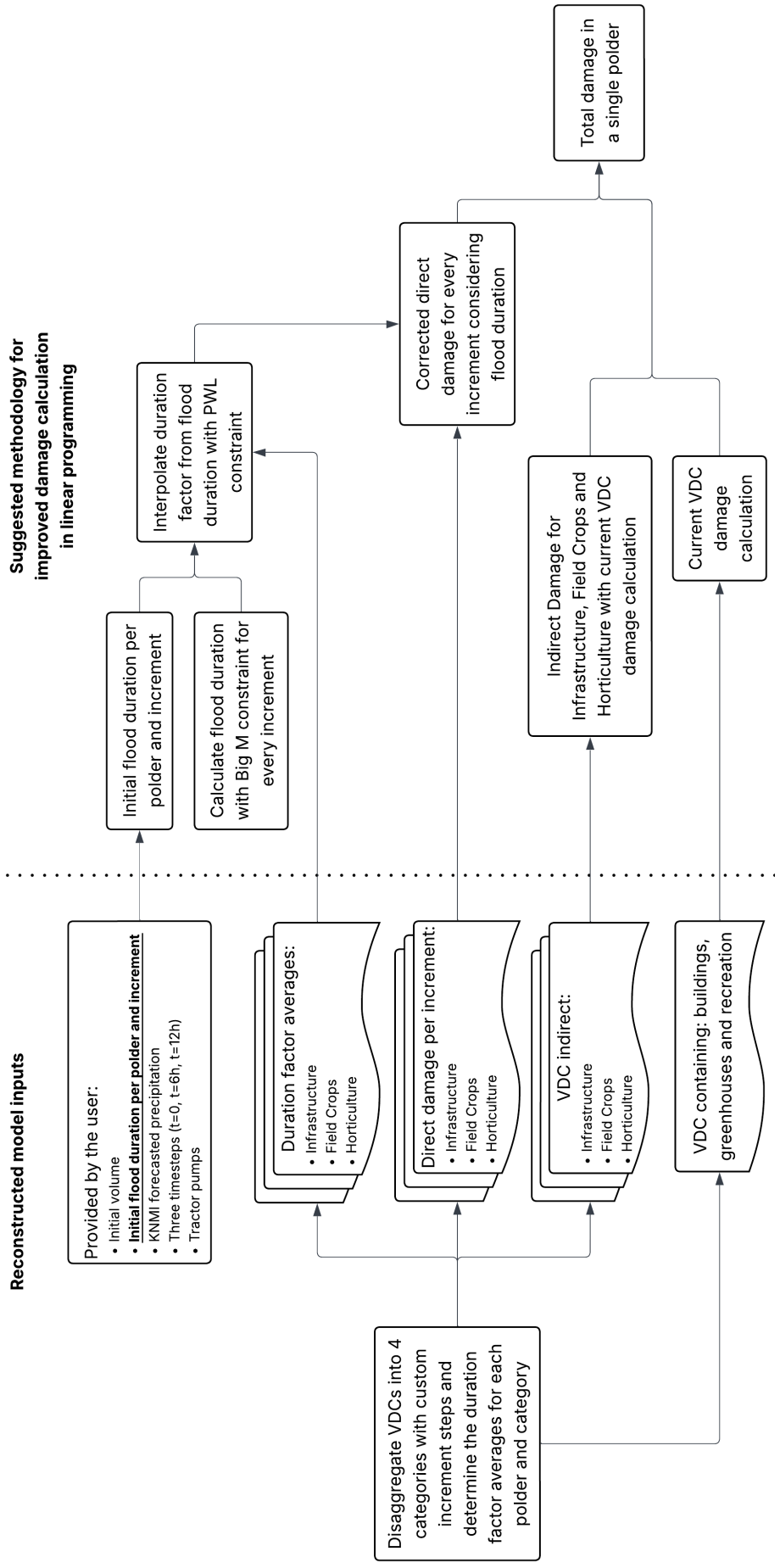


Figure 6.3: Suggested improved model methodology to incorporate duration in the optimization model.

Finalizing the Updated Objective Function

Once the duration is known, the duration factor ($\gamma_{duration}$) for the example of Afd. H-ON (table 6.1) can be interpolated using a piecewise linear (PWL) constraint for every increment:

$$duration^{AB} = \sum b * size\ timestep \tag{6.5}$$

$$\gamma^{AB}_{duration} = PWL(duration^{AB}, array_{\gamma_{duration}}) \tag{6.6}$$

$$Uncorrected\ damage^{AB}_{direct} = damage(B) - damage(A) \tag{6.7}$$

$$Damage^{AB}_{direct} = Uncorrected\ damage^{AB}_{increment} * \gamma^{AB}_{duration} \tag{6.8}$$

The same procedure applies to every increment and every land use category.

To obtain the polder total, the corrected damage values are summed for all increments, after which the indirect term is added. Indirect damage depends only whether a cell is flooded, with the user-specified $y_{recovery\ time}$. The factor is uniform in the polder and calculated once:

$$damage_{indirect} = PWL(V^{max}, VDC_{indirect}) \tag{6.9}$$

In the formulas below, T denotes the volume threshold and n the increment index.

$$Damage^{infrastructure} = damage^{infrastructure}_{indirect} + \sum_n damage^{T_{n+1}-T_n}_{direct} \cdot \gamma^{T_n-T_{n+1}}_{duration,infrastructure} \tag{6.10}$$

$$Damage^{field\ crops} = damage^{field\ crops}_{indirect} + \sum_n damage^{T_{n+1}-T_n}_{direct} \cdot \gamma^{T_n-T_{n+1}}_{duration,field\ crops} \tag{6.11}$$

$$Damage^{horticulture} = damage^{horticulture}_{indirect} + \sum_n damage^{T_{n+1}-T_n}_{direct} \cdot \gamma^{T_n-T_{n+1}}_{duration,horticulture} \tag{6.12}$$

For recreation, greenhouses and buildings the duration factor is 1, irrespective of the duration, so the existing damage calculation remains suitable:

$$Damage^{recreation,greenhouses,buildings} = PWL(V^{max}, VDC) \tag{6.13}$$

Summing the direct and indirect damages for all landuse categories finally becomes:

$$\begin{aligned} \text{Damage}_{\text{single polder}} = \text{Damage}^{\text{recreation, greenhouses, buildings}} + \text{Damage}^{\text{field crops}} + \\ \text{Damage}^{\text{horticulture}} + \text{Damage}^{\text{infrastructure}} \end{aligned} \tag{6.14}$$

This formulation preserves linearity in het model, allows depth and duration to be influential in polder damages, and leaves the indirect damage term in a polder unchanged.

6.4.2. Solution to Increased Model Size When Duration Enters the Objective Function

Introducing the flood-duration interpolation terms from the previous section increases the number of PWL interpolations that have to be calculated for each polder. This means that each branch and bound node becomes much more 'expensive'. Currently, for each node, there is 1 PWL constraint interpolation. But with flood duration, the number is $(n+1)*3+1$. Where n is the number of increments. If one polder has 40 increments this means 124 interpolations. So the number and size of the increments have to be carefully considered. With the increased number of interpolations, keeping the pump placement variable of order $O(t \cdot k \cdot p)$, with realistic calculation time is no longer realistic. A straightforward solution is to return the pump placement variable to $O(t \cdot p)$ and replace the individual pump indices with four generic capacity classes of 18, 20, 30 and $45\ \mathrm{m^3\,min^{-1}}$, as is done in the First-Stage. All modeled constraints must then be rewritten to fit this aggregated representation.

This down-scaling has two important consequences: Individual pumps can no longer be tracked and the difference between First- and Second-Stage disappears. As a result, there can no longer be a subset selection of promising polders. Without subset selection the model size depends directly on the number of polders, the number of time steps and the chosen number of polder increments. The current VDCs use 2 cm steps as input for the VDC points, as shown in the grid on Figure 6.1. This create very small increments at low volumes and an excessive number of PWL points. The selection of increment steps, whether by water level or by volume, thus becomes a key design choice in future improvements.

6.4.3. Reducing the Number of Polders: Screening Strategy With Polder Damage Profiles

In future improvements, all 149 polders suitable for pump placement will need to be evaluated. Because a model of reduced placement variable order can no longer can distinguish between a First- and Second-Stage, subset selection based on a two stage approach is no longer possible. Instead, the classification system with Type 1, 2, and 3 polders can serve as a replacement. This system does not identify the best polders but helps to deselect or exclude polders where pumps offer minimal benefit (Type 3) or where damage remains below a tipping point (Type 2). In this way, the classification functions as a preselection method that reduces the solution space. The currently constructed damage profiles and polder classification is based based on a fixed flood duration of 72 hours. For this duration, the $\gamma_{duration}$ is quite large, as can be seen in Figure 6.4. Because shorter durations result in smaller damage factor values, the seventy-two-hour scenario is conservative. Since damages cannot increase when duration shortens, profiles with already low damage are suitable for exclusion.

The method can still be improved. They could be classified using all durations, as shown in figure 6.4. It should also be noted that the polder types stem from visual analysis, which is qualitative. Despite its simplicity, this system is a starting point, which can be adapted or refined over time.

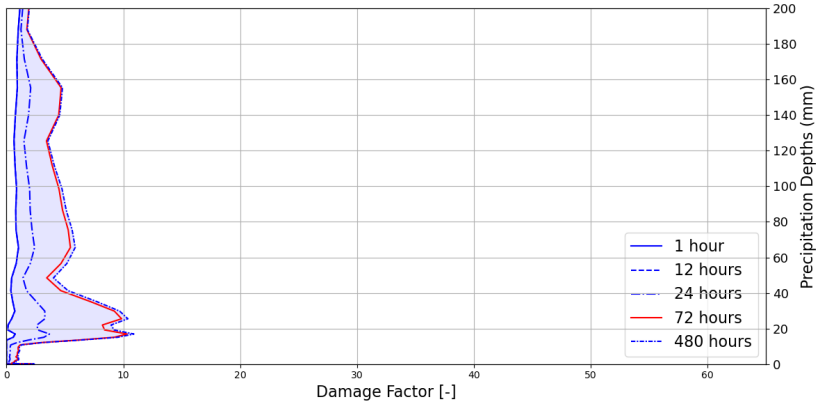


Figure 6.4: Damage profile for all uniform flood durations in polder Afd. H-ON. The red line is the currently used duration.

6.4.4. Wrapping up: Short-Horizon Model

In addition to the polder selection and a smaller pump placement variable size, improvements should focus on constructing a short-horizon model. The current case study approach of a 3-day window was fine for initial concept creation, but with the use of dynamic flood duration and the increased model size, runtimes will not be feasible with 25 timesteps (First-Stage) or 13 timesteps (Second-Stage). This Short-Horizon model should use at most three timesteps (t=0, +6h, +12h), which can be run with real time initial conditions and forecasts.

Model Inputs

To run the model on a short time horizon, it requires measured input data instead of generic initial conditions. The following components are needed:

- Initial water volume. HHNK has the *Vullingsgraad* program, which estimates the current filling in each polder based on logger data. This can serve as the starting point for each model run.
- Expected precipitation. In Appendix I, a script is included that shows how KNMI forecasts can be downloaded and spatially allocated to individual polders.
- Current pump placements. Knowing where tractor pumps are located in the field allows the user to fix their placements in the model. This reduces the number of free variables in the optimization model and leads to faster solving time.
- Polder discharge capacity. More accurate measurements of this capacity improve the realism of the model results.
- User interface. A clear interface is needed to manage both data input and output. LP models require structured input files and produce long variable lists that are not interpretable without post processing. Possible features include a map to toggle pump locations and sliders to adjust timestep length and number of steps.

Linear Programming or Alternative Path Forward

For practical use, code based models that do not use a solver, where the user or techniques such as Monte Carlo decided the pump placements, could be an alternative to a LP approach. Models constructed without a solver offer faster runtimes, can incorporate uncertainty in precipitation forecasts and allow for better modelled detail such as polder-to-polder travel times.

However, code based models require the user to provide the pump placements, and with up to 149 polders and 20 pumps, that could be a large number of placements. Even larger if an increased allowed number of pumps in a single polder. The drawback of manual pump placement is that it could lead to suboptimal placements, particularly when decisions in early timesteps influence later placements. This reflects a tradeoff: code based models are easier to work with and have more flexibility, but may not produce optimal solutions that optimization models can.

7

Conclusion

7.1. General Conclusions from Case Study

This thesis set out to answer the main research question: "*What framework is required to optimize the allocation of tractor pumps to candidate polders?*" Adressing this question demands a model that links inundation damage in polders to the preventive effect tractor pumps can have through pre-computed relations, and then to search for the pump placement set that minimizes the total damage over all polders. The required precomputed relations are Volume Damage Curves (VDCs), one for each polder, constructed from terrain elevation, land use, land use asset values and the WaterSchadeSchatter method of damage calculation. The VDC format allows direct integration in an optimization model, so that no simulation or damage assessment is needed during the optimization run. Pump placement is optimized with a Linear Programming (LP) model that allocates the available tractor pumps over all polders and timesteps under modeled constraints. Using the VDCs for damage calculation, it selects the combination of placements that minimizes the total damage. The model was tested on the three day flood in Hoogheemraadschap Hollands Noorderkwartier from 18 to 20 June 2021, with 48 selected polders and 20 tractor pumps of varying capacities.

Answers to the Sub-questions

1. **What are the main insights from applying the prototype to the 2021 flood event, in terms of model performance and limitations?**

   Succesfull aspects

   - VDCs are well suited to form the basis of the LP model.
   - The Two-Stage approach worked well: the First-Stage produced a subset of 19 candidate polders, after which the Second-Stage tracked individual pumps in greater detail.
   - The Slack Volume variable prevented infeasibility caused by negative volumes.
   - Introducing specific pump-to-polder travel times is infeasible, but a generic three hour travel penalty was succesfully introduced.
   - The study introduced polder classification based on the derivative of each VDC, which labels polders as Type 1 (always relevant), Type 2 (tipping point dependent) or Type 3 (low priority), and these categories closely mirror which polders are deselected by the first-stage model.

   Observed shortcomings

   - Damage in each polder is currently linked only to the maximum volume. As a result, there is no incentive to deploy pumps after the volume peak.
   - Inundation duration is fixed at 72 hours in the current VDCs, which can overestimate agricultural damage.
   - Individual pump tracking in the Second-Stage, with binary variables of order $O(t \cdot k \cdot p)$ limits scalability.

   These findings call for adjustments to both the VDC construction and the structure in the optimization model.

2. **What improvements are required in the damage assessment method and model structure to enhance scalability and usability?**
    - Deaggregate the VDCs into land use categories. For the VDCs containing agriculture and infrastructure, split the VDC further into direct and indirect damage parts, and correct the direct damage for the flood duration per elevation increment.
    - Calculate the inundation duration and its representing factor with Big M constraints and piecwise linear interpolation.
    - Replace the Two-Stage approach with a faster single stage with a Pump Placement variable of order $O(t \cdot p)$.
    - Use the polder classification types to preselect a polder subset before solving, so that the runtime remains manageable.

    These changes require new input data for each polder before the model can be used in practice.
3. **What is needed to make the optimisation model usable as an operational decision support tool during emergencies?**
    - Run the model with a short horizon of three timesteps: now, +6 hours and +12 hours.
    - Feed the model with real-time initial volumes derived from water levels and with precipitation forecasts assigned to polders.
    - Store inundation duration for every polder elevation increment after each run so that the next run include them as initial duration.
    - Allow an option to 'override' pump placements, so that they can be fixed for future timesteps.

Although the framework still needs refinement, it offers a good basis for tractor pump allocation. With the proposed data updates, model reformulation and real time inputs it can evolve into a practical decision support instrument for flood response.

7.2. Recommendations

Recommended Follow-Up Work for Model Development

- *Update and Refine VDCs*
  Incorporate the most recent AHN5 elevation and land use data to ensure accurate flood damage estimation.
- *Include Indirect Damages for Infrastructure*
  Indirect losses for infrastructure is still missing, because the separate road maps could not be downloaded through the Lizard API and manually setting up the maps and identifying when a 100 m^2 floods in the aggregated polder csvs proved too difficult.
- *Reconstruct the VDCs into Four Categories*
  Split the VDCs into four land use categories: Infrastructure, Horticulture, Field Crops and a combined group for Greenhouses, Buildings and Recreation. For the first three categories, separate direct and indirect damages and construct the direct group without the fixed $\gamma_{duration}$ factor.
- *Consider Constructing the VDCs with Improved Flood Dynamics*
  Instead of assuming a uniform water level based on the lowest-filling principle, using inundation maps provides a more realistic representation of flood routing. Because VDCs do not require a uniform water level, they can use inundation maps generated from hydrodynamic simulations based on precipitation scenarios with specific return period (herhalingstijd in Dutch).
- *Calculate Flood Duration per Elevation Increment Dynamically*
  Add flood duration as a model input and choose non-uniform increment steps, concentrating model detail where damages rise fastest.
- *Redesign the model to accomodate the pump placement variable of order* $O(t \cdot k)$
  The current Second-Stage tracks individual pumps, giving the variable a size of $O(t \cdot k \cdot p)$ and lengthening solution times. Reformulate the model so each pump type is counted only per time step, keeping runtimes short enough to test multiple precipitation or initial condition scenarios.

Recommendations for HHNK

- *Define Threshold for Pump Allocation*
  Decide at what water level in the polder tractor pump allocation becomes optional (for example: $< X\ mNAP$ no pumps are assigned). Also specify when and where pumps should not be placed using the classification systems and the Tipping Points.
- *Extend and/or Improve the Damage Profile Classifications*
  Apply the Type 1, 2 or 3 classification system to all polders in the region, as the current analysis covered only 48 out of 149. The classification method of the damage profile can be adjusted as needed. This supports faster identification of high-impact polders and helps exclude low-return areas from the pump allocation process.
- *Account for Polder Discharges*
  Pump placement also depends on the discharge capacity of individual polders. In this study, the total discharge was estimated by summing the maximum capacities of all relevant pumping stations and applying a correction based on an HHNK discharge survey (Appendix C). Collect measured outflow per polder to replace these estimates and improve model accuracy.
- *Develop a Short Horizon Model*
  Include the initial volume via the 'Vullingsgraad', the latest KNMI precipitation forecast, and current pump positions so the model knows whether a pump must stay in place. Limiting the number of time steps keeps run time short while still allowing for uncertainty in rainfall and pump performance.

Broader Recommendations for Other Water Boards

- *Construct VDCs or DDCs for each Polder*
  Even if tractor pumps are not part of the calamity response strategy, building VDCs or Depth Damage Curves reveals the water levels or rainfall volumes at which damage accelerates and helps to identify polder tipping points.
- *Use WaterSchadeSchatter Data Directly for Quick Damage Calculation* Even if tractor pumps are not part of the calamity response strategy, WSS land-use layers, asset values, and the damage formula behind its portal can be freely obtained. By downloading these datasets instead of uploading flood maps, it is possible to compute expected damage with other simplified flood scripts. If resource allocation is not the objective, the calculation can be done with conventional non-LP code.

References

[1] Hunze en Aa's. *Kennisbank*. URL: https://kennis.hunzeenaas.nl/index.php/Id-ca1bee24-116c-4b44-94a9-a16e3592015b (visited on 02/23/2025).

[2] S. Afshari et al. "Comparison of new generation low-complexity flood inundation mapping tools with a hydrodynamic model". In: *Journal of Hydrology* 556 (2018), pp. 539–556.

[3] BBA Pumps. *BBA B300 Pump Specifications*. URL: https://www.bbapumps.com/b300-t3wgt-tractor-driven-self-priming-emergency-pump (visited on 02/05/2025).

[4] B. Becker et al. "Optimization methods in water system operation". In: *WIREs Water* 11 (2024).

[5] United Weather Centres-West. *HARMONIE Cy43*. URL: https://www.knmidata.nl/open-data/harmonie (visited on 12/29/2024).

[6] Dymaxionlabs. *QGIS zonal statistics multiband*. URL: https://github.com/dymaxionlabs/qgis-zonal-statistics-multiband/blob/master/README.md (visited on 12/29/2024).

[7] *Eindadvies Beleidstafel wateroverlast en hoogwater*. Beleidstafel wateroverlast en hoogwater. 2022.

[8] Gurobi. *Mixed-Integer Programming (MIP) - A Primer on the Basics*. URL: https://www.gurobi.com/resources/mixed-integer-programming-mip-a-primer-on-the-basics/ (visited on 10/21/2024).

[9] S. Hartmann and D. Briskorn. "A survey of variants and extensions of the resource-constrained project scheduling problem". In: *European Journal of Operational Research* 207 (1 2009).

[10] HHNK. *Organisatie*. URL: https://www.hhnk.nl/organisatie (visited on 04/17/2025).

[11] O. Hoes. *Vergelijking WaterSchadeSchatter en de Schade en Slochtoffer Module: overeenkomsten, verschillen en aanbevelingen*. Tech. rep. STOWA, 2024.

[12] O. Hoes, F. Nelen, and E. van Leeuwen. *WaterSchadeSchatter (WSS) gebruikershandleiding*. Tech. rep. STOWA, 2013.

[13] E. Kalvelagen. *MIP solving stopping criterion*. URL: https://yetanothermathprogrammingconsultant.blogspot.com/2019/11/mip-solver-stopping-criteria.html (visited on 02/06/2025).

[14] J. Kang, M. Su, and L. Chang. "Loss functions and framework for regional flood damage estimation in residential area". In: *Journal of Marine Science and Technology* 13 (2005).

[15] H. Kreibich et al. "Development of FLEMOcs - a new model for the estimation of flood losses in the commercial sector". In: *Hydrological Sciences Journal* 55.8 (2010), pp. 1302–1314.

[16] J.G. Leskens et al. "Why are decisions in flood siaster management so poorly supported by information from flood models?" In: *Environmental Modelling and Software* 53 (2014), pp. 53–61.

[17] J. van der Lingen. *Specificatie bladen Calamiteiten materieel*. Hoogheemraadschap Hollands Noorderkwartier. 2022.

[18] D.P Loucks and E. van Beek. *Water Resource Systems Planning and Management*. Springer Nature, 2017.

[19] H. McGrath et al. "A comparison of simplified conceptual models for rapid web-based flood inundation mapping". In: *Natural Hazards* 93 (2018), pp. 905–920.

[20] B. Merz et al. "Review article "Assessment of economic flood damage"". In: *Natural Hazards and Earth System Sciences* 10 (2010), pp. 1697–1724.

[21] S. Mohd Mushar et al. "Flood Damage Assessment: A Preliminary Studies". In: *Environmental Research, Engineering and Management* 75 (2019), pp. 55–70.

[22] O. Naud et al. "Chapter 4 - Support to decision-making". In: *Agricultural Internet of Things and Decision Support for Precision Smart Farming* (2020), pp. 183–224.

[23] Nelen and Schuurmans. *Lizard API*. URL: https://stowa.lizard.net/api/v4/rasters/30ffe21c-4842-4724-8dea-9d69c12311e8/.

[24] Nelen and Schuurmans. *WaterSchadeSchatter*. URL: `https://www.waterschadeschatter.nl/damage/` (visited on 12/31/2024).

[25] A.E.M. van Oostrum. *Concept Visie Crisisbeheersing 2030*. Tech. rep. Hoogheemraadschap Hollands Noorderkwartier, 2024.

[26] J. Pool and D. Laubscher. “Design-based research: is this a suitable methodology for short-term projects?” In: *Educational Media International* 53 (1 2015).

[27] B. Restemeyer, M. van den Brink, and J. Woltjer. “Between adaptability and the urge to control: making long-term water policies in the Netherlands”. In: *Journal of Environmental Planning and Management* 60 (2017), pp. 920–940.

[28] N.S. Romali, Z. Yusop, and Z. Ismail. “Flood damage assessment: A review of flood stage-damage function curve”. In: *ISFRAM* (2015), pp. 147–159.

[29] J. Teng et al. “Flood inundation modelling: A review of methods, recent advances and uncertainty analysis”. In: *Environmental Modelling and Software* 90 (2022), pp. 201–216.

[30] Wageningen Universiteit. *KWIN-AGV gewasprijzen*. Yearly actualisation of crops prices and yield. 2024.

[31] E. Van den Berg and W. Kouwenhoven. “Ontwerponderzoek in vogelvlucht”. In: *Tijdschrift voor lerarenopleiders* 29 (4 2008).

[32] L. Van Turnhout, D. Andriessen, and P. Cremers. *Handboek ontwerpgericht wetenschappelijk onderzoek*. Koninklijke Boom uitgevers, 2023.

[33] R. Versteeg et al. *Meteobase: online archief van neerslag- en verdampingsgegevens voor het waterbeheer*. Tech. rep. STOWA, 2013.

[34] A. Voinov and F. Bousquet. “Modelling with stakeholders”. In: *Environmental Modelling and Software* 25 (2010), pp. 1268–1281.

[35] A. Voinov et al. “Tools and Methods in Participatory Modeling: Selecting the Right Tool for the Job”. In: *Environmental Modelling and Software* 109 (2018), pp. 232–255.

[36] M.J.L. van de Vondervoort. *Crisisbeheersingsplan HHNK 2020*. Tech. rep. Hoogheemraadschap Hollands Noorderkwartier, 2020.

[37] M.J.L. van de Vondervoort. *Evaluatie bestrijding wateroverlast juni 2021*. Tech. rep. Hoogheemraadschap Hollands Noorderkwartier, 2021.

[38] L. Wang et al. “A review of the flood management: from flood control to flood resilience”. In: *Heliyon* 8.11 (2022).

[39] *WaterSchadeSchatter gebruikshandleiding*. Nelen and Schuurmans. 2024.

[40] J. Yazdi, B. Zahraie, and S. Neyshaobouri. “A Stochastic Optimization Algorithm for Optimizing Flood Risk Management Measures Including Rainfall Uncertainties and Nonphysical Flood Damages”. In: *Journal of Hydrological Engineering* 21 (5 2016).

[41] M. Zandvoort and R. de Graaff. *Leren van Wateroverlast*. Tech. rep. STOWA, 2019.

[42] J. Zhang et al. “A survey for solving mixed integer programming via machine learning”. In: *Neurocomputing* 519 (2023), pp. 205–217.

A

Polder Selection

The light blue (9) and yellow (48) polders in figure A.1 are those for which motorkap posters have already been made. The posters for the brown polders are set to be constructed somewhere in 2025. For this study, only the places where posters have already been made have been included in this study. These nine light blue polders were excluded from the analysis because the removal capacity from the polders was either unknown by the waterboard, or used other methods of discharge such as weirs or free drainage. This is visualized in figure C.1 from Appendix C.

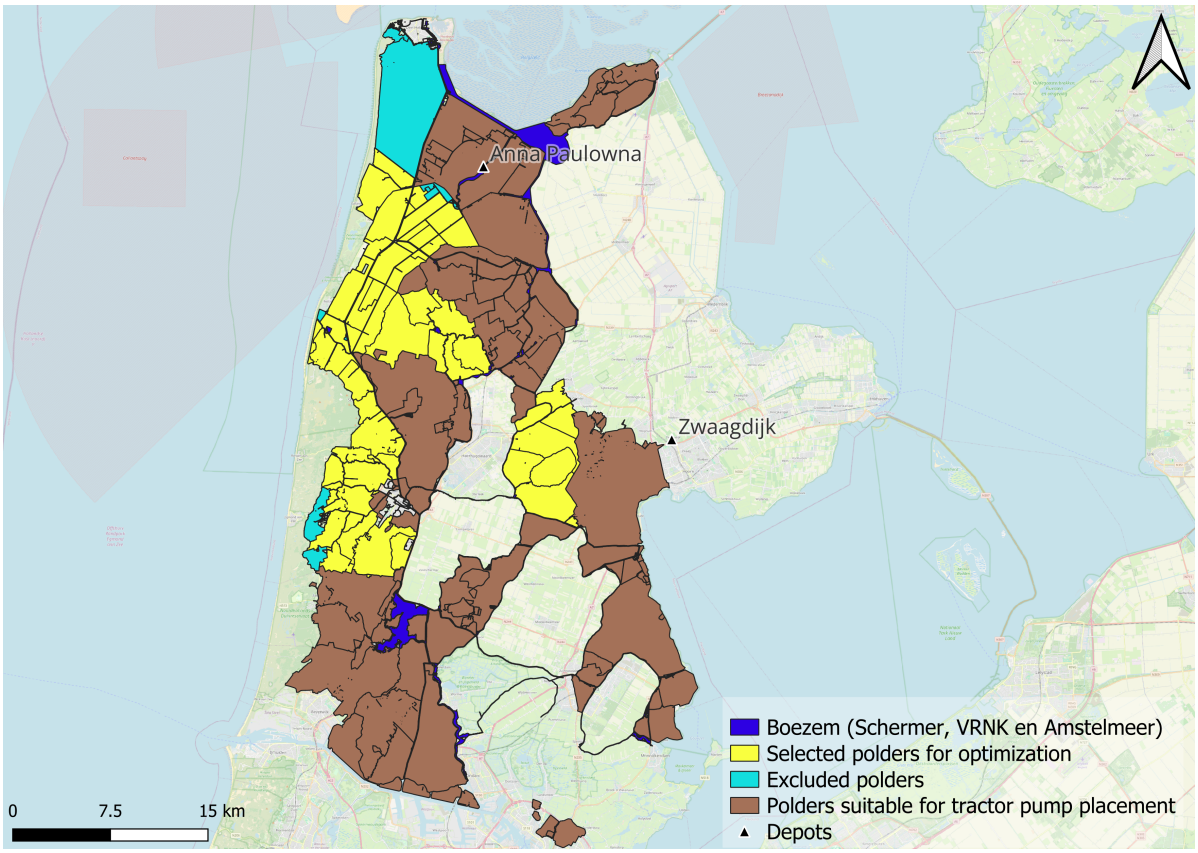


Figure A.1: Selection of polders by HHNK that are suitable for tractor pump placement.

B

Aggregation of WSS Landuse Categories

Table B.1: Categories showing which landuses and configuration file section number are added to the category.

Category	Landuse and configuration file section number
Water, gras and nature	bassins (35), berm (44), bezinkbak (34), binnenwater (51), bos (39), bos / natuur (43), braak (73), buitenwater (52), gras (42), groenvoorziening (40), natuur (115), niet ingevuld (14), opslagtank (33), overig (24), overig gras/groen (41), overige gebruiksfunctie (13), spoorberm (45), transformatorstation (32), vliegveld (30), water (50), water (156), water (254)
Infrastructure	Fietspad (166), Voetpad (165), lokale_weg (28), overige wegdelen (29), regionale_weg (26), snelweg (25), spoor (31), verkeerseiland (27)
Field crops	aardbeien_op_stelling (55), aardbeien_open_grond (56), aardperen (57), akkerbouw (59), andijvie (60), asperges (62), augurk (63), blasrammenas (64), bloemkool (68), boerenkool (69), bospeen (72), broccoli (74), bruinebonen (75), chinesekool (77), cichorei (78), consumptieaardappelen (79), courgette (80), erwten (83), gerst (86), granen (87), groente_in_open_grond (90), haver (91), hennep (92), ijsbergsla (93), kapucijners (94), klaver (98), knoflook (99), knolselderij (100), knolvenkel (101), komkommer (102), koolraap (103), koolrabi (104), koolzaad (105), kruiden (106), luzerne (108), mais_corncob (109), mais_energie (110), mais_korrel (111), mais_snij (112), mais_suiker (113), paksoi (117), pastinaak (118), peulen (120), pompoen (121), pootaardappelen (122), prei (123), pronkbonen (124), rabarber (126), radijs (127), rode_bieten (129), rodekool (131), rogge (132), schorseneren (134), selderij (135), sla (138), sojabonen (139), sperziebonen (140), spinazie (141), spitskool (142), spruitjes (144), suikerbieten (145), tarwe (146), triticale (148), voederbieten (154), weidehooi (157), zetmeelaardappelen (163)
Horticulture	appelen (61), blauwebessen (65), bloembollen (66), bloembollen_en_sierteelt (67), boom_en_heesterkweek (70), bos_en_haagplanten (71), buxus (76), cranberry (81), frambozen (84), fruitteelt (85), kersen (95), kersen_zuur (96), kerstbomen (97), laanbomen (107), miscanthus (114), notenbomen (116), peren (119), pruimen (125), rodebessen (130), rozen (133), vaste_planten (152), wijndruiven (158), zwartebessen (164)
Recreation	bedrijventerrein (16), begraafplaats (20), dagrecreatief terrein (17), glastuinbouw (22), sportterrein (19), verblijfsrecreatief terrein (18), volkstuinen (21), woongebied (15)
Greenhouses	kas (7)
Buildings	bijeenkomstfunctie (9), celfunctie (3), gezondheidszorgfunctie (12), industriefunctie (4), kantoorfunctie (5), logiesfunctie (8), onderwijsfunctie (11), sportfunctie (10), winkelfunctie (6), woonfunctie (2)

C

Pumping Station Specifics

The polders excluded from the figure A.1 are the polders: Koegras, Huisduinen, Hazepolder West, Mosselwiel, Oningepolderde landen onder Egmond Binnen, 't Zijer Eilant, Afd. W - Mosselwiel, Westerkogge, Waterberging LQ.
The removal capacity of the Ringpolder was manually altered, as there are no pumping stations in this polder. The polder contains a large weir capable of large quantities of discharge. According to the waterboard, the discharge capacity of this polder is ranges from 14.4 to 22 mm/day. Therefore, the capacity is manually adjusted to 22 mm/day for this polder. Another polder that was manually altered was Obdam, which initially showed a removal rate of 42.9 mm/day based on a pumping capacity of 100 m3/min, this was changed to 22 mm/day (with a matching capacity of 51.3 m3/min) based on capacity from figure C.1.

Table C.1: All pumping stations capacities.

Polder	Pumping Station Name	Pumping Station Capacity
Aagtdorperpolder	Aagtdorperpolder	25
Afd. AB	AB	65
Afd. C	C	30.8
Afd. D	D	19
Afd. E	E	52
Afd. F	F	19
Afd. H-ON	H	35
Afd. H-ON	ON	36
Afd. I Noord	I Noord	27
Afd. I zuid	I Zuid	8
Afd. KP	KP	37
Afd. LQ	LQ	32
Afd. NG	NG	27
Afd. NMR	Grote R	27.1
Afd. NMR	NM Zuid	70
Afd. NMR	Kleine R	10
Afd. NMR	NM Noord	13
Afd. NS	NS	24
Afd. OT-PV	O	12
Afd. OT-PV	OT-PV	39
Afd. OT-PV	PV	8
Afd. W	W	20
Afd. Z	Z Uit	110
Afd. Z	Z In	39
Afd. ZG-ZM	ZG	27
Afd. ZG-ZM	ZM Afvoer	16.9
Baafjespolder	Baafjespolder	55
Bergermeer	Defensiegemaal	50
Bergermeer	Bergermeer	86

Polder	NAAM	MAXIMALECA
Boekelermeer	Boekel Boekelermeerpolder	38
Callantsoog	Koetensluis	45
Callantsoog	Rechtendijk	45
Damlanderpolder	Damlander	21
De Kaag	Kaagpolder Opmeer	40
Egmondermeer	Egmondermeer	80
Groeterpolder	Groeterpolder	24
Grootdammerpolder	Grootdammer	33
Hargerpolder	Hargerpolder	38.4
Hensbroek	Hensbroek	60
Huisduinen	Huisduinen	11
Koegras	Callantsoogervaart	20
Koegras	Kooypunt	10
Lage Hoek	De Lage Hoek	47.4
Leipolder	Leipolder	9.6
Obdam	Obdam	100
Oosterzijpolder	De Leije, Heiloo	12
Oosterzijpolder	Boekel Oosterzijpolder	86
Philisteinsepolder	Philisteinsche molen	20
Polder de Berkmeer	Berkmeer	30
Polder de Woudmeer	Woudmeer	40
Polder Schagerwaard	Schagerwaard	76.2
Sammerspolder	Sammerspolder	58
Slootgaardpolder	Slootgaard	76
Speketerspolder	Speketer	40
’t Hoekje	t Hoekje	48
’t Hoekje	Burger	4
Ursem	Ursem	119
Valkkoog	Valkkoog	50
Vennewaterspolder	Vennewaterspolder	32
Verenigde Polders	De Rekere	88
Wimmenummerpolder	Wimmenummer	8
Wogmeer	Wogmeer Boven	65

Table C.2: Final adjusted polder removal capacities.

Polder	Max. Capacity [m3/min]	Area [ha]	Removal rate [mm/day]
’t Hoekje	52	388	19.3
Aagtdorperpolder	25	284	12.7
Afd. AB	65	543	17.2
Afd. C	30.8	316	14.0
Afd. D	19	56	48.9
Afd. E	52	563	13.3
Afd. F	19	138	19.8
Afd. H-ON	71	498	20.5
Afd. I-noord	27	202	19.2
Afd. I-zuid	8	69	16.7
Afd. KP	37	356	15.0
Afd. LQ	32	299	15.4
Afd. NG	27	215	18.1
Afd. NMR	120.1	692	25.0
Afd. NS	24	208	16.6
Afd. OT-PV	59	586	14.5
Afd. W	20	159	18.1
Afd. Z	149	791	27.1
Afd. ZG-ZM	43.9	381	16.6
Baafjespolder	55	461	17.2
Bergermeer	136	846	23.1

Polder	Max. Capacity [m3/min]	Area [ha]	Removal rate [mm/day]
Boekelermeer	38	334	16.4
Callantsoog	90	740	17.5
Damlanderpolder	21	282	10.7
De Kaag	40	409	14.1
Egmondermeer	80	714	16.1
Groeterpolder	24	301	11.5
Grootdammerpolder	33	461	10.3
Hargerpolder	38.4	361	15.3
Hensbroek	60	567	15.2
Lage Hoek	47.4	327	20.9
Leipolder	9.6	94	14.7
Obdam	51.3	336	22.0
Oosterzijpolder	98	1127	12.5
Philisteinsepolder	20	285	10.1
Polder Schagerwaard	76.2	659	16.7
Polder Valkkoog	50	512	14.1
Polder de Berkmeer	30	287	15.1
Polder de Woudmeer	40	327	17.6
Ringpolder	142.6	1426	14.4
Sammerspolder	58	451	18.5
Slootgaardpolder	76	570	19.2
Speketerspolder	40	405	14.2
Ursem	119	1065	16.1
Vennewaterspolder	32	338	13.6
Verenigde Polders	88	916	13.8
Wimmenummerpolder	8	115	10.0
Wogmeer	65	691	13.5

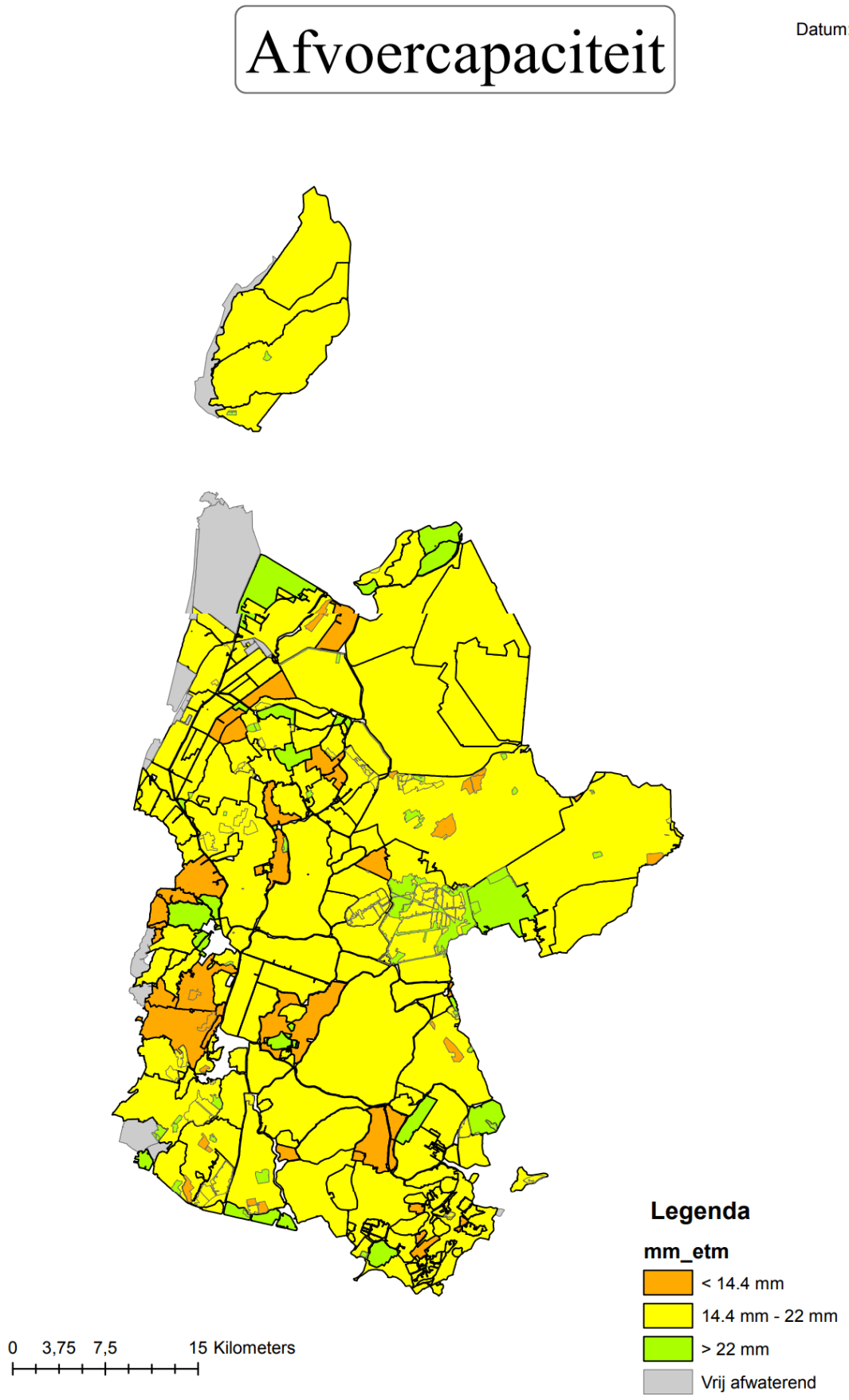


Figure C.1: Discharge capacity for each polder of HHNK.

D

Optimization methods

The linear programming in continuous variables consists in optimizing a criterion, otherwise called objective function, calculated from some of the variables using a formula, while assuring that constraints on the variables are met. In LP, constraints are linear and the program is solved through the Simplex algorithm [22].

The placement of pumps is classified as a Mixed Integer Programming problem (MIP). Such a problem contains both integer and continuous variables. If the objective function is expressed with only linear forms, the problem is termed a Mixed Integer Linear Programming (MILP) [42]. MIP or MILP problems are not solved directly since integer constraints make the feasible region difficult to analyze.

Integer (linear) programming is when the decision space should be expressed in integer variables instead of variables in R+. These problems do not admit polynomial algorithms to solve them. It is necessary to propose alternative solutions to the simplex algorithm, which rely on traversals in a tree of solutions, such that at each node of the tree, some routines are executed. This is called branching. A combination of both techniques is called mixed-integer programming, in such a program, integer variable constraints are 'relaxed' so that they become continuous within certain bounds [22].

Branch-and-Bound is a method used to solve optimization problems by systematically breaking the problem into smaller sub-problems. It consists of systematic enumeration of candidate solutions by means of state space search. The method avoids evaluating all candidate solutions by discarding sub-problems that cannot contain the optimal solution.

1. **Branching**: The problem is divided into smaller sub-problems or "branches" based on the decision variables. This forms a tree structure, where the root node represents the entire problem (top node) and each branch represents a partial solution or a restriction on decision variables. The process of creating sub-problems from a node is called branching. Leaf nodes are the bottommost nodes, where no more branching can occur. These nodes are where you evaluate the solution quality and check whether it is better than the current solution.
2. **Bounding**: For each sub-problem (node in the tree), a bound on the best possible solution in that branch is computed. If the bound is worse than the best solution found so far (known as the "incumbent"), the entire branch is discarded or pruned, as it cannot lead to a better solution. Upper bound for minimization problem.
3. **Pruning**: Branches that cannot improve on the current best solution (incumbent) are pruned. This reduces the number of candidate solutions that need to be evaluated. The algorithm stores the best solution found at each step, and only explores branches that have the potential to contain a better solution than the current best.

Effectively, the technique recursively splits the search space into smaller spaces and minimizes f(x) on those spaces. It keeps track of the bounds on the minimum and 'prunes' the search space to eliminate candidate solutions. Branching methods create a tree during solving. It consists of dividing the feasible set of a problem into subsets, where each node represents a subproblem that only searches the subset at that node. The process of creating sub problems from a node is called 'branching'. For each subproblem, a lower bound (for a minimization problem) is computed. The lower bound is the best possible outcome within that subproblem and may correspond to a non-feasible solution (such

as having fractional variables). The lower bound is calculated with the Simplex algorithm. Within this branch there will not be an integer solution with an higher value than this lower bound. The algorithm also keeps track of the best feasible solution, which is called the incumbent. The algorithm compares the lower bound of the current branch to the incumbent solution's value, if the lower bound is worse than the incumbent, it 'prunes' the tree, discarding the branch. This is done because no feasible solution within that subproblem can be better than the current incumbent [8].

The branch-and-cut method combines the branch-and-bound and the cutting plane method. With the cutting plane method, additional constraints (cuts) are implemented to the relaxed problem to eliminate fractional solutions of the solution space, iteratively converging to a feasible integer solution [42].

NP-hard stands for Non-deterministic Polynomial-time hard. For NP-hard problems, the time required to find an optimal solution increases exponentially with the size of the problem. This makes finding exact solutions intractable for large problems, as even computers would take an impractical amount of time to solve them. Relaxation is a technique used to make NP-hard problems easier to solve by simplifying certain constraints. Specifically, relaxation involves loosening or relaxing some of the strict requirements of the problem so that it becomes easier to solve (often turning it into a polynomial-time problem). One common relaxation technique is to remove the integer constraint, allowing the variables to take on any real (continuous) value between 0 and 1 (for example, $0X_i1$ instead of $X_i0, 1$). This transforms the original problem into a linear programming (LP) problem, which is solvable in polynomial time.

MIP Branch and Cut Branch-and-Cut is a method to solve ILP or MILP problems, it combines the branch-and-bound and cutting planes techniques. Since the technique is exact, it guarantees optimality. By relaxing the problem, the MIP becomes a linear program (LP), which can be solved more efficiently using methods like the Simplex algorithm (more common) or Interior Point methods. In every Branch-and-Bound step, after solving the relaxed LP, if the solution involves fractional values a cutting plane algorithm is used. Cutting planes are additional linear constraints added to the LP relaxation to reduce the feasible region and eliminate parts that don't contain feasible integer solutions. The method identifies regions of the feasible set that violate the integer constraint. It adds a cut (a new constraint) to exclude that fractional solution while preserving the integer solution. This process is repeated iteratively, (progressively) tightening the LP relaxation until an integer solution is found or the relaxation cannot be further improved. Advantages: Fewer branches need to be explored in compared to pure branch-and-bound. The method is effective in solving large-scale integer and mixed-integer problems because it significantly reduces the search space through cuts.

E

Code for the Creation of Polder VDCs

E.1. Functions for the Creation of the Master CSV

```
import geopandas as gpd
import numpy as np
import pandas as pd
import os
from osgeo import gdal
from shapely.geometry import Point

# Base directory containing subregions
output_folder = "C:\\Users\\Pchva\\Documents\\ENVM4000\\3_Coding\\05_Lizard\\csvs"
base_folder = "C:\\Users\\Pchva\\Documents\\ENVM4000\\3_Coding\\05_Lizard\\subregions"

# Set the target resolution for resampling
target_resolution = 5  # in meters

# Function to merge raster tiles
def merge_rasters(raster_files):
    src_ds = gdal.Open(raster_files[0])
    dst_ds = gdal.Warp('/vsimem/merged.tif', raster_files, format='GTiff', dstNodata=-999)
    return dst_ds

# Function to resample AHN raster to the target resolution
def resample_ahn_raster(ahn_ds):
    # Get the original transform and size
    original_transform = ahn_ds.GetGeoTransform()
    original_projection = ahn_ds.GetProjection()

    # Calculate the new dimensions
    width = int((ahn_ds.RasterXSize * original_transform[1]) / target_resolution)
    height = int((ahn_ds.RasterYSize * -original_transform[5]) / target_resolution)

    # Create the target raster
    target_ds = gdal.GetDriverByName('GTiff').Create('/vsimem/resampled_ahn.tif', width,
        height, 1, gdal.GDT_Float32)

    # Set the new transform
    new_transform = (original_transform[0], target_resolution, 0, original_transform[3], 0, -
        target_resolution)
    target_ds.SetGeoTransform(new_transform)
    target_ds.SetProjection(original_projection)

    # Set the nodata value for the output band
    target_band = target_ds.GetRasterBand(1)
    target_band.SetNoDataValue(-999)

    # Resample the data
    gdal.ReprojectImage(ahn_ds, target_ds, original_projection, original_projection, gdal.
        GRA_Average)

    return target_ds

# Sample data from AHN and Landuse
def sample_raster_data(ahn_ds, landuse_ds):
```

```
    sampled_data = []

    # Get the transform of the AHN dataset
    original_transform = ahn_ds.GetGeoTransform()
    nodata_value = ahn_ds.GetRasterBand(1).GetNoDataValue()

    # Loop through each pixel in the AHN dataset
    for row in range(ahn_ds.RasterYSize):
        for col in range(ahn_ds.RasterXSize):
            ahn_value = ahn_ds.ReadAsArray(col, row, 1, 1)[0, 0]

            # Check if the AHN value is valid
            if ahn_value == nodata_value:  # Assuming 0 means no data or invalid
                continue

            # Get coordinates of AHN pixel
            x_geo = original_transform[0] + col * original_transform[1]
            y_geo = original_transform[3] + row * original_transform[5]

            # Get corresponding landuse value (sampling from landuse dataset)
            landuse_value = landuse_ds.ReadAsArray(
                int((x_geo - landuse_ds.GetGeoTransform()[0]) / landuse_ds.GetGeoTransform()[1]),
                int((y_geo - landuse_ds.GetGeoTransform()[3]) / landuse_ds.GetGeoTransform()[5]),
                1, 1)[0, 0]

            sampled_data.append((x_geo, y_geo, ahn_value, landuse_value))

    return sampled_data

def assign_polder_name(sampled_data_df, shapefile_path):
    # Load the polders shapefile using geopandas
    polders_gdf = gpd.read_file(shapefile_path)

    # Check and ensure both the sampled data and polders are in the same CRS
    sampled_gdf = gpd.GeoDataFrame(sampled_data_df,
                                   geometry=[Point(xy) for xy in zip(sampled_data_df['X'], sampled_data_df['Y'])],
                                   crs=polders_gdf.crs)  # Reproject to match the CRS of polders_gdf

    # Perform a spatial join (using 'intersects' to account for points on polygon boundaries)
    joined_gdf = gpd.sjoin(sampled_gdf, polders_gdf[['NAAM', 'geometry']], how='left', predicate='intersects')

    # Add the polder name to the original DataFrame
    sampled_data_df['Polder'] = joined_gdf['NAAM']

    # Check for missing data (in case some points are outside all polders)
    if sampled_data_df['Polder'].isnull().any():
        print("Warning: Some points were not assigned a polder name (they may be outside the polders).")

    return sampled_data_df

def assign_water_levels(sampled_data_df, peilvakken_path):
    # Laad de peilvakken shapefile met geopandas
    peilvakken_gdf = gpd.read_file(peilvakken_path)

    # Zorg ervoor dat de sampled data en peilvakken dezelfde CRS hebben
    sampled_gdf = gpd.GeoDataFrame(
        sampled_data_df,
        geometry=[Point(xy) for xy in zip(sampled_data_df['X'], sampled_data_df['Y'])],
        crs=peilvakken_gdf.crs
    )

    # Reset index om duplicaten in de index te vermijden
    sampled_gdf = sampled_gdf.reset_index(drop=True)

    # Voer een ruimtelijke join uit om winterpeil en zomerpeil toe te wijzen
    joined_gdf = gpd.sjoin(sampled_gdf, peilvakken_gdf[['winterpe_1', 'zomerpei_1', 'geometry']], how='left', predicate='intersects')

    # Verwijder eventuele duplicaten in de spatial join
```

```
    joined_gdf = joined_gdf[~joined_gdf.index.duplicated(keep='first')]

    # Reset de index van joined_gdf om verdere conflicten te voorkomen
    joined_gdf = joined_gdf.reset_index(drop=True)

    # Voeg de winterpeil- en zomerpeilwaarden toe aan de oorspronkelijke DataFrame
    sampled_data_df['Winterpeil'] = joined_gdf['winterpe_1']
    sampled_data_df['Zomerpeil'] = joined_gdf['zomerpei_1']

    # Controleer of er ontbrekende data is (punten buiten de peilvakken)
    if sampled_data_df[['Winterpeil', 'Zomerpeil']].isnull().any().any():
        print("Warning: Some points were not assigned water levels (they may be outside the peilvakken).")

    return sampled_data_df

# Process each subregion
def process_subregion(subregion_folder, output_folder, shapefile_path, peilvakken_path):
    ahn_folder = os.path.join(subregion_folder, 'ahn3_tiles')
    landuse_folder = os.path.join(subregion_folder, 'landuse2019_tiles')

    ahn_tiles = [os.path.join(ahn_folder, file) for file in os.listdir(ahn_folder) if file.endswith('.tif')]
    landuse_tiles = [os.path.join(landuse_folder, file) for file in os.listdir(landuse_folder) if file.endswith('.tif')]

    # Merge the AHN raster tiles
    ahn_ds = merge_rasters(ahn_tiles)

    # Resample the AHN raster to 5x5 meters
    ahn_resampled_ds = resample_ahn_raster(ahn_ds)

    # Merge Landuse tiles
    landuse_ds = merge_rasters(landuse_tiles)

    # Sample AHN and Landuse data
    sampled_data = sample_raster_data(ahn_resampled_ds, landuse_ds)

    # Convert sampled data to a DataFrame
    df = pd.DataFrame(sampled_data, columns=['X', 'Y', 'AHN', 'Landuse'])

    # Assign the polder name to each point
    df = assign_polder_name(df, shapefile_path)

    # Assign water levels (winterpeil and zomerpeil) to each point
    df = assign_water_levels(df, peilvakken_path)

    # Save the sampled data to CSV
    subregion_name = os.path.basename(subregion_folder)
    output_path = os.path.join(output_folder, f'{subregion_name}_sampled_data.csv')
    df.to_csv(output_path, index=False)

    print(f'Saved: {output_path}')
```

E.2. Create Master CSVs

```
# List of subregions to process
subregions = ['01texel', '02wmr', '03wf', '04kop', '05schermer', '06ad', '07waterland', '08beemster']

# Base directory containing subregions
output_folder = "C:\\Users\\Pchva\\Documents\\ENVM4000\\3_Coding\\05_Lizard\\csvs"
base_folder = "C:\\Users\\Pchva\\Documents\\ENVM4000\\3_Coding\\05_Lizard\\subregions"

# Shapefile containing polders
# shapefile_path = "polders_indeling.shp"
shapefile_path = "polders_selectie_edit.shp"
peilvakken_path = "peilvakken_streefpeilen.shp"

import time

# Process each subregion
for subregion in subregions:
    start_time = time.time()
```

```
    print(start_time)

    subregion_folder = os.path.join(base_folder, subregion)
    process_subregion(subregion_folder, output_folder, shapefile_path, peilvakken_path)

    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f'Processing time for {subregion}: {elapsed_time:.2f} seconds')
```

E.3. Splitting the Master CSVs into polder CSVs

```
import pandas as pd
import os

# Define the file paths
csv_path = 'csvs/00_beemster_schermer_kop_wf_sampled_data.csv'  # Replace with the path to your CSV
output_folder = 'csvs_polders_combined'   # Replace with the path to where CSVs should be saved

# Load the CSV
df = pd.read_csv(csv_path)

# Get rows with no polder name (NaN or empty)
unassigned_points = df[df['Polder'].isna() | (df['Polder'] == '')]

# Drop rows where 'Polder' is NaN or empty for valid export
df = df[df['Polder'].notna() & (df['Polder'] != '')]

# Get unique polder values
polders = df['Polder'].unique()

# Ensure the output folder exists
os.makedirs(output_folder, exist_ok=True)

# Export each polder's data as a separate CSV file
for polder in polders:
    # Create a valid file name by replacing unwanted characters
    valid_polder_name = polder.replace(" ", "_").replace("'", "").replace("-", "_")

    # Filter for the current polder
    polder_df = df[df['Polder'] == polder]

    # Define output path for the CSV
    csv_path = os.path.join(output_folder, f'points_{valid_polder_name}.csv')

    # Save to CSV
    polder_df.to_csv(csv_path, index=False)

    print(f"Saved {csv_path}")

print('hello')
# If there are unassigned points, save them to a separate CSV
if not unassigned_points.empty:
    unassigned_csv_path = os.path.join(output_folder, 'unassigned_points_00combined.csv')
    unassigned_points.to_csv(unassigned_csv_path, index=False)
    print(f"Saved {unassigned_csv_path}")

print("All valid polders have been exported as CSV files.")
```

E.4. Merge Polder CSVs with WSS Configuration File Into Dataframe

```
import configparser
from collections import defaultdict
import geopandas as gpd
import numpy as np
import pandas as pd
import os
from osgeo import gdal
import matplotlib.pyplot as plt

# Create a ConfigParser object
config = configparser.ConfigParser()
```

```
12
13 # Read the .cfg file
14 config.read('WSS_tabel.cfg')
15
16 data = []
17
18 # Process configuration file
19 for section in config.sections():
20     if 'omschrijving' in config[section]:
21         # Initialize row dictionary
22         row = {'omschrijving': config[section]['omschrijving']}
23
24         # Collect direct and indirect values
25         row['Landuse'] = float(section)
26         row['direct_eenheid'] = config[section]['direct_eenheid']
27         row['direct_gem'] = float(config[section]['direct_gem'])
28         row['direct_min'] = float(config[section]['direct_min'])
29         row['direct_max'] = float(config[section]['direct_max'])
30         row['indirect_eenheid'] = config[section]['indirect_eenheid']
31         row['indirect_gem'] = float(config[section]['indirect_gem'])
32         row['indirect_min'] = float(config[section]['indirect_min'])
33         row['indirect_max'] = float(config[section]['indirect_max'])
34
35         # Collect gamma values for months
36         gamma_maand = eval(config[section]['gamma_maand'])
37         months = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'okt', 'nov'
               , 'dec']
38         for i, month in enumerate(months):
39             row[f'gamma_{month}'] = gamma_maand[i]
40
41
42         # Collect gamma values for inundatieduur
43         gamma_inundatieduur = eval(config[section]['gamma_inundatieduur'])
44         row.update({
45             'gamma_inundatieduur_1': gamma_inundatieduur[0],
46             'gamma_inundatieduur_12': gamma_inundatieduur[1],
47             'gamma_inundatieduur_24': gamma_inundatieduur[2],
48             'gamma_inundatieduur_72': gamma_inundatieduur[3],
49             'gamma_inundatieduur_480': gamma_inundatieduur[4],
50         })
51
52         # Collect gamma values for herstelperiode
53         gamma_herstelperiode = eval(config[section]['gamma_herstelperiode'])
54         row.update({
55             'gamma_herstelperiode_0': gamma_herstelperiode[0],
56             'gamma_herstelperiode_6': gamma_herstelperiode[1],
57             'gamma_herstelperiode_24': gamma_herstelperiode[2],
58             'gamma_herstelperiode_48': gamma_herstelperiode[3],
59             'gamma_herstelperiode_120': gamma_herstelperiode[4],
60             'gamma_herstelperiode_240': gamma_herstelperiode[5],
61         })
62
63         # Collect gamma values for inundatiediepte
64         gamma_inundatiediepte = eval(config[section]['gamma_inundatiediepte'])
65         row.update({
66             'gamma_inundatiediepte_000': gamma_inundatiediepte[0],
67             'gamma_inundatiediepte_001': gamma_inundatiediepte[1],
68             'gamma_inundatiediepte_005': gamma_inundatiediepte[2],
69             'gamma_inundatiediepte_015': gamma_inundatiediepte[3],
70             'gamma_inundatiediepte_030': gamma_inundatiediepte[4],
71         })
72
73         # Append the row to the data list
74         data.append(row)
75
76 # Convert the list of dictionaries to a DataFrame
77 df_config = pd.DataFrame(data)
78
79 ########### Waarden en eenheden directe schade in zelfde eenheid zetten
80 ha_mask = df_config['direct_eenheid'] == '/ha'
81 df_config.loc[ha_mask, 'direct_gem'] /= 10000
82 df_config.loc[ha_mask, 'direct_min'] /= 10000
83 df_config.loc[ha_mask, 'direct_max'] /= 10000
84 df_config.loc[ha_mask, 'direct_eenheid'] = '/m2'
85 df_config.loc[ha_mask, 'omschrijving'] = df_config.loc[ha_mask, 'omschrijving'].str.replace('
```

```
    /ha', '/m²')

########## Indirecte schade snel-, regionale-, lokale- en spoorwegen verwijderen (was per wegvak)
ha_mask = df_config['indirect_eenheid'] == '/wegvak/dag'
df_config.loc[ha_mask, 'indirect_gem'] = 0
df_config.loc[ha_mask, 'indirect_min'] = 0
df_config.loc[ha_mask, 'indirect_max'] = 0

df_config.head(10)
```

E.5. Functions for the Damage Calculations

```
def vectorized_interpolation(depth, f_x_pts, f_y_pts_matrix):
    n_cells = depth.shape[0]
    n_pts = len(f_x_pts)

    # Find indices where depth would be inserted
    indices = np.searchsorted(f_x_pts, depth) - 1
    indices = np.clip(indices, 0, n_pts - 2)

    # Get x0, x1
    x0 = np.take(f_x_pts, indices)
    x1 = np.take(f_x_pts, indices + 1)

    # Get y0, y1
    y0 = f_y_pts_matrix[np.arange(n_cells), indices]
    y1 = f_y_pts_matrix[np.arange(n_cells), indices + 1]

    # Compute slopes and interpolate
    slope = (y1 - y0) / (x1 - x0)
    factor_depth = y0 + slope * (depth - x0)

    # Handle depths outside the interpolation range
    factor_depth = np.where(
        depth <= f_x_pts[0],
        f_y_pts_matrix[:, 0],
        np.where(
            depth >= f_x_pts[-1],
            f_y_pts_matrix[:, -1],
            factor_depth
        )
    )

    return factor_depth

def calculate_damage(df_merged):
    from scipy.interpolate import interp1d
    # Land use categories (as before)
    landuse_categories = {
    'no_damage': [13, 14, 24, 30, 32, 33, 34, 35, 39, 40, 41, 42, 43, 44, 45, 50, 51, 52, 73,
         115, 156, 254],
    'infrastructure': [25, 26, 27, 28, 29, 31, 165, 166],
    'field_crops': [108, 109, 111, 157,
                    86, 87, 91, 105, 132, 146, 148,
                    112,
                    145, 154,
                    55, 56, 57, 59, 60, 62, 63, 64, 68, 69, 72, 74, 75, 77, 78, 80, 83, 90,
                        92, 93, 94, 98, 99, 100, 101, 102, 103, 104, 106, 110, 113, 117, 118,
                         120, 121, 123, 124, 126, 127, 129, 131, 134, 135, 138, 139, 140,
                        141, 142, 144,
                    163, 79, 122],
    'horticulture': [125, 130, 158, 164, 61, 65, 81, 84, 85, 95, 96, 116, 119,
                     66, 67, 114, 133, 152,
                     70, 71, 76, 97, 107],
    'nature_recreation': [15, 16, 19, 22,
                          17, 18, 20, 21],
    'greenhouses': [7],
    'buildings': [3, 10, 2, 4, 5, 6, 8, 9, 11, 12]
}

    # Convert mNAP and other relevant columns to arrays
    mNAP = np.array(df_merged['AHN'])
    landuse = np.array(df_merged['Landuse'])
```

```
58
59      # Extract water levels
60      winterpeil = np.array(df_merged['Winterpeil'].fillna(-9999))
61      zomerpeil = np.array(df_merged['Zomerpeil'].fillna(-9999))
62
63      # Identify valid water levels
64      valid_winter_mask = (winterpeil > -20) & (~np.isnan(winterpeil))
65      valid_zomer_mask = (zomerpeil > -20) & (~np.isnan(zomerpeil))
66
67      # Initialize depth arrays
68      depth_winter = np.zeros_like(mNAP)
69      depth_zomer = np.zeros_like(mNAP)
70
71      # Calculate depths where valid
72      depth_winter[valid_winter_mask] = np.maximum(winterpeil[valid_winter_mask] - mNAP[
            valid_winter_mask], 0)
73      depth_zomer[valid_zomer_mask] = np.maximum(zomerpeil[valid_zomer_mask] - mNAP[
            valid_zomer_mask], 0)
74
75      # Calculate initial volumes
76      cell_area = 25  # Area per cell (m²)
77      init_vol_winter = np.sum(depth_winter * cell_area)
78      init_vol_zomer = np.sum(depth_zomer * cell_area)
79
80      # Define water levels for damage calculation
81      min_winterpeil = np.min(winterpeil[valid_winter_mask])
82      test_wlvl = np.linspace(min_winterpeil, min_winterpeil + 2, 100)
83
84      # Extract variables from df_merged
85      y_max_min = np.array(df_merged['direct_min'])
86      y_max_gem = np.array(df_merged['direct_gem'])
87      y_max_max = np.array(df_merged['direct_max'])
88
89
90      y_1_u = np.array(df_merged['gamma_inundatieduur_1'])
91      y_12_u = np.array(df_merged['gamma_inundatieduur_12'])
92      y_24_u = np.array(df_merged['gamma_inundatieduur_24'])
93      y_72_u = np.array(df_merged['gamma_inundatieduur_72'])
94      y_480_u = np.array(df_merged['gamma_inundatieduur_480'])
95
96      y_herstel_24 = np.array(df_merged['gamma_herstelperiode_24'])
97      y_herstel_48 = np.array(df_merged['gamma_herstelperiode_48'])
98      y_herstel_120 = np.array(df_merged['gamma_herstelperiode_120'])
99
100     y_max_ind = np.array(df_merged['indirect_gem'])  # Corrected variable name
101
102     y_jun = np.array(df_merged['gamma_jun'])
103
104     # Depth factors for inundation depth
105     y_wlvl_000 = np.array(df_merged['gamma_inundatiediepte_000'])
106     y_wlvl_001 = np.array(df_merged['gamma_inundatiediepte_001'])
107     y_wlvl_005 = np.array(df_merged['gamma_inundatiediepte_005'])
108     y_wlvl_015 = np.array(df_merged['gamma_inundatiediepte_015'])
109     y_wlvl_030 = np.array(df_merged['gamma_inundatiediepte_030'])
110
111     # Prepare dictionaries for combinations
112     damage_types = {'min': y_max_min, 'gem': y_max_gem, 'max': y_max_max}
113     durations = {'1u': y_1_u, '12u': y_12_u, '24u': y_24_u, '72u': y_72_u, '240u': y_480_u}
114     herstelperiodes = {'24h': y_herstel_24, '48h': y_herstel_48, '120h': y_herstel_120}
115
116     # Initialize the damage dictionary
117     damage = {}
118
119     # Prepare output lists
120     cum_vol_wlvl = []
121     cum_area_wlvl = []
122
123     som_area = len(mNAP) * cell_area  # Total area
124
125     f_x_pts = [0, 0.01, 0.05, 0.15, 0.3, 10]  # Depth points for interpolation
126     f_y_pts = np.vstack([y_wlvl_000, y_wlvl_001, y_wlvl_005, y_wlvl_015, y_wlvl_030, np.
            ones_like(y_wlvl_000)]).T #actual factors for all cells
127
128     landuse_percentages_wlvl = {wlvl: {cat: 0 for cat in landuse_categories} for wlvl in
            test_wlvl}
```

```
129
130     # Iterate over each water level
131     for wlvl_idx, wlvl in enumerate(test_wlvl):
132         depth = np.maximum(wlvl - mNAP, 0)
133         area_cells = np.where(depth > 0, cell_area, 0)
134
135         factor_depth = vectorized_interpolation(depth, f_x_pts, f_y_pts)
136
137         # Volume and area calculations
138         volume_cells = depth * cell_area
139         area_cells = np.where(depth > 0, cell_area, 0)
140
141         total_vol = np.sum(volume_cells)
142         total_area = np.sum(area_cells)
143
144         # Append to cumulative lists
145         cum_vol_wlvl.append(total_vol)
146         cum_area_wlvl.append(total_area)
147
148         # Categorize flooded area by land use
149         for category, landuse_codes in landuse_categories.items():
150             mask1 = np.isin(landuse, landuse_codes)  # Cells belonging to this category
151             flooded_area = np.sum(area_cells[mask1])
152             total_area_category = np.sum(cell_area * mask1)  # Total area of this category
153             percentage = (flooded_area / total_area_category * 100) if total_area_category >
                    0 else 0
154             landuse_percentages_wlvl[wlvl][category] = percentage
155
156         # Iterate over combinations
157         for damage_type_name, y_max_array in damage_types.items():
158             for duration_name, duration_array in durations.items():
159                 for herstel_name, herstel_array in herstelperiodes.items():
160                     combination_name = f"jun_{damage_type_name}_{duration_name}_{herstel_name
                            }"
161
162                     # Initialize arrays to store damage per cell
163                     damage_direct = y_max_array * factor_depth * duration_array * y_jun *
                            cell_area
164
165                     # Calculate indirect damage per cell
166                     damage_indirect = np.where(damage_direct > 0, y_max_ind * herstel_array *
                             cell_area, 0)
167
168                     # Total damage per cell
169                     total_damage_cells = damage_direct + damage_indirect
170
171                     # Initialize damage per category
172                     damage_by_category_step = {cat: 0 for cat in landuse_categories}
173                     damage_by_category_step['total'] = np.sum(total_damage_cells)
174
175                     # Categorize damage by land use
176                     for category, landuse_codes in landuse_categories.items():
177                         mask = np.isin(landuse, landuse_codes)
178                         damage_by_category_step[category] = np.sum(total_damage_cells[mask])
179
180                     # Store damage for this combination and water level
181                     if combination_name not in damage:
182                         # Initialize lists for each category
183                         damage[combination_name] = {cat: [] for cat in
                                damage_by_category_step}
184                     for cat in damage_by_category_step:
185                         damage[combination_name][cat].append(damage_by_category_step[cat])
186
187     return cum_vol_wlvl, cum_area_wlvl, test_wlvl, som_area, init_vol_winter, init_vol_zomer,
            damage, landuse_percentages_wlvl
188
189 import time
190
191 # Initialize an empty dictionary to store damage results if not already done
192 damage_dict = {}
193
194 # folder_path = "C:\\Users\\Pchva\\Documents\\ENVM4000\\3_Coding\\05_Lizard\\
        csvs_polders_combined"
195 folder_path = "C:\\Users\\Pchva\\Documents\\ENVM4000\\3_Coding\\05_Lizard\\csvs_polders"
196
```

```
# Create a list to store polder names
polder_names = []

# Define land use categories for percentages (should match calculate_damage)
landuse_categories = {
    'no_damage': [13, 14, 24, 30, 32, 33, 34, 35, 39, 40, 41, 42, 43, 44, 45, 50, 51, 52, 73,
         115, 156, 254],
    'infrastructure': [25, 26, 27, 28, 29, 31, 165, 166],
    'field_crops': [108, 109, 111, 157,
                    86, 87, 91, 105, 132, 146, 148,
                    112,
                    145, 154,
                    55, 56, 57, 59, 60, 62, 63, 64, 68, 69, 72, 74, 75, 77, 78, 80, 83, 90,
                        92, 93, 94, 98, 99, 100, 101, 102, 103, 104, 106, 110, 113, 117, 118,
                         120, 121, 123, 124, 126, 127, 129, 131, 134, 135, 138, 139, 140,
                        141, 142, 144,
                    163, 79, 122],
    'horticulture': [125, 130, 158, 164, 61, 65, 81, 84, 85, 95, 96, 116, 119,
                     66, 67, 114, 133, 152,
                     70, 71, 76, 97, 107],
    'nature_recreation': [15, 16, 19, 22,
                          17, 18, 20, 21],
    'greenhouses': [7],
    'buildings': [3, 10, 2, 4, 5, 6, 8, 9, 11, 12]
}

def calculate_landuse_percentages(df, landuse_categories):
    """
    Calculate the percentage of each land use category.
    """
    total_count = len(df)
    percentages = {}
    for category, codes in landuse_categories.items():
        category_count = df['Landuse'].isin(codes).sum()
        percentages[f'{category}_percentage'] = (category_count / total_count) * 100 if
            total_count > 0 else 0
    return percentages
```

E.6. Calling the Damage Functions for Individual Polders

```

counter = 0

for iteration, file_name in enumerate(os.listdir(folder_path)):
    if file_name.startswith('points_') and file_name.endswith('.csv'):
        file_path = os.path.join(folder_path, file_name)
        df = pd.read_csv(file_path)

        counter += 1

        # Data cleaning and processing
        df['Landuse'] = df['Landuse'].fillna(254)
        df['Landuse'] = df['Landuse'].replace(0, 254)
        df['Landuse'] = df['Landuse'].replace(253, 254)

        # Merging df_config with dataframe
        df_merged = pd.merge(df, df_config, on='Landuse', how='left')

        df_merged.sort_values(by='AHN', inplace=True)
        df_merged = df_merged.reset_index(drop=True)

        # Extract the text using slicing
        polder_name = file_name[7:-len('.csv')]  # Get text from index 7 to the end minus the
             length of suffix
        print(f"Iteration {iteration}: Polder Name: {polder_name}")

        # Store the polder name in a list for later use
        polder_names.append(polder_name)

        start_time = time.time()

        # # Calculate damage, volume, and area for the polder
        vol_df_merged, area_df_merged, wlvl, tot_area, init_vol_winter, init_vol_zomer,
            damage, landuse_percentages_wlvl = calculate_damage(df_merged)
```

```
        landuse_percentages = calculate_landuse_percentages(df_merged, landuse_categories)

        # Store the damage, volume, area, and landuse percentages in the dictionary
        damage_dict[polder_name] = {
            'volume': vol_df_merged,      # Volume for each test_wlvl
            'area': area_df_merged,       # Flooded area for each test_wlvl
            'wlvl': wlvl,
            'total_area': tot_area,
            'init_vol_winter': init_vol_winter,
            'init_vol_zomer': init_vol_zomer,
            'landuse_percentages': landuse_percentages,
            'landuse_percentages_wlvl': landuse_percentages_wlvl,  # Percentages per wlvl
            'damage': damage  # Add the damage data here
        }

        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f'Processing time: {(elapsed_time / 60):.1f} minutes')
```

E.7. Store the Polder Dictionaries

```
# Define the folder path for saving .npy files
folder_path = "C:\\Users\\Pchva\\Documents\\ENVM4000\\3_Coding\\04_dicts_new"

# Loop through each polder in the damage_dict
for name, data in damage_dict.items():

    # Define the file path for saving the modified data
    file_path = os.path.join(folder_path, f"{name}.npy")

    # Save the modified dictionary to an .npy file
    np.save(file_path, data)

    # Optional: print to confirm each file saved
    print(f"Saved {name} data to {file_path}")
```

F

First-Stage code

```
import gurobipy as gp
from gurobipy import GRB
import numpy as np

# Initialize the optimization model
model = gp.Model("PolderOptimization_FirstStage_Aggregated")

num_timesteps = len(polders_dictionary['Aagtdorperpolder']['precipitation_3hr'])
num_polders = len(polders_dictionary)
total_pumps = 20  # total number of pumps

# Suppose we use uniform pump capacity for first stage
avg_pump_capacity_m3min = 24.25  # from previous calculation
pump_capacity_m3h = avg_pump_capacity_m3min * 60 * 3

# Decision variable: pump_count[t,p] integer
pump_count_1st = model.addVars(num_timesteps, num_polders, vtype=GRB.INTEGER, lb=0, ub=5,
    name="pump_count")
infiltration_1st = model.addVars(num_timesteps, num_polders, vtype=GRB.CONTINUOUS, lb=0, name
    ="infiltration_1st")

water_volume_optimized_1st = model.addVars(num_timesteps, num_polders, lb=0, name="
    water_volume_optimized")
max_water_volume_optimized_1st = model.addVars(num_polders, lb=0, name="
    max_water_volume_optimized")
max_polder_damage_optimized_1st = model.addVars(num_polders, name="
    max_polder_damage_optimized")

for p in range(num_polders):
    max_polder_damage_optimized_1st[p].UB = dam_upper_bound[p]
    max_water_volume_optimized_1st[p].UB = vol_upper_bound[p]
    for t in range(num_timesteps):
        water_volume_optimized_1st[t,p].UB = vol_upper_bound[p]
        infiltration_1st[t,p].UB = infiltration_upper_bound[p]

# Constraint: total pumps per timestep must at most 20
for t in range(num_timesteps):
    model.addConstr(gp.quicksum(pump_count_1st[t, p] for p in range(num_polders)) <=
        total_pumps, name=f"total_pumps_{t}")

# Compute total pumping capacity: total_pump_pumping_capacity[t,p] = pump_count[t,p] *
    pump_capacity_m3h
total_pump_pumping_capacity = {
    (t, p): pump_count_1st[t, p] * pump_capacity_m3h
    for t in range(num_timesteps)
    for p in range(num_polders)
}

max_pumps_per_polder = 4  # Example: At most 3 pumps per polder at any timestep

# Constraint: at most max_pumps_per_polder pumps can operate per polder at any timestep
for t in range(num_timesteps):
    for p in range(num_polders):
```

```
        model.addConstr(pump_count_1st[t, p] <= max_pumps_per_polder, name=f"max_pumps_per_polder_{t}_{p}")

net_inflow_array_1st = np.zeros((num_timesteps, num_polders))


# Add constraints for each polder
for p, (polder, data) in enumerate(polders_dictionary.items()):
    sum_inflow = 0
    sum_gemalen = 0
    volume_array = data['volume']
    wlvl_array = data['wlvl']
    damage_array = np.array(data['damage']['jun_gem_72u_120h']['total']) / 1000000
    area_array = data['area']
    area = data['total_area']
    gemalen_capacity = data['gemalen_cap'] * 60 * 3
    precip_gen = data['precipitation_3hr']
    initial_volume = data['init_vol_zomer']

    # Remove duplicates if function defined
    volume_array, damage_array, wlvl_array, area_array = remove_duplicates_across_arrays(volume_array, damage_array, wlvl_array, area_array, area)

    for t in range(num_timesteps):
        net_inflow = precip_gen[t] * area / 1000 - gemalen_capacity
        net_inflow_array_1st[t, p] = net_inflow

        if t == 0:
            model.addConstr(water_volume_optimized_1st[t, p] == initial_volume)
        else:
            model.addConstr(water_volume_optimized_1st[t, p] == water_volume_optimized_1st[t-1, p] + net_inflow -
                            total_pump_pumping_capacity[(t, p)] + infiltration_1st[t,p])

        model.addConstr(max_water_volume_optimized_1st[p] >= water_volume_optimized_1st[t, p])

        sum_inflow += net_inflow
        sum_gemalen += 2*gemalen_capacity

        if sum_inflow > sum_gemalen:
            model.addConstr(infiltration_1st[t,p] == 0, name=f"infiltration_no_overflow_{t}_{p}")
        else:
            model.addConstr(infiltration_1st[t,p] <= gemalen_capacity, name=f"infiltration_{t}_{p}")

    # Damage PWL
    model.addGenConstrPWL(max_water_volume_optimized_1st[p], max_polder_damage_optimized_1st[p], volume_array, damage_array)

# Cost per hour of pump operation (assuming each pump_count[t, p] represents one pump)
cost_per_pump = 0.001
operational_cost = gp.quicksum(pump_count_1st[t, p] * cost_per_pump for t in range(num_timesteps) for p in range(num_polders))
damage_reduction = gp.quicksum(max_polder_damage_optimized_1st[p] for p in range(num_polders))

# Objective: minimize the optimized damage
total_objective = damage_reduction + operational_cost
model.setObjective(total_objective, GRB.MINIMIZE)

# Solver parameters
model.setParam("TimeLimit", 3600)  # 1 hour time limit
# model.setParam("Heuristics", 0.1)  # Focus more on heuristics
model.setParam("MIPFocus", 1)  # Focus on finding feasible solutions
model.setParam("PoolSolutions", 5)  # Collect up to 10 solutions
with open('First_stage.log', "w") as file:
    file.write("")  # Clear the log file
model.setParam("LogFile", "First_stage.log")

# Optimize the model
model.optimize()

if model.status in [GRB.OPTIMAL, GRB.TIME_LIMIT, GRB.INTERRUPTED]:
```

```
    print("First␣stage␣optimization␣completed.")
    # Analyze pump_count solution to determine key polders
else:
    print("No␣feasible␣solution␣found␣in␣Gurobi␣Optimizer␣version␣11.0.1␣build␣v11.0.1rc0␣(
        win64␣-␣Windows␣11.0␣(22631.2))first␣stage.")
```

G

Second-Stage code

```
import gurobipy as gp
from gurobipy import GRB
import numpy as np

model = gp.Model("PolderOptimization")

tractor_pumps = [18, 18, 18, 18, 18, 18, 18, 18, 18, 20, 20, 20, 20, 30, 30, 30, 30, 30, 30, 45]
tractor_location = ["Anna␣Paulowna", "Kwadijk", "Anna␣Paulowna", "Anna␣Paulowna", "Kwadijk", "Anna␣Paulowna", "Anna␣Paulowna", "Anna␣Paulowna", "Anna␣Paulowna", "Anna␣Paulowna", "Anna␣Paulowna", "Kwadijk", "Kwadijk", "Anna␣Paulowna", "Anna␣Paulowna", "Kwadijk", "Anna␣Paulowna", "Anna␣Paulowna", "Anna␣Paulowna", "Anna␣Paulowna"]
tractor_type = ["Veneroni␣AT30-5", "Veneroni␣AT30-5", "Veneroni␣AT30-5", "Veneroni␣AT30-5", "Veneroni␣AT30-5", "Veneroni␣AT30-5", "Veneroni␣AT30-5", "Veneroni␣AT30-5", "Veneroni␣AT30-5",
                "BBA␣B300", "BBA␣B300", "BBA␣B300", "BBA␣B300", "Veneroni␣AT400/5", "Veneroni␣AT400/5", "Veneroni␣AT400/5", "Veneroni␣AT400/5", "Veneroni␣AT400/5", "Veneroni␣AT400/5",
                "Veneroni␣AT500/5"]

S = 1  # Penalty duration in timesteps
par_t_step = 6
pump_capacities = 60 * par_t_step * np.array(tractor_pumps)
num_timesteps = len(next(iter(selected_polder_data.values()))['precipitation_6hr'])
num_polders = len(selected_polder_data)
num_pumps = len(tractor_pumps)

# Variables
pump_assignment = model.addVars(num_timesteps, num_pumps, num_polders, vtype=GRB.BINARY, name="pump_assignment")
pump_moved_to_polder = model.addVars(num_timesteps, num_pumps, num_polders, vtype=GRB.BINARY, name="pump_moved_to_polder")

infiltration = model.addVars(num_timesteps, num_polders, lb=0, vtype=GRB.CONTINUOUS, name="infiltration_1st")
penalty_volume = model.addVars(num_timesteps, num_polders, lb=0, name="penalty_volume")

water_volume_optimized = model.addVars(num_timesteps, num_polders, lb=0, name="water_volume_optimized")
max_water_volume_optimized = model.addVars(num_polders, lb=0, name="max_water_volume_optimized")
max_polder_damage_optimized = model.addVars(num_polders, ub=10, name="max_polder_damage_optimized")


for p in range(num_polders):
    max_polder_damage_optimized[p].UB = dam_upper_bound[p]
    max_water_volume_optimized[p].UB = vol_upper_bound[p]
    for t in range(num_timesteps):
        water_volume_optimized[t,p].UB = vol_upper_bound[p]
        infiltration[t,p].UB = infiltration_upper_bound[p]
        penalty_volume[t,p].UB = 45*60*par_t_step

# Constraints
```

```
# Initial pump assignment (all pumps unassigned at time 0)
for k in range(num_pumps):
    model.addConstr(gp.quicksum(pump_assignment[0,k,p] for p in range(num_polders)) == 0,
        name=f"initial_assignment_{k}")

# 1 polder per pump per timestep
for t in range(num_timesteps):
    for k in range(num_pumps):
        model.addConstr(gp.quicksum(pump_assignment[t,k,p] for p in range(num_polders)) <= 1,
             name=f"assignment_limit_{t}_{k}")

max_pumps_per_polder = 3
for t in range(num_timesteps):
    for p in range(num_polders):
        model.addConstr(gp.quicksum(pump_assignment[t,k,p] for k in range(num_pumps)) <=
            max_pumps_per_polder, name=f"max_pumps_per_polder_{t}_{p}")

# Detect movement
for t in range(1, num_timesteps):
    for k in range(num_pumps):
        for p in range(num_polders):
            # Define pump_moved_to_polder
            model.addConstr(pump_moved_to_polder[t,k,p] >= pump_assignment[t,k,p] -
                pump_assignment[t-1,k,p], name=f"moved_to_polder_{t}_{k}_{p}")

# Add constraint: each pump can be moved at most once during the time horizon
for k in range(num_pumps):
    model.addConstr(
        gp.quicksum(pump_moved_to_polder[t, k, p] for t in range(1, num_timesteps) for p in
            range(num_polders)) <= 2, name=f"max_one_move_{k}")

# Calculate penalty_volume
for t in range(num_timesteps):
    for p in range(num_polders):
        penalty_terms = []
        for k in range(num_pumps):
            for d in range(S):
                if t - d >= 0:
                    penalty_terms.append(pump_moved_to_polder[t - d, k, p] * (pump_capacities
                        [k]*0.5))
        if penalty_terms:
            model.addConstr(penalty_volume[t, p] == gp.quicksum(penalty_terms), name=f"
                penalty_volume_{t}_{p}")
        else:
            model.addConstr(penalty_volume[t, p] == 0, name=f"penalty_volume_{t}_{p}")

net_inflow_array = np.zeros((num_timesteps, num_polders))
tot_pump_capacity = np.zeros((num_timesteps, num_polders))

for p, (polder, data) in enumerate(selected_polder_data.items()):
    volume_array = data['volume']
    wlvl_array = data['wlvl']
    damage_array = np.array(data['damage']['jun_gem_72u_120h']['total'])/1e6
    area_array = data['area']
    area = data['total_area']
    gemalen_capacity = data['gemalen_cap'] * 60 * par_t_step
    precip_gen = data['precipitation_6hr']
    volume_array, damage_array, wlvl_array, area_array = remove_duplicates_across_arrays(
        volume_array, damage_array, wlvl_array, area_array, area)
    initial_volume = data['init_vol_zomer']

    baseline_vol = initial_volume
    max_vol = baseline_vol

    for t in range(num_timesteps):
        net_inflow = precip_gen[t]*area/1000 - gemalen_capacity
        net_inflow_array[t, p] = net_inflow

        if t == 0:
            model.addConstr(water_volume_optimized[t,p] == initial_volume, name=f"
                initial_volume_{t}_{p}")
        else:
            total_pump_pumping_capacity = gp.quicksum(pump_assignment[t,k,p] *
                pump_capacities[k] for k in range(num_pumps))

```

```
            model.addConstr(water_volume_optimized[t,p] == water_volume_optimized[t-1,p] +
                net_inflow - total_pump_pumping_capacity
                            + penalty_volume[t,p] + infiltration[t,p], name=f"water_balance_{
                                t}_{p}")

            baseline_vol += net_inflow
            if baseline_vol > max_vol:
                max_vol = baseline_vol

        model.addConstr(max_water_volume_optimized[p] >= water_volume_optimized[t,p], name=f"
            max_vol_opt_{t}_{p}")

        sum_inflow += net_inflow
        sum_gemalen += 1.5*gemalen_capacity

        if sum_inflow > sum_gemalen:
            model.addConstr(infiltration[t,p] == 0, name=f"infiltration_no_overflow_{t}_{p}")
        else:
            model.addConstr(infiltration[t,p] <= gemalen_capacity, name=f"infiltration_{t}_{p
                }")

    model.addGenConstrPWL(max_water_volume_optimized[p], max_polder_damage_optimized[p],
        volume_array, damage_array, name=f"pwl_damage_optimized_{p}")

pump_operation_cost = 0.001
move_penalty = 0.02
damage_reduction = gp.quicksum(max_polder_damage_optimized[p] for p in range(num_polders))
operational_cost = gp.quicksum(pump_assignment[t, k, p] * pump_operation_cost
                               for t in range(num_timesteps)
                               for k in range(num_pumps)
                               for p in range(num_polders))

total_objective = damage_reduction + operational_cost # + total_movement_penalty
model.setObjective(total_objective, GRB.MINIMIZE)

model.setParam("PoolGap", 0.20)
model.setParam("PoolSolutions", 5)
model.setParam("TimeLimit", 12*3600)
with open('second_stage.log', "w") as file:
    file.write("")  # Clear the log file
model.setParam("LogFile", "second_stage.log")

model.optimize()


if model.status == GRB.OPTIMAL:
    optimized_damage_total = sum(max_polder_damage_optimized[p].X for p in range(num_polders)
        )
    print(f"Optimized␣Total␣Damage:␣{optimized_damage_total}")
else:
    print("No␣optimal␣solution␣found.")
```

H

Second-Stage Water Volume and Pump Placement

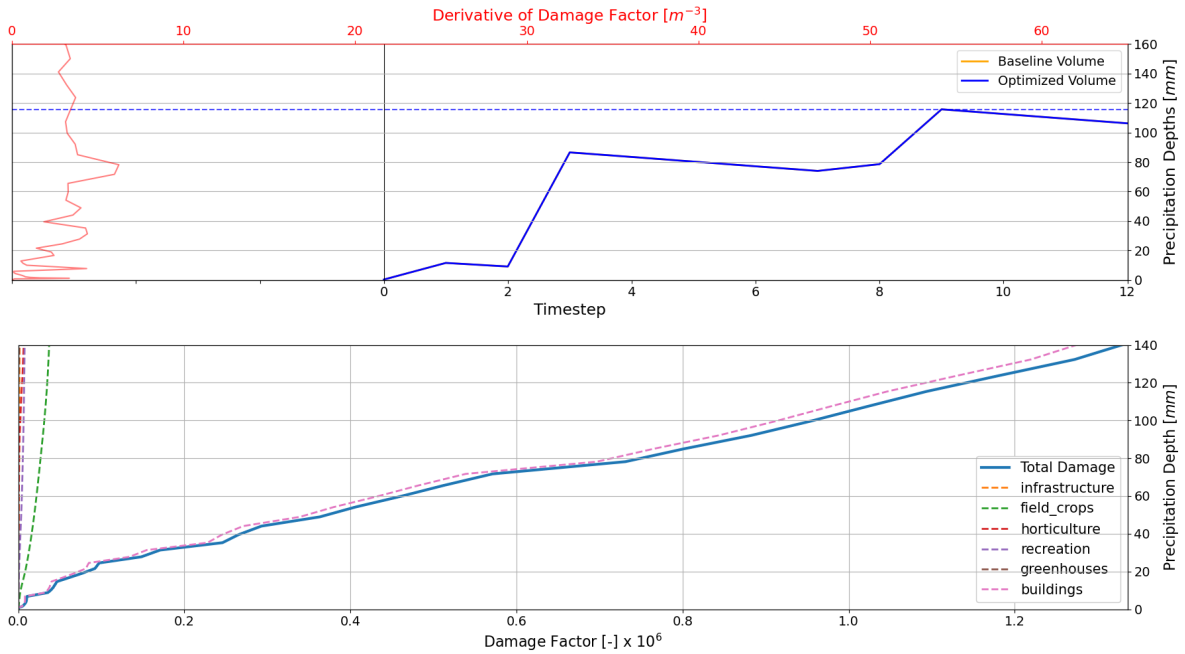


Figure H.1: Aagtdorperpolder

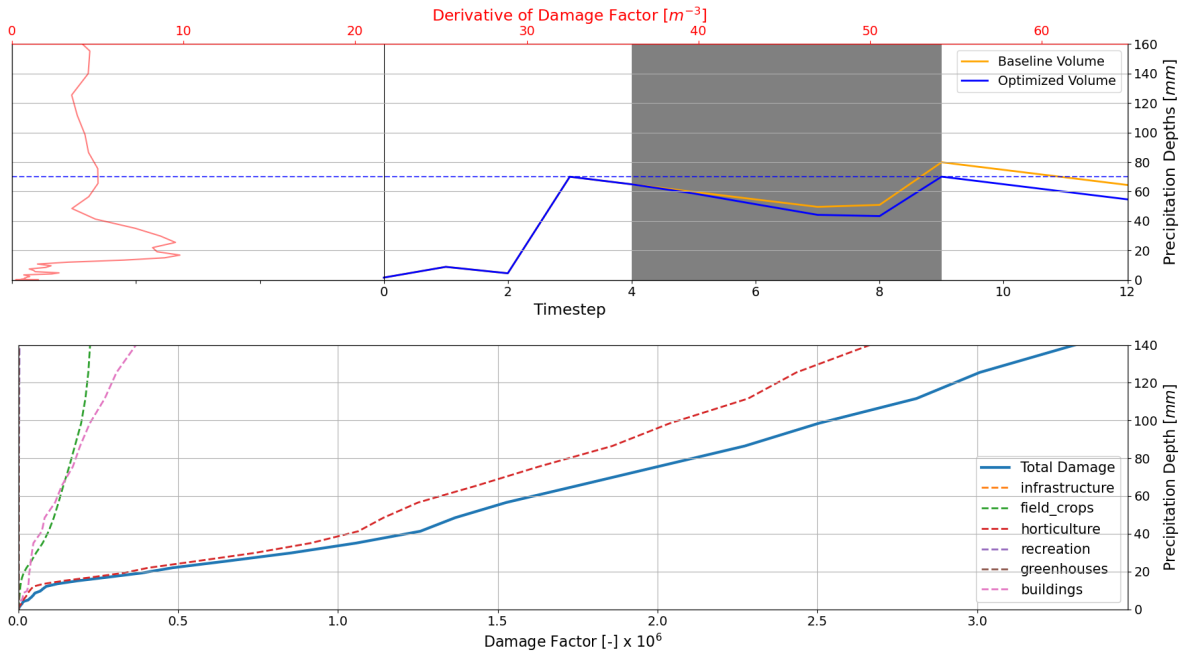


Figure H.2: Afd. H-ON

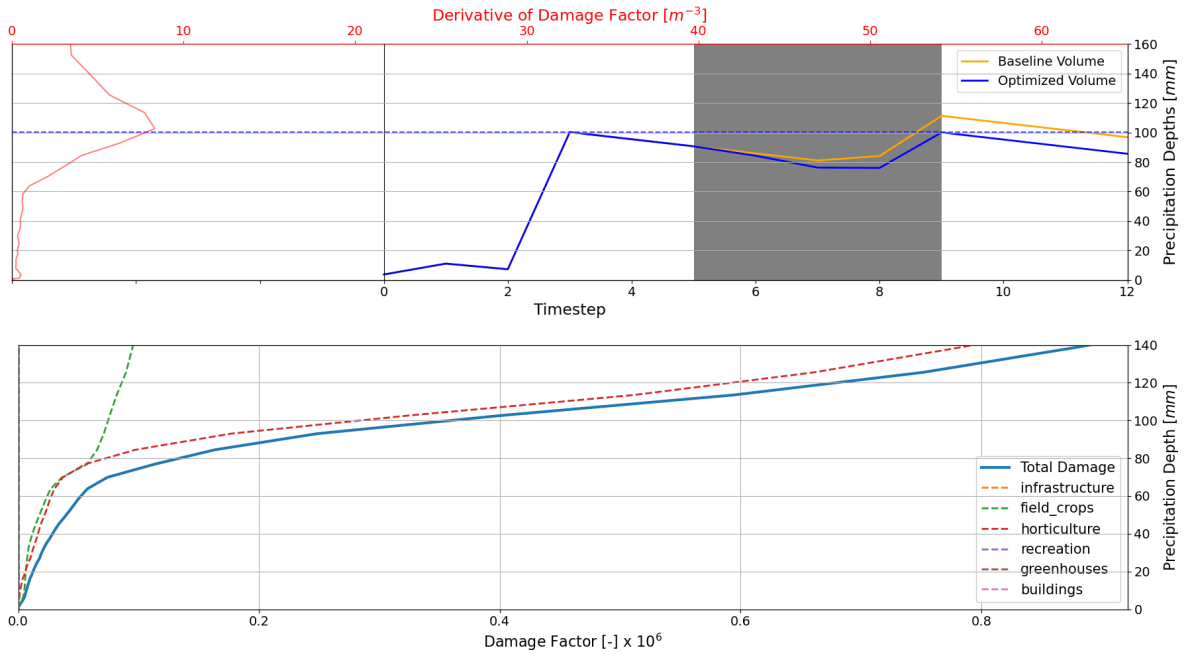


Figure H.3: Afd. I-noord

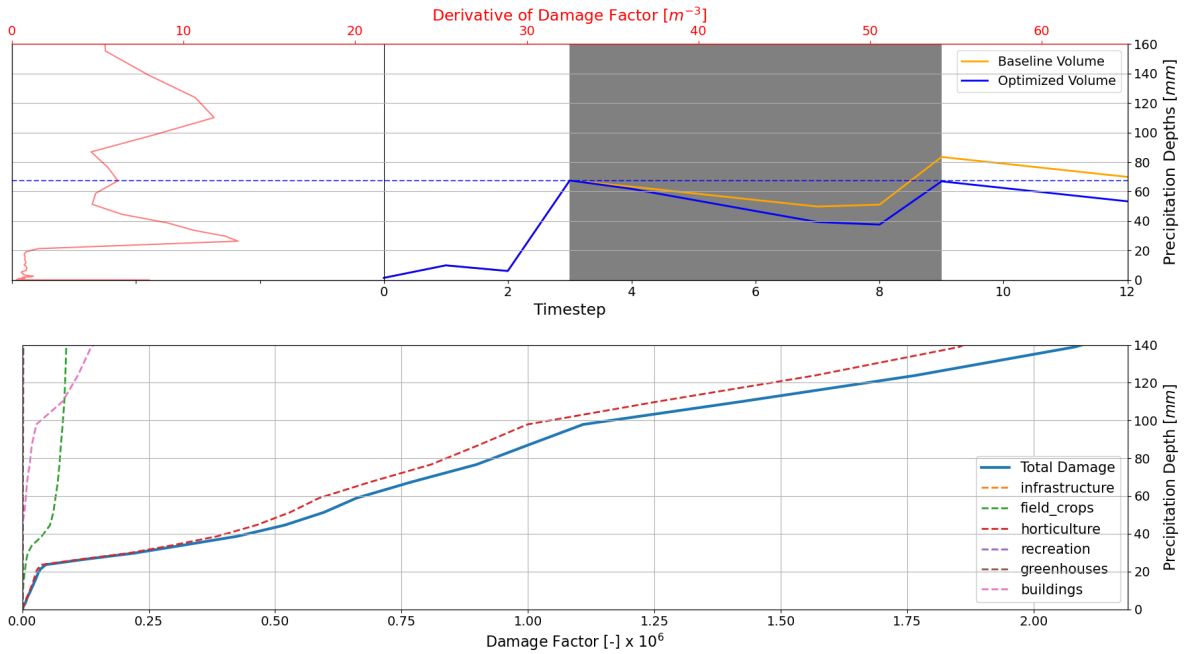


Figure H.4: Afd. NG

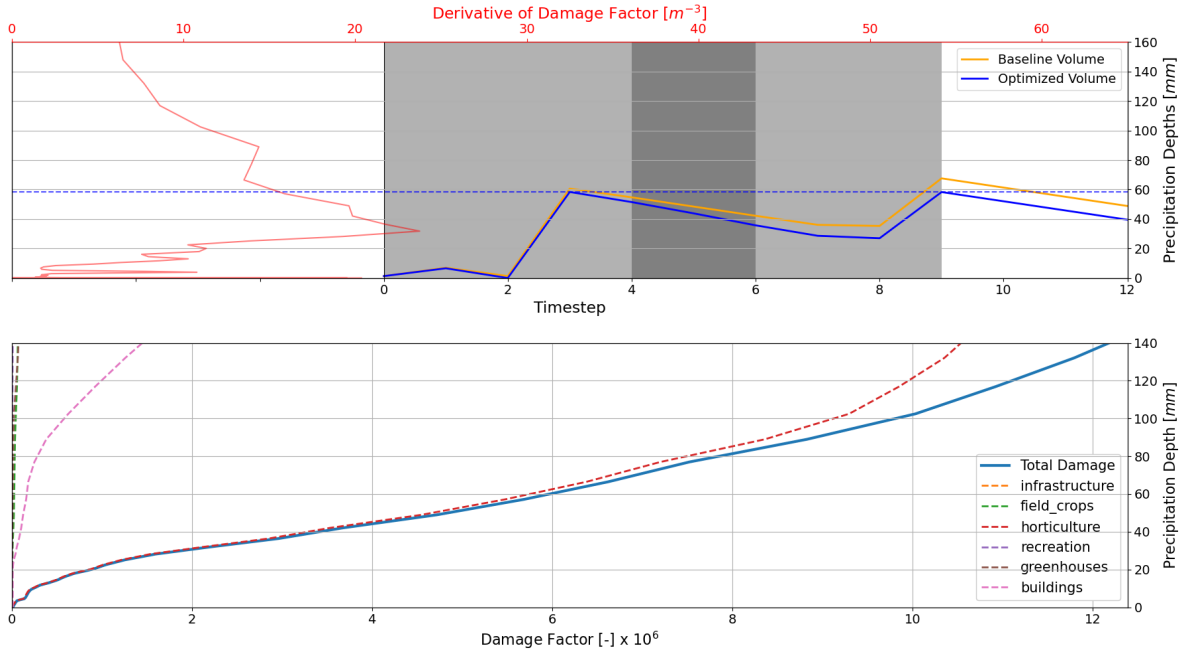


Figure H.5: Afd. NMR

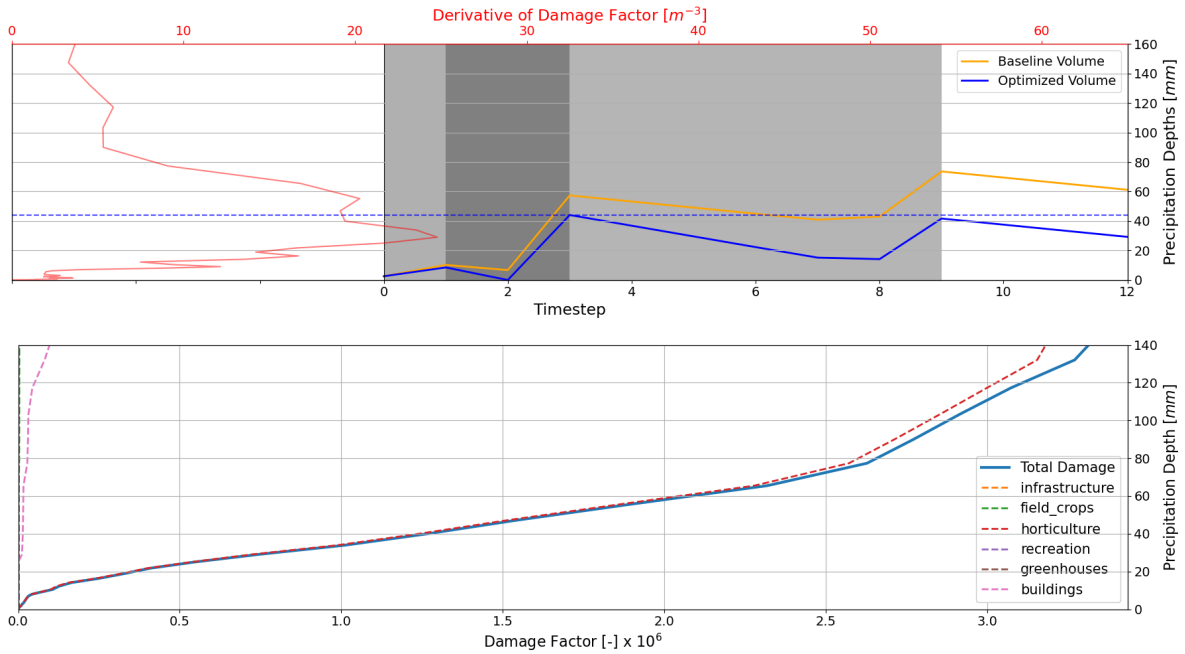


Figure H.6: Afd. NS

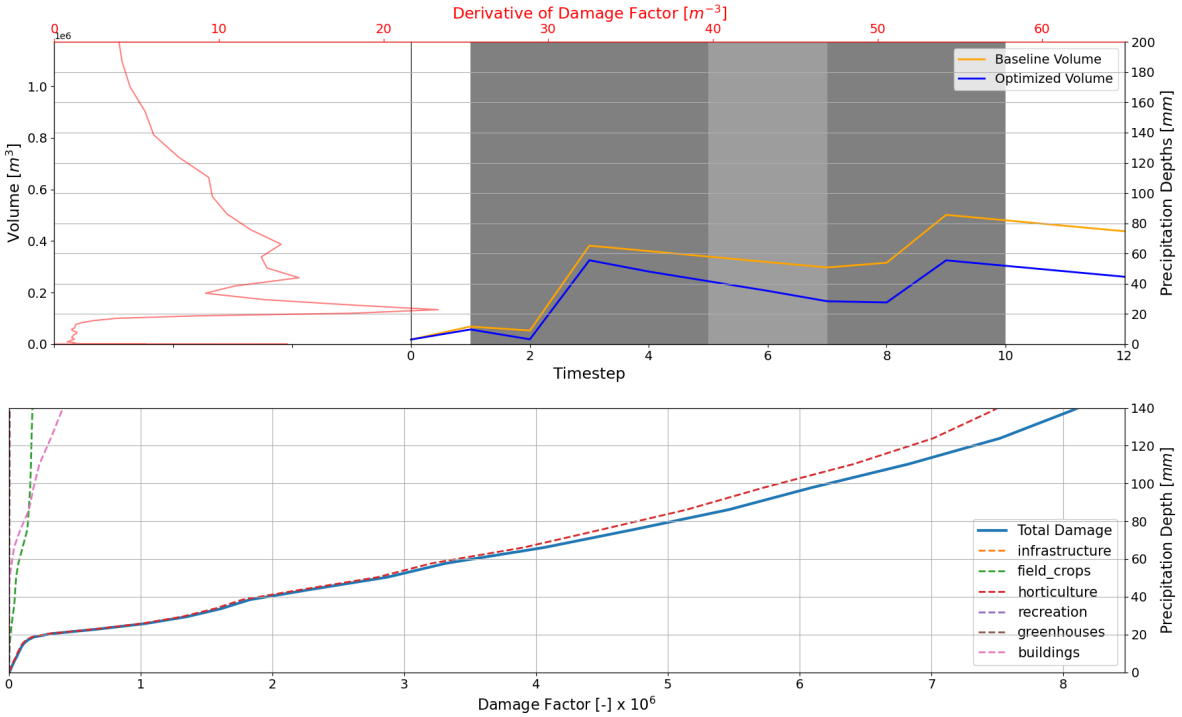


Figure H.7: Afd. OT-PV

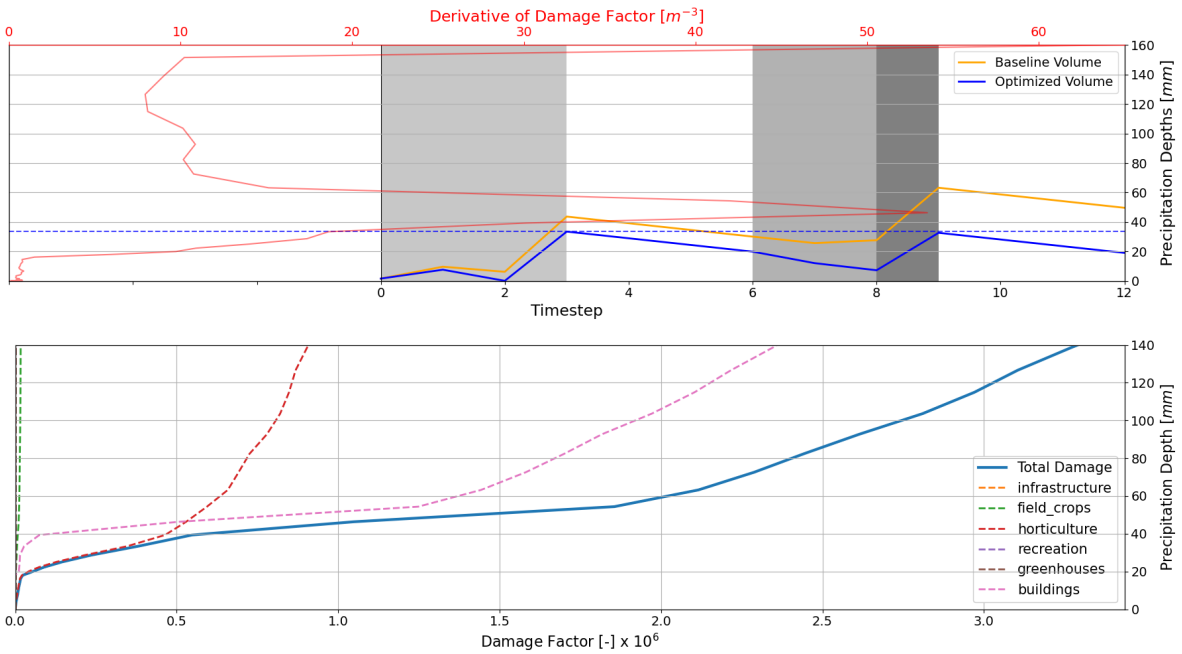


Figure H.8: Afd. W

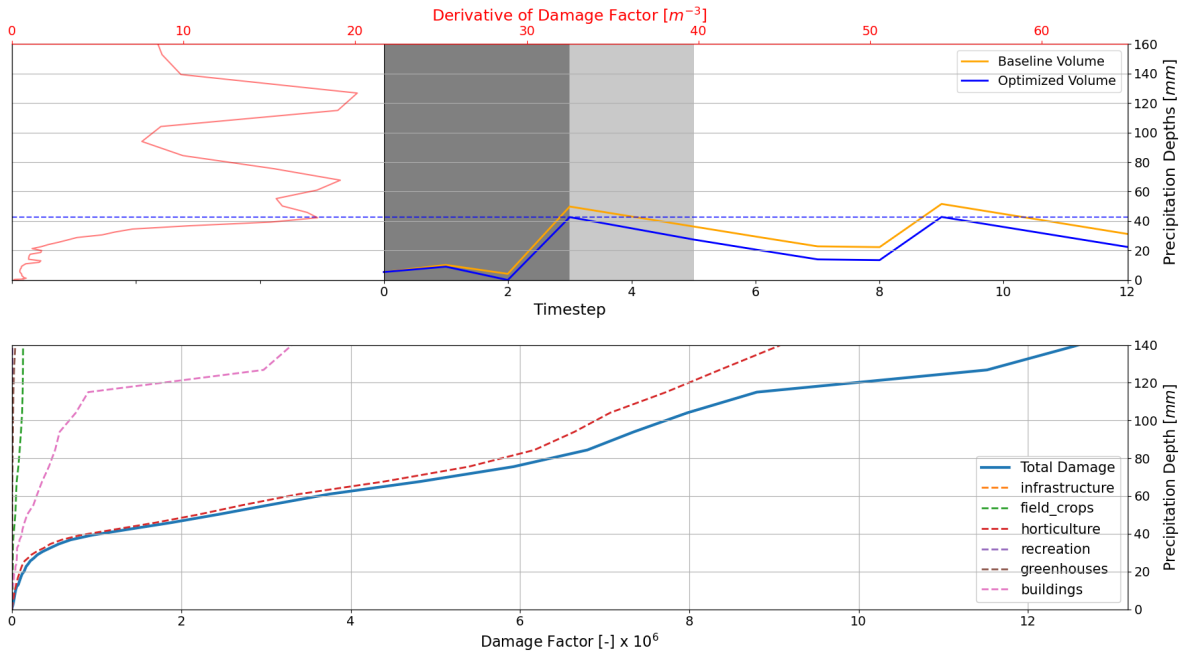


Figure H.9: Afd. Z

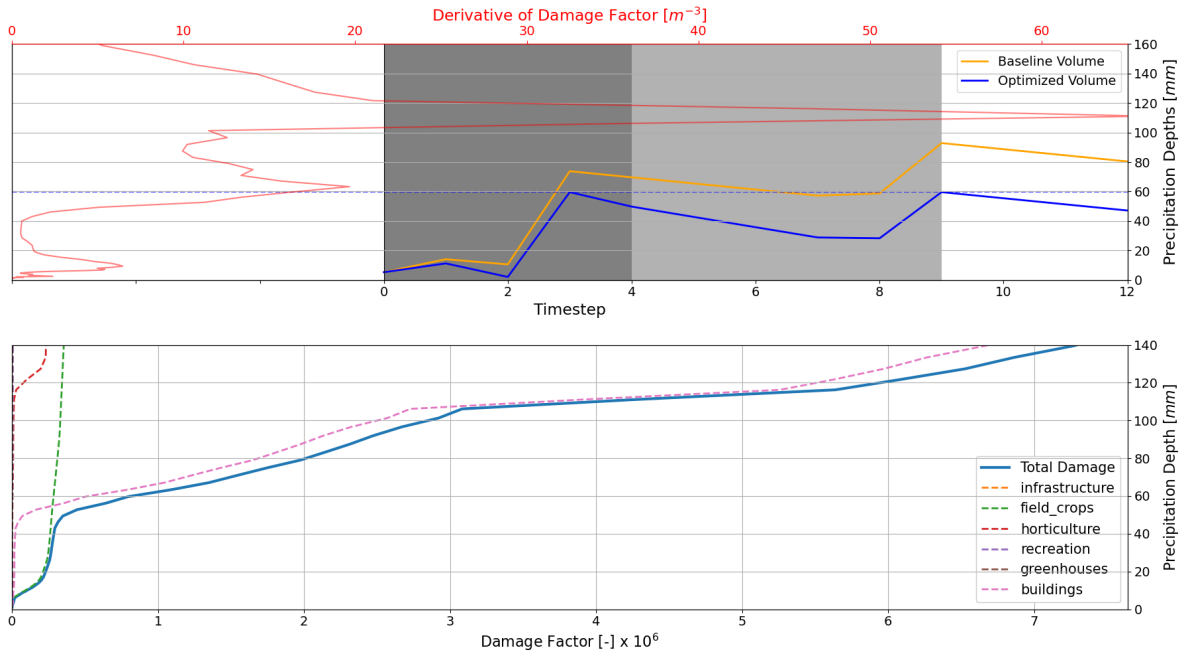


Figure H.10: Afd. ZG-ZM

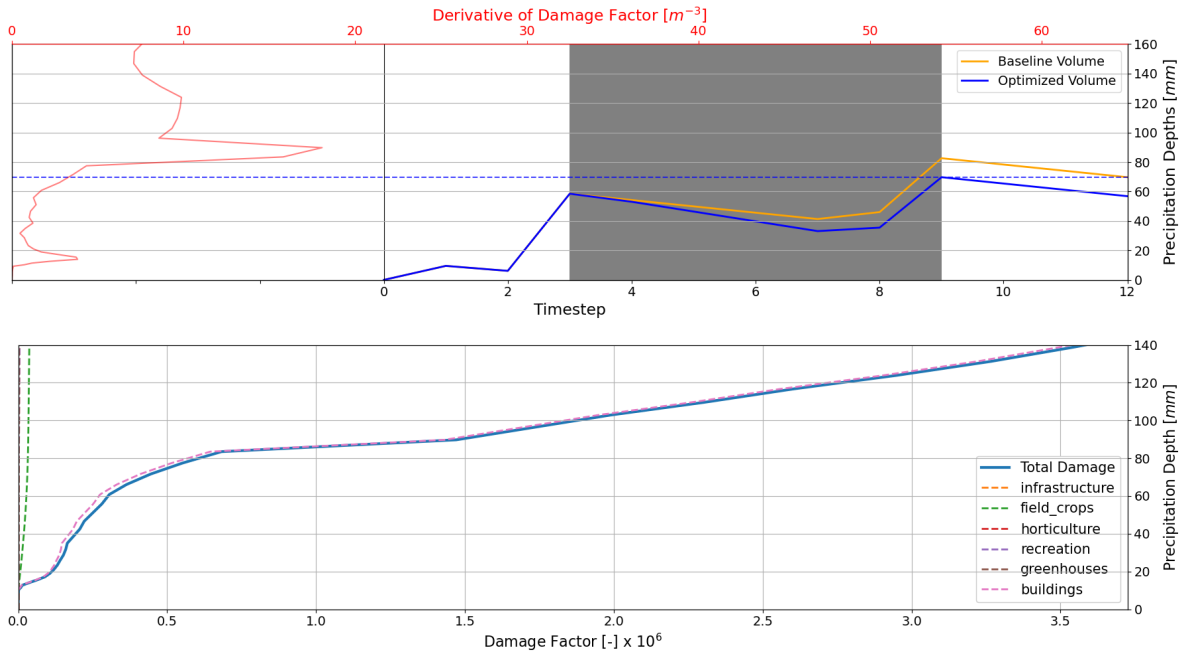


Figure H.11: Baafjespolder

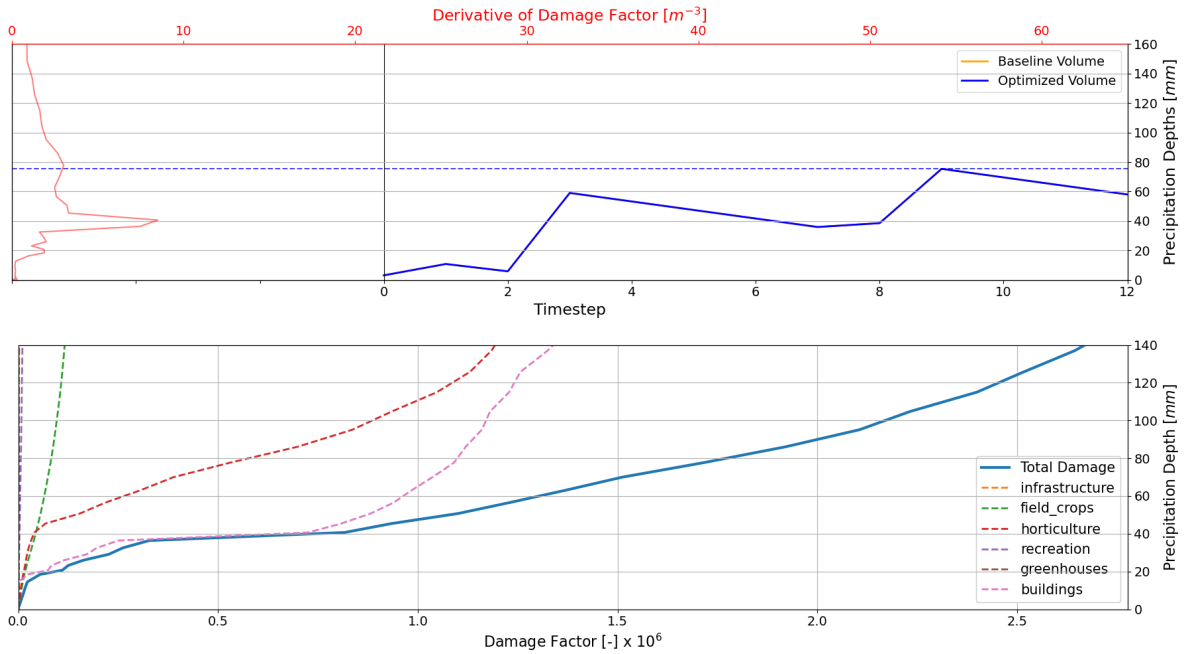


Figure H.12: Bergermeer

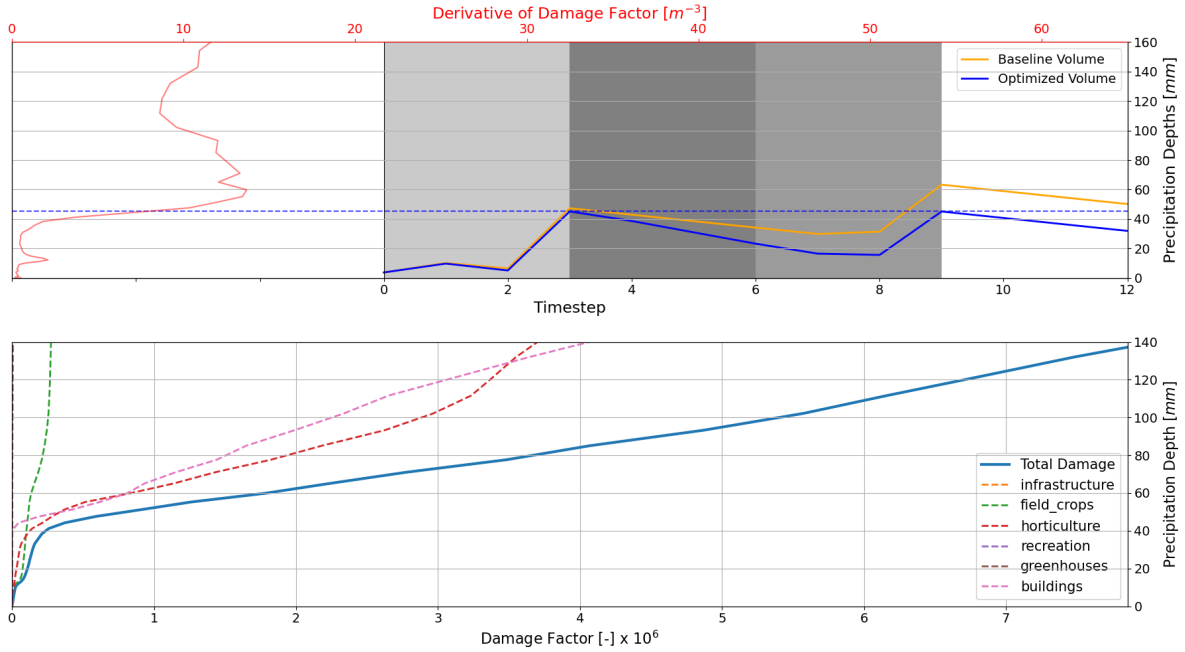


Figure H.13: Callantsoog

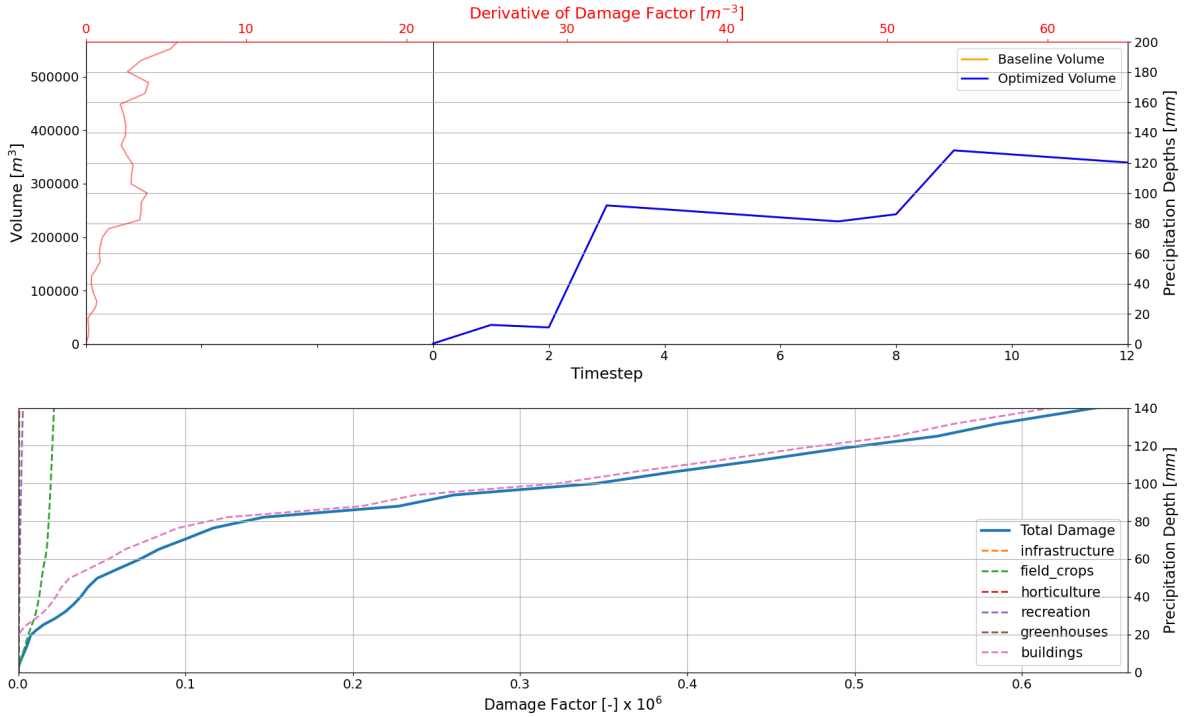


Figure H.14: Damlanderpolder

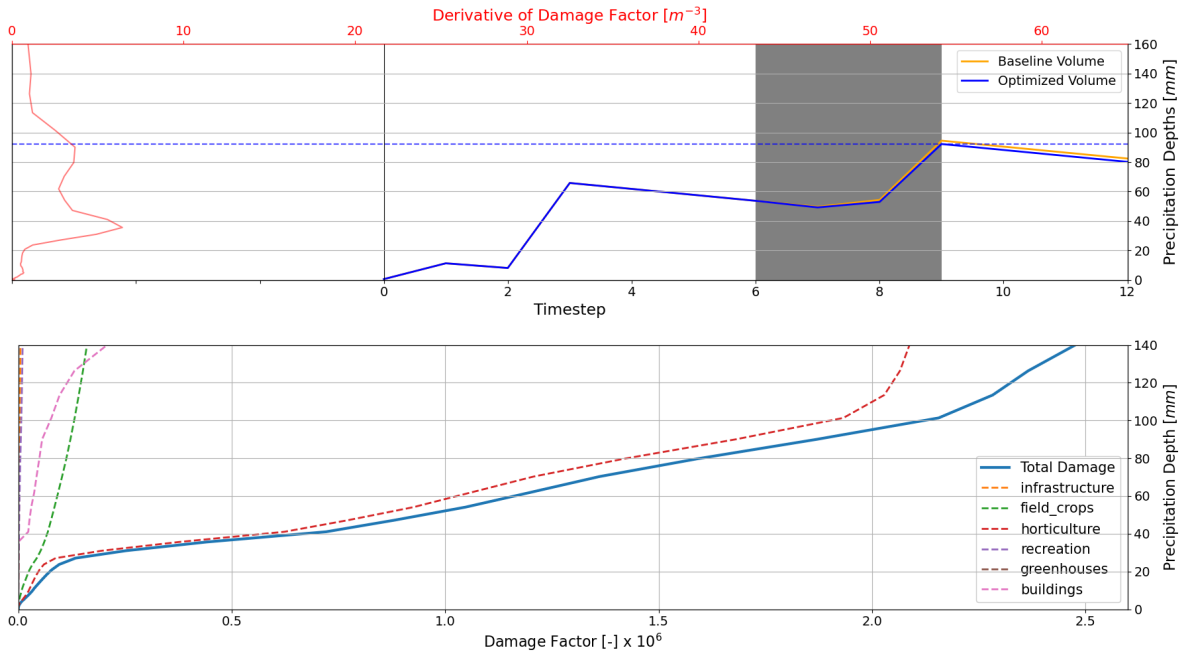


Figure H.15: Egmondermeer

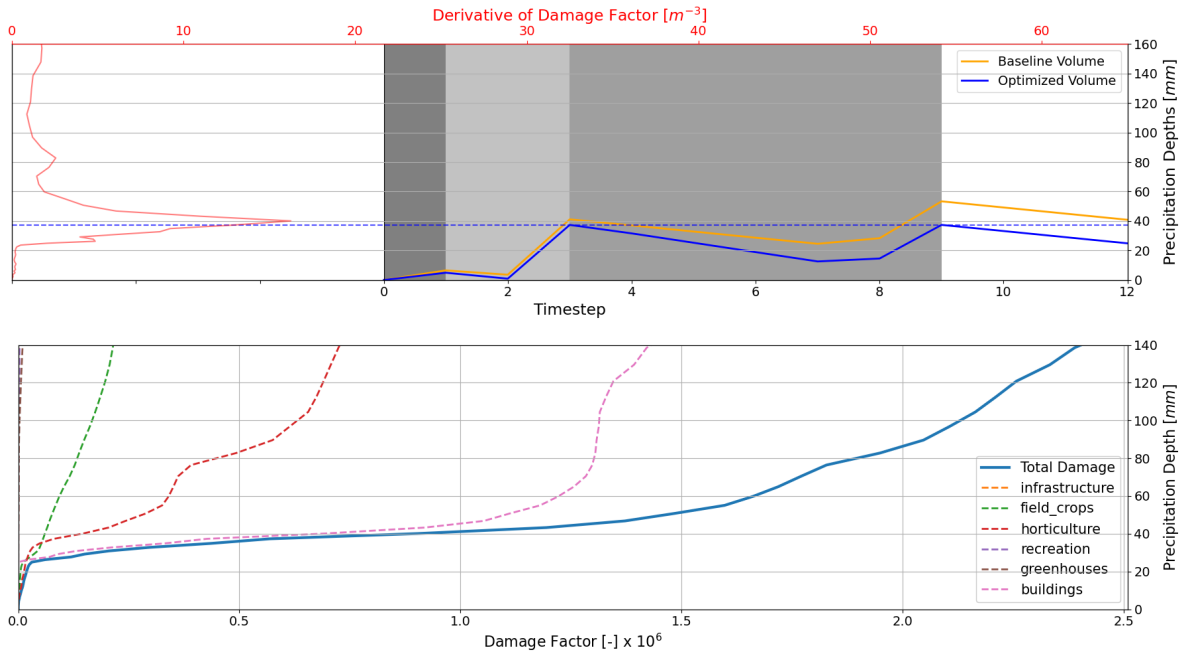


Figure H.16: Polder Schagerwaard

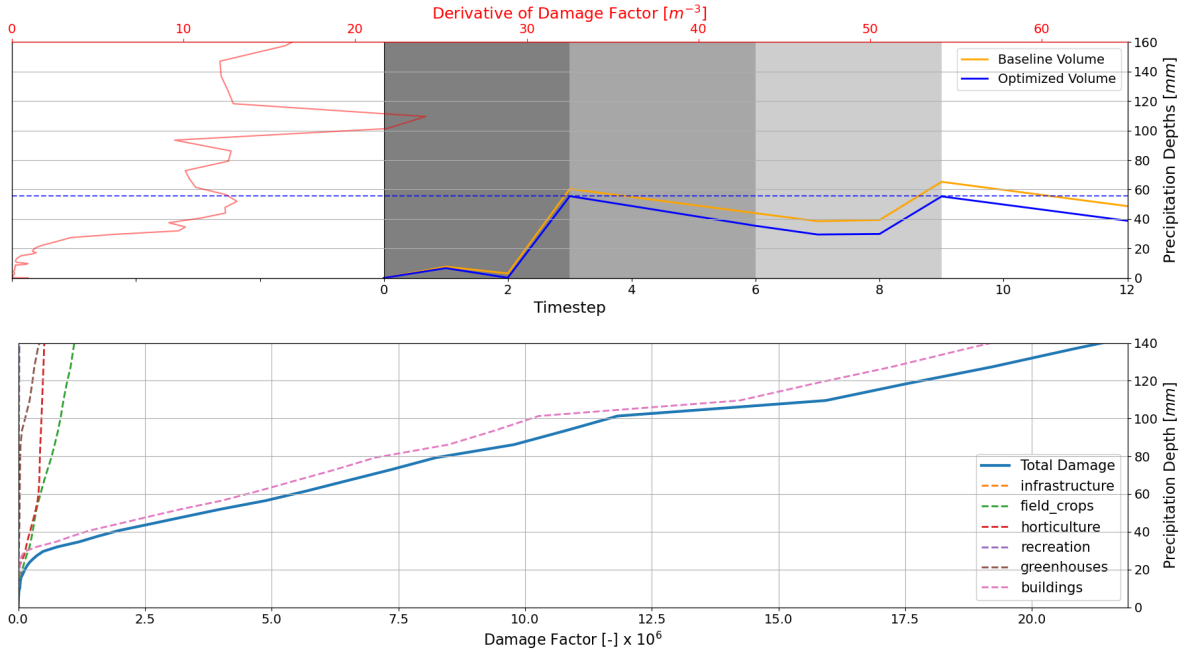


Figure H.17: Ringpolder

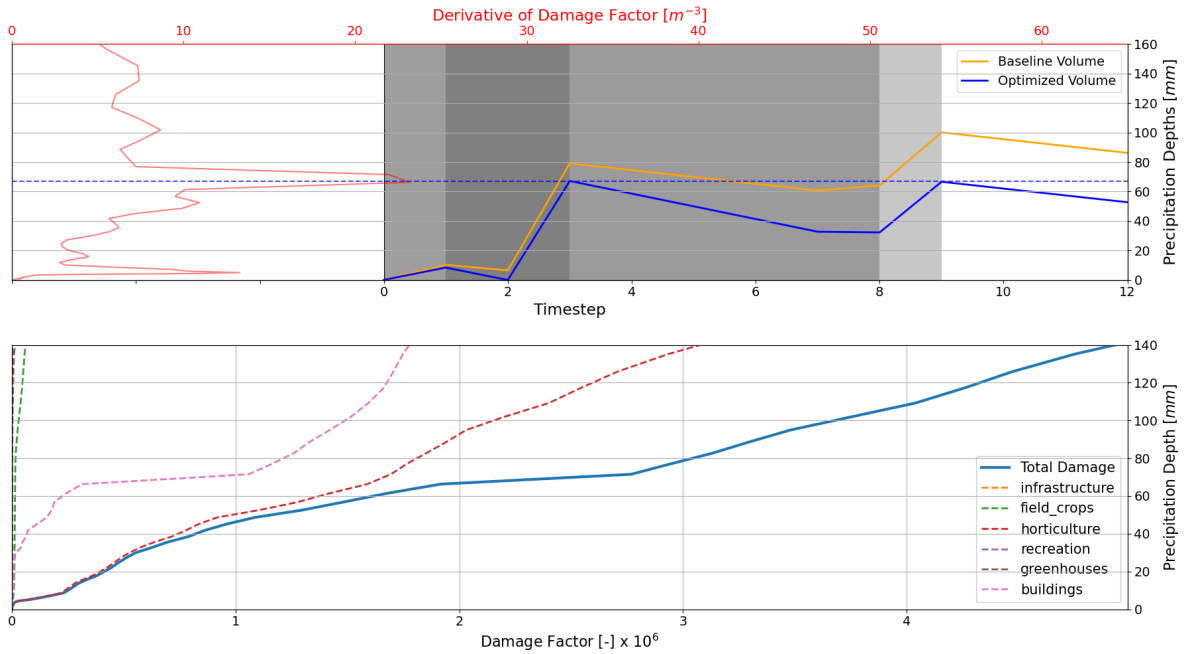


Figure H.18: Sammerspolder

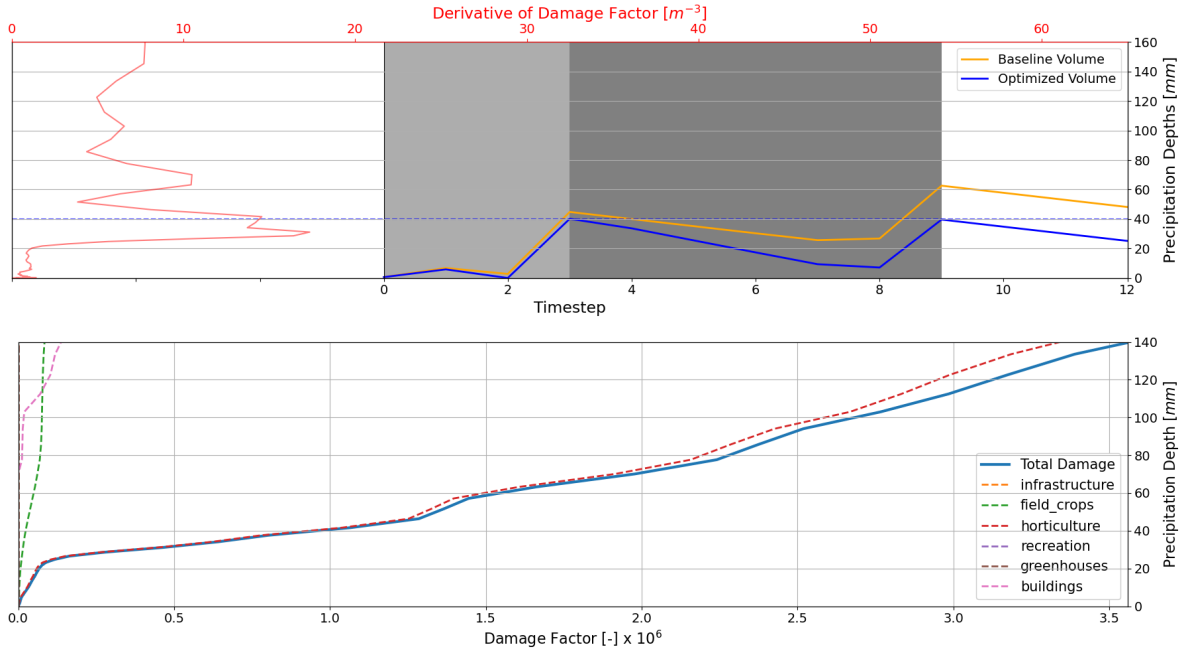


Figure H.19: 't Hoekje

KNMI HARMONIE Cy43

The precipitation forecast of KNMI is never exactly up to date, and always lags 1 or 2 hours. So to calculate the precipitation forecast, the current time has to be calculated and found from the latest forecast and the forecast 6 hours in the future (in this case) has to be found. The GRIB data can be obtained via commands and codes. In the case of precipitation, this code is 181 [5]. Then, the timeRangeIndicator can be set to either 4 for cumulative precipitation (from the start of the .tar file) or 0, per hour precipitation. This the code below the cumulative is calculated. The steps are as follows:

1. The (zipped) .tar folder is extracted.
2. Using pygrib, the precipitation values are extracted for the timeframes and georeferenced in WGS84.
3. The relevant forecasts (6 hours in the future in this case) is extracted and converted to raster.
4. Precipitation mean per polder is calculated given a polder GeoJSON file for the polder locations (WGS84).

```
import os
import tarfile
from datetime import datetime, timedelta
import pytz
import numpy as np
import pygrib
import rasterio
from rasterio.transform import from_origin
from rasterstats import zonal_stats
import geopandas as gpd

# Define paths
extract_dir = "data/results"
results_dir = "data/results"
geojson_path = 'data/source/polders.geojson'
reprojected_geojson_path = 'data/source/polders_reprojected.geojson'
output_geojson = 'data/source/polders_with_precipitation.geojson'
output_tiff = 'precipitation.tif'

# Extract GRIB files from TAR archive
tar_path = [file for file in os.listdir(extract_dir) if file.endswith(".tar")][0]
with tarfile.open(os.path.join(extract_dir, tar_path), "r") as tar:
    tar.extractall(extract_dir)

# Filter and load GRIB files
grib_files = [file for file in os.listdir(extract_dir) if file.endswith("_GB")]
latest_forecast = datetime.strptime(grib_files[0].split("_")[2], "%Y%m%d%H%M")
current_time = datetime.utcnow().replace(tzinfo=pytz.utc)
delta_time = current_time - latest_forecast
time_6h = current_time + timedelta(hours=6)

# Calculate forecast intervals
gribfile_now = f"{int(delta_time.total_seconds() // 3600):03d}00"
gribfile_6h = f"{int((time_6h - latest_forecast).total_seconds() // 3600):03d}00"

# Get GRIB files for the desired timeframes
```

```
grib_files_filtered = [file for file in grib_files if file.split("_")[3] in [gribfile_now, gribfile_6h]]
values_list, lats, lons = [], None, None

# Extract precipitation data from GRIB files
for grib_file in grib_files_filtered:
    with pygrib.open(os.path.join(extract_dir, grib_file)) as grbs:
        for grb in grbs:
            if grb.typeOfLevel == 'heightAboveGround' and grb.indicatorOfParameter == 181:
                values = grb.values[::-1]
                lats, lons = grb.latlons()
                values_list.append(values)

# Create a GeoTIFF from precipitation data
transform = from_origin(lons[0, 0], lats[0, 0] + values.shape[0] * (lats[1, 0] - lats[0, 0]),
                        lons[0, 1] - lons[0, 0], lats[1, 0] - lats[0, 0])
metadata = {
    'driver': 'GTiff', 'count': 1, 'dtype': 'float32', 'width': values.shape[1],
    'height': values.shape[0], 'crs': 'EPSG:4326', 'transform': transform
}
with rasterio.open(output_tiff, 'w', **metadata) as dst:
    dst.write(values.astype(np.float32), 1)

# Reproject GeoJSON to match raster CRS
def reproject_geojson(src_path, dst_path, target_crs):
    gdf = gpd.read_file(src_path)
    gdf.to_crs(target_crs).to_file(dst_path, driver='GeoJSON')

reproject_geojson(geojson_path, reprojected_geojson_path, 'EPSG:4326')

# Calculate zonal statistics for each polder
polders = gpd.read_file(reprojected_geojson_path)
with rasterio.open(output_tiff) as src:
    stats = zonal_stats(polders, src.read(1), affine=src.transform, all_touched=True, nodata=src.nodata, stats='mean')

# Add precipitation statistics to GeoJSON
polders['mean_precipitation'] = [stat['mean'] for stat in stats]
polders.to_file(output_geojson, driver='GeoJSON')
print("Average precipitation added to polders GeoJSON.")
```

J

Logfile First Stage

```
Gurobi 11.0.1 (win64) logging started Fri Jan 31 14:56:08 2025

Set parameter LogFile to value "test_First_stage.log"
Gurobi Optimizer version 11.0.1 build v11.0.1rc0 (win64 - Windows 11.0 (22631.2))

CPU model: AMD Ryzen 7 4700U with Radeon Graphics, instruction set [SSE2|AVX|AVX2]
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 4825 rows, 3696 columns and 10656 nonzeros
Model fingerprint: 0xdca6eabd
Model has 48 general constraints
Variable types: 2496 continuous, 1200 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 4e+03]
  Objective range  [1e-03, 1e+00]
  Bounds range     [6e-03, 3e+06]
  RHS range        [2e-01, 9e+05]
  PWLCon x range   [2e-02, 3e+06]
  PWLCon y range   [0e+00, 3e+01]
Presolve removed 3920 rows and 918 columns
Presolve time: 0.03s
Presolved: 905 rows, 2778 columns, 6973 nonzeros
Presolved model has 48 SOS constraint(s)
Variable types: 1751 continuous, 1027 integer (13 binary)

Root relaxation: objective 3.970998e+01, 1674 iterations, 0.01 seconds (0.01 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   39.70998    0  110          -   39.70998      -     -    0s
H    0     0                       99.3450424   39.70998  60.0%     -    0s
H    0     0                       42.9522744   39.70998  7.55%     -    0s
     0     0   39.80295    0   65   42.95227   39.80295  7.33%     -    0s
     0     0   39.81941    0   60   42.95227   39.81941  7.29%     -    0s
     0     0   39.83671    0   61   42.95227   39.83671  7.25%     -    0s
H    0     0                       42.7304410   39.94132  6.53%     -    0s
     0     0   39.94132    0   59   42.73044   39.94132  6.53%     -    0s
     0     0   40.00686    0   51   42.73044   40.00686  6.37%     -    0s
     0     0   40.12618    0   52   42.73044   40.12618  6.09%     -    0s
     0     2   40.12618    0   52   42.73044   40.12618  6.09%     -    0s
H  220   240                       42.6958403   40.23159  5.77%  20.3    1s
```

```
H  228   240                      42.6522459   40.23159  5.68%  20.0    1s
H  236   240                      42.5636423   40.23159  5.48%  19.9    1s
H  312   338                      42.4590410   40.23159  5.25%  18.1    1s
H  355   402                      42.4375315   40.23159  5.20%  16.7    1s
H  526   569                      42.4236784   40.23159  5.17%  14.5    2s
H  615   624                      42.3279863   40.23159  4.95%  14.1    2s
H 1745  1416                      42.3204105   40.23159  4.94%  10.7    3s
H 1752  1412                      42.3190544   40.23159  4.93%  10.6    3s
H 1760  1311                      42.2896668   40.23159  4.87%  10.6    3s
H 1877  1447                      42.2820686   40.23159  4.85%  10.5    3s
H 1990  1519                      42.2775334   40.23159  4.84%  10.6    3s
H 1990  1517                      42.2756457   40.23159  4.84%  10.6    3s
H 2615  1901                      42.1290126   41.18203  2.25%  10.5    5s
H 2615  1806                      42.1290125   41.18203  2.25%  10.5    5s
H 3000  1957                      42.1265134   41.55299  1.36%  11.4    6s
H 3011  1867                      42.1002787   41.55299  1.30%  11.4    6s
H 3023  1781                      42.0942339   41.55299  1.29%  11.4    6s
H 3043  1705                      42.0936001   41.55299  1.28%  11.3    7s
H 3045  1630                      42.0929158   41.55299  1.28%  11.3    7s
H 3049  1559                      42.0895175   41.55299  1.27%  11.3    7s
H 3200  1589                      42.0423683   41.55299  1.16%  10.9    8s
H 3208  1531                      42.0403485   41.55299  1.16%  10.9    8s
H 4332  2187                      42.0403485   41.55299  1.16%   8.9    9s
H 4360  2121                      42.0403484   41.55299  1.16%   8.9    9s
  4507  2151   41.90363  139  107   42.04035   41.55299  1.16%   8.7   10s
H 4509  2095                      42.0378949   41.55299  1.15%   8.7   10s
H 5005  2349                      42.0378948   41.55299  1.15%   8.3   11s
H 5030  2291                      42.0378948   41.55299  1.15%   8.3   11s
H 5500  2856                      42.0378948   41.57036  1.11%   7.9   13s
  6172  3418   41.77555  144  130   42.03789   41.57036  1.11%   7.5   15s
  8714  5945   41.78404  106  131   42.03789   41.57052  1.11%   6.7   20s
 11297  7626 infeasible  129        42.03789   41.57066  1.11%   5.9   26s
 11936  8668   41.83370  206  134   42.03789   41.57066  1.11%   5.7   30s

                                    ...

 948513 782291   41.62854   97  148   42.03786   41.57332  1.11%   4.0 3531s
 949324 783462   41.87537  189  115   42.03786   41.57332  1.11%   4.0 3540s
 950940 784563   41.65331   97  140   42.03786   41.57332  1.11%   4.0 3550s
 952328 785507   41.87149  258  117   42.03786   41.57332  1.11%   4.0 3558s
 953474 785794   41.84086  154  132   42.03786   41.57332  1.11%   4.0 3567s
 953856 786759   41.62854   77  146   42.03786   41.57332  1.11%   4.0 3577s
 955261 787738   41.87725  222  112   42.03786   41.57332  1.11%   4.0 3590s
 956341 789161   41.69585  128  147   42.03786   41.57332  1.11%   4.0 3600s

Cutting planes:
  Gomory: 123
  Implied bound: 10
  MIR: 76
  Flow cover: 20
  RLT: 1
  Relax-and-lift: 2

Explored 958157 nodes (3874363 simplex iterations) in 3600.35 seconds (924.04 work units)
Thread count was 8 (of 8 available processors)

Solution count 5: 42.0379 42.0379 42.0379 ... 42.0379

Time limit reached
Best objective 4.203786117178e+01, best bound 4.157332163341e+01, gap 1.1051%
```

K

Logfile Second Stage

```
Gurobi 11.0.1 (win64) logging started Thu Jan 30 16:47:56 2025

Set parameter LogFile to value "Ssecond_stage.log"
Gurobi Optimizer version 11.0.1 build v11.0.1rc0 (win64 - Windows 11.0 (22631.2))

CPU model: AMD Ryzen 7 4700U with Radeon Graphics, instruction set [SSE2|AVX|AVX2]
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 6095 rows, 10659 columns and 39919 nonzeros
Model fingerprint: 0xa2a5cae1
Model has 19 general constraints
Variable types: 779 continuous, 9880 integer (9880 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+04]
  Objective range  [1e-03, 1e+00]
  Bounds range     [1e+00, 3e+06]
  RHS range        [1e+00, 8e+05]
  PWLCon x range   [4e-01, 3e+06]
  PWLCon y range   [0e+00, 3e+01]
Presolve removed 130 rows and 153 columns
Presolve time: 0.09s
Presolved: 5965 rows, 10506 columns, 44166 nonzeros
Variable types: 1061 continuous, 9445 integer (9147 binary)

Root relaxation: objective 2.420623e+01, 28193 iterations, 2.10 seconds (2.67 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   24.20623    0  377          -   24.20623      -     -    2s
     0     0   26.78435    0  602          -   26.78435      -     -    9s
H    0     0                       33.4691527   26.82019  19.9%     -   15s
     0     0   26.83049    0  596   33.46915   26.83049  19.8%     -   15s
     0     0   26.88009    0  576   33.46915   26.88009  19.7%     -   18s
     0     0   26.89114    0  556   33.46915   26.89114  19.7%     -   19s
     0     0   26.89116    0  572   33.46915   26.89116  19.7%     -   20s
     0     0   26.93982    0  605   33.46915   26.93982  19.5%     -   23s
     0     0   26.99880    0  588   33.46915   26.99880  19.3%     -   27s
     0     0   27.00150    0  584   33.46915   27.00150  19.3%     -   28s
     0     0   27.00150    0  608   33.46915   27.00150  19.3%     -   28s
     0     0   27.02171    0  612   33.46915   27.02171  19.3%     -   32s
     0     0   27.02256    0  614   33.46915   27.02256  19.3%     -   34s
```

	0	0	27.02257	0	622	33.46915	27.02257	19.3%	-	34s
	0	0	27.03133	0	589	33.46915	27.03133	19.2%	-	37s
H	0	0				33.2737387	27.03133	18.8%	-	37s
	0	0	27.03161	0	612	33.27374	27.03161	18.8%	-	38s
H	0	0				33.1853307	27.03425	18.5%	-	40s
	0	0	27.03425	0	670	33.18533	27.03425	18.5%	-	40s
H	0	0				32.1597155	27.03577	15.9%	-	41s
	0	0	27.03577	0	645	32.15972	27.03577	15.9%	-	41s
	0	0	27.03748	0	654	32.15972	27.03748	15.9%	-	42s
	0	0	27.03795	0	686	32.15972	27.03795	15.9%	-	46s
H	0	0				31.9978276	27.03961	15.5%	-	49s
H	0	0				31.8736718	27.03961	15.2%	-	49s
	0	0	27.03961	0	734	31.87367	27.03961	15.2%	-	50s
	0	0	27.03961	0	685	31.87367	27.03961	15.2%	-	50s
	0	2	27.03961	0	676	31.87367	27.03961	15.2%	-	57s
	1	4	27.05387	1	675	31.87367	27.05387	15.1%	637	60s
	3	6	27.15889	2	674	31.87367	27.15889	14.8%	4482	83s
	7	10	27.27491	3	655	31.87367	27.15889	14.8%	22361	101s
	21	18	27.28456	5	774	31.87367	27.27876	14.4%	19568	117s
	29	26	27.53036	6	762	31.87367	27.27876	14.4%	21374	120s
	45	38	27.53218	7	668	31.87367	27.27876	14.4%	17407	129s
	53	46	27.69948	8	637	31.87367	27.27876	14.4%	16475	132s
	61	55	27.71204	9	660	31.87367	27.27876	14.4%	15077	135s
H	65	55				31.8270770	27.27876	14.3%	14362	135s
H	68	55				31.7494492	27.27876	14.1%	13903	135s
H	71	57				31.6335976	27.27876	13.8%	13629	138s
	81	62	27.72064	11	601	31.63360	27.27876	13.8%	12359	146s
	92	71	27.79388	11	635	31.63360	27.27876	13.8%	12317	151s
H	101	82				31.5824083	27.27876	13.6%	11809	159s
H	105	82				31.5703042	27.27876	13.6%	11925	159s
H	111	82				31.5157358	27.27876	13.4%	11565	159s
H	112	90				31.4776232	27.27876	13.3%	11537	178s
	120	103	28.00261	13	546	31.47762	27.27876	13.3%	12162	182s
	133	116	28.19446	13	599	31.47762	27.27876	13.3%	11595	186s
H	146	125				31.3752108	27.27876	13.1%	11086	198s
	155	142	28.00265	15	459	31.37521	27.27876	13.1%	11711	202s
	172	157	28.05518	17	490	31.37521	27.27876	13.1%	11044	207s
	187	170	28.04894	18	517	31.37521	27.27876	13.1%	10702	215s
H	191	170				31.3739251	27.27876	13.1%	10565	215s
	200	187	28.07686	20	537	31.37393	27.27876	13.1%	10467	220s
H	236	221				31.3739250	27.27876	13.1%	9691	230s
	251	240	28.05218	25	550	31.37393	27.27876	13.1%	9463	235s
	270	264	28.05295	26	520	31.37393	27.27876	13.1%	9229	241s
	294	288	28.07375	27	483	31.37393	27.27876	13.1%	8888	246s
H	305	288				31.3687520	27.27876	13.0%	8698	246s
	318	309	28.06659	29	502	31.36875	27.27876	13.0%	8583	251s
	339	330	28.06689	30	521	31.36875	27.27876	13.0%	8370	257s
H	360	353				31.3570544	27.27876	13.0%	8206	262s
H	374	353				31.3519768	27.27876	13.0%	8078	262s
	383	382	28.06715	34	501	31.35198	27.27876	13.0%	8008	267s
H	412	406				31.3519762	27.27876	13.0%	7737	274s
	436	435	28.09946	40	499	31.35198	27.27876	13.0%	7546	281s
	465	466	28.09981	43	523	31.35198	27.27876	13.0%	7391	288s
H	487	466				31.3519754	27.27876	13.0%	7237	288s
	496	496	28.22810	45	514	31.35198	27.27876	13.0%	7207	295s
	530	516	28.24111	48	491	31.35198	27.27876	13.0%	7021	302s
H	537	516				31.3519753	27.27876	13.0%	6981	302s
	550	553	28.23811	49	492	31.35198	27.27876	13.0%	6931	309s
	587	597	28.23311	51	406	31.35198	27.27876	13.0%	6764	316s

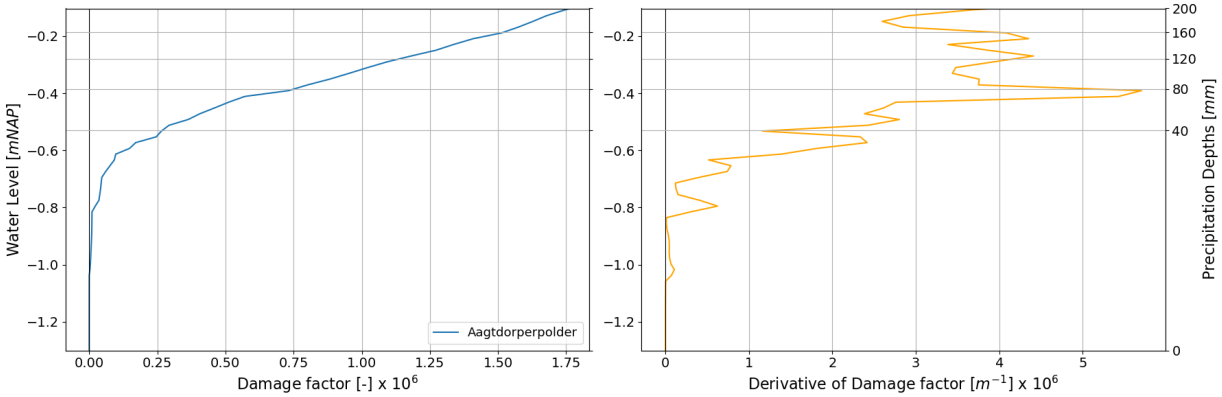
```
                                   ...

 160925 90246   30.54113   96  351   30.60160   30.28522  1.03%  3853 37754s
 162002 90710   30.45017   72  694   30.60160   30.28539  1.03%  3853 38025s
 162863 91229   30.49768   51  947   30.60160   30.28604  1.03%  3856 38298s
H163273 91229                      30.6016046   30.28619  1.03%  3855 38298s
 163720 91880   30.55204   88  568   30.60160   30.28635  1.03%  3858 38530s
 164643 92425   30.38042   55  793   30.60160   30.28684  1.03%  3857 38780s
 165412 93057   30.56620   67  702   30.60160   30.28721  1.03%  3859 39062s
 166366 93823   30.30823   55  871   30.60160   30.28765  1.03%  3860 39348s
 167444 94542     cutoff   57        30.60160   30.28805  1.02%  3860 39635s
 168407 95354   30.52518   77  625   30.60160   30.28827  1.02%  3859 39902s
 169426 96143   30.54119  150  311   30.60160   30.28854  1.02%  3859 40170s
H169675 96143                      30.6016043   30.28860  1.02%  3859 40170s
 170561 96675   30.59338   57  909   30.60160   30.28877  1.02%  3855 40445s
 171301 97328   30.58526  105  332   30.60160   30.28888  1.02%  3860 40706s
 172297 98134   30.44904   93  456   30.60160   30.28892  1.02%  3860 40969s
 173417 98660   30.47598   64  863   30.60160   30.28905  1.02%  3858 41230s
 174472 99359   30.43938   69  639   30.60160   30.28957  1.02%  3855 41492s
 175499 100015   30.36983   51  925   30.60160   30.28979  1.02%  3855 41757s
 176606 100192   30.58671   60  776   30.60160   30.28997  1.02%  3852 42009s
H176681 100021                     30.6010771   30.28997  1.02%  3852 42009s
 176883 100513   30.38313   49  834   30.60108   30.28999  1.02%  3851 42289s
 177736 100951   30.37215   45 1149   30.60108   30.29049  1.01%  3853 42570s
 178391 101415     cutoff   61        30.60108   30.29093  1.01%  3858 42816s
H178531 101415                     30.6010771   30.29098  1.01%  3859 42816s
 179131 102249   30.55911   49  884   30.60108   30.29100  1.01%  3860 43137s
 180437 102340   30.56203   83  790   30.60108   30.29146  1.01%  3858 43200s

Cutting planes:
  Gomory: 864
  Lift-and-project: 18
  Cover: 3
  Projected implied bound: 2
  MIR: 129
  StrongCG: 5
  Flow cover: 136
  Inf proof: 7
  Zero half: 5
  Relax-and-lift: 2

Explored 180638 nodes (696979574 simplex iterations) in 43200.12 seconds (45403.57 work units)
Thread count was 8 (of 8 available processors)

Solution count 5: 30.6011 30.6016 30.6016 ... 30.602

Time limit reached
Best objective 3.060107707818e+01, best bound 3.029157436085e+01, gap 1.0114%
```

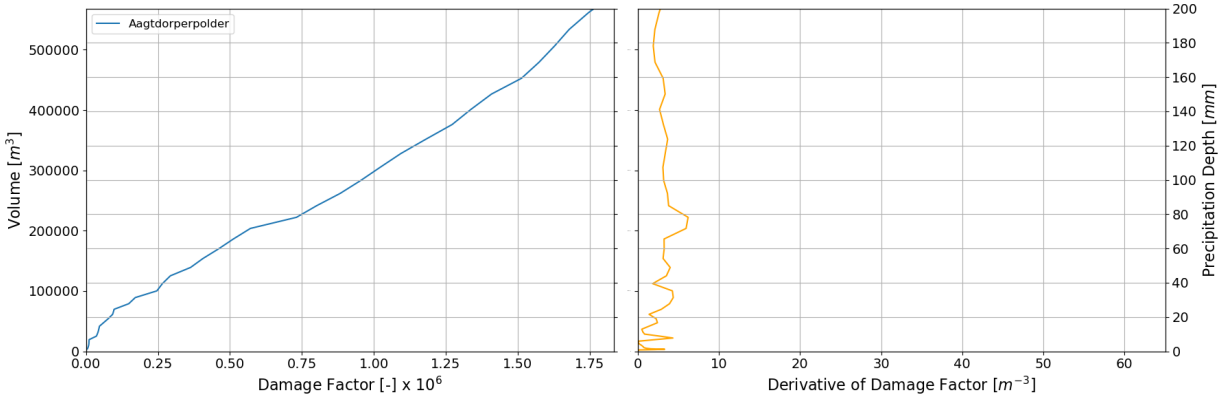
L

Polder Characteristics

Table L.1: Aagterdorperpolder characteristics

Aagterdorperpolder			Type 1	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	12.7		284	144
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	115.7	1.10	0	0
Optimized	115.7	1.10		





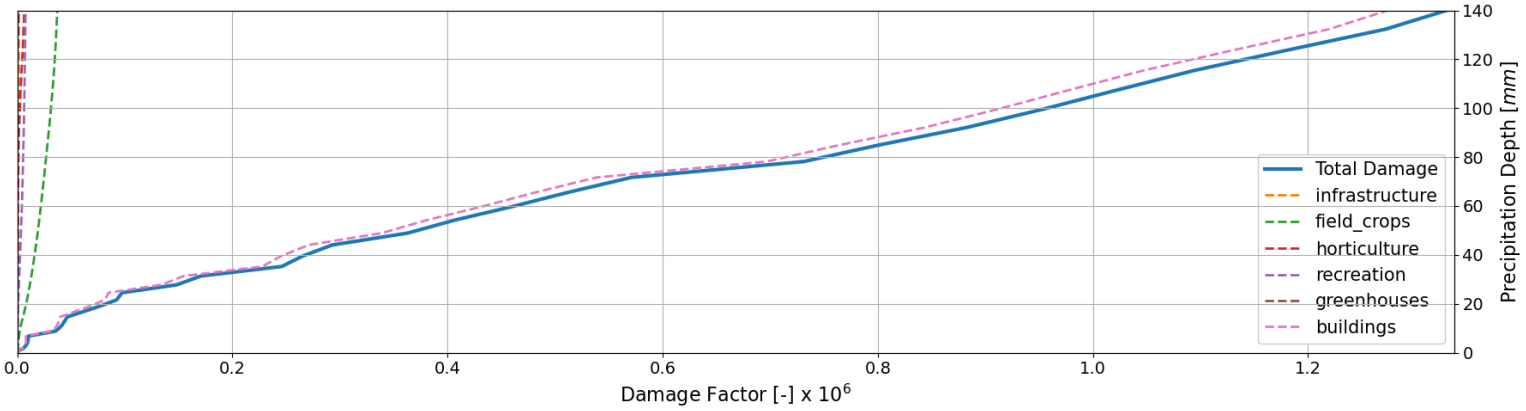
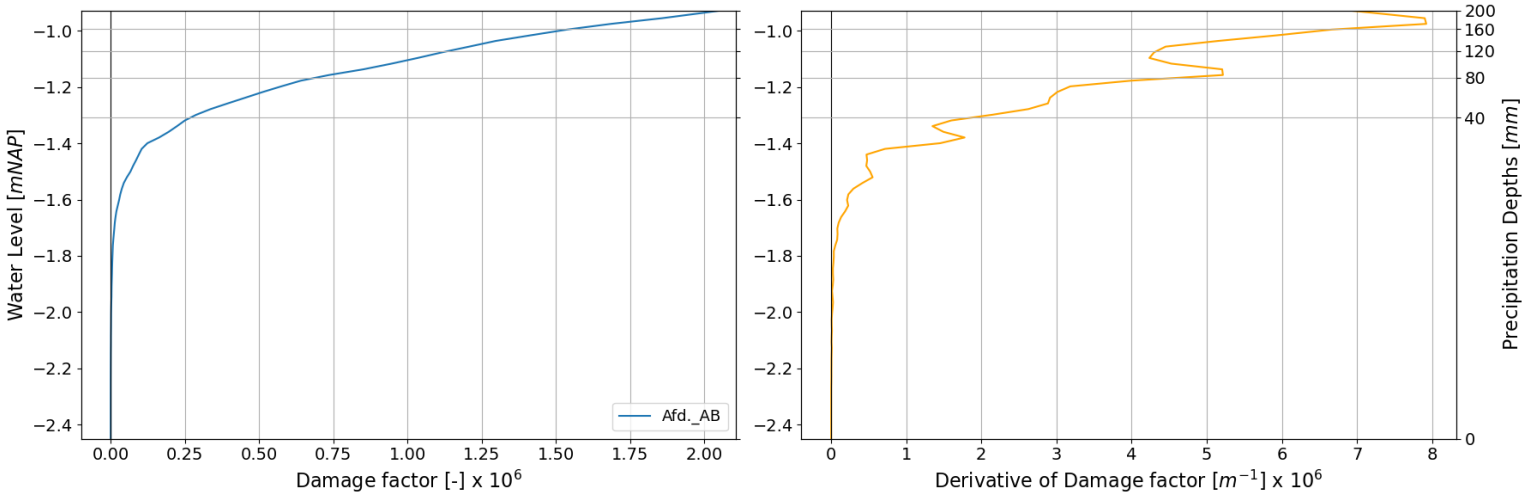
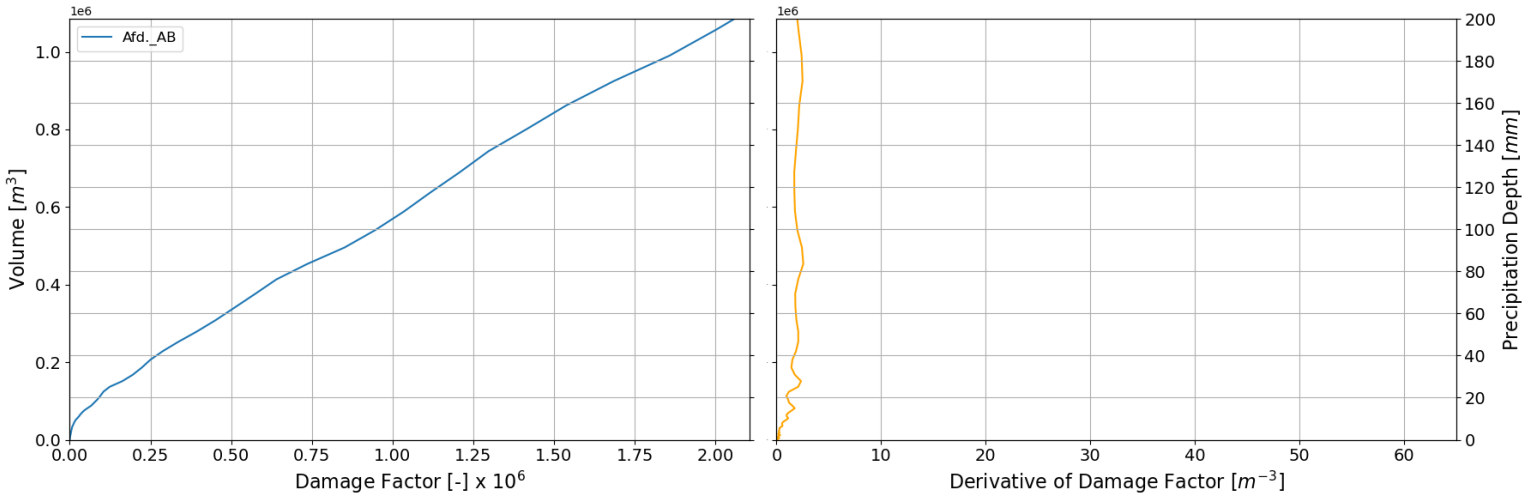


Table L.2: Afd. AB characteristics

Afd. AB			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	17.2		543	122
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	86	0.77	0	0
Optimized	86	0.77		





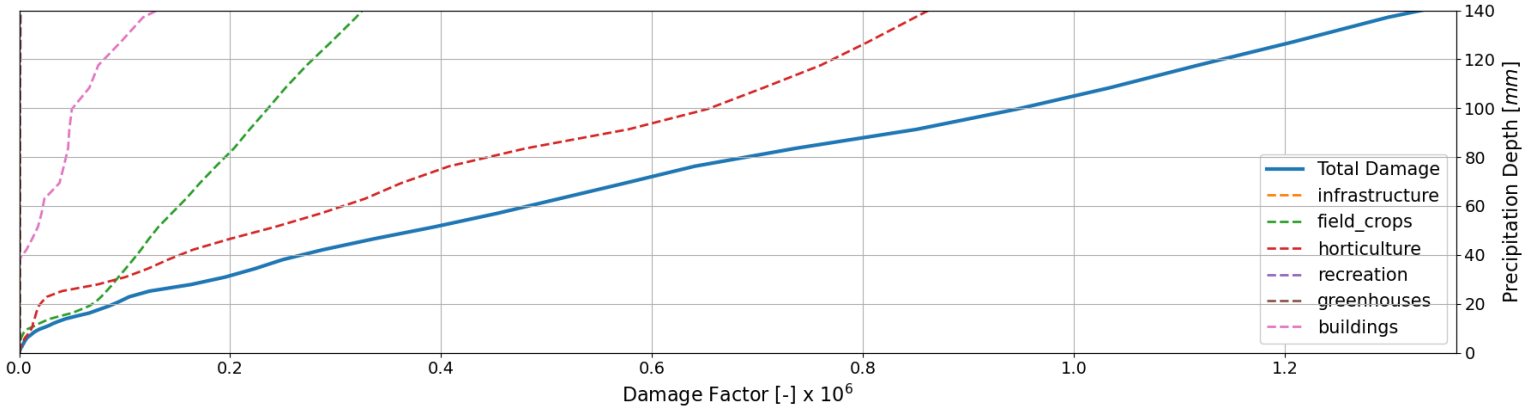
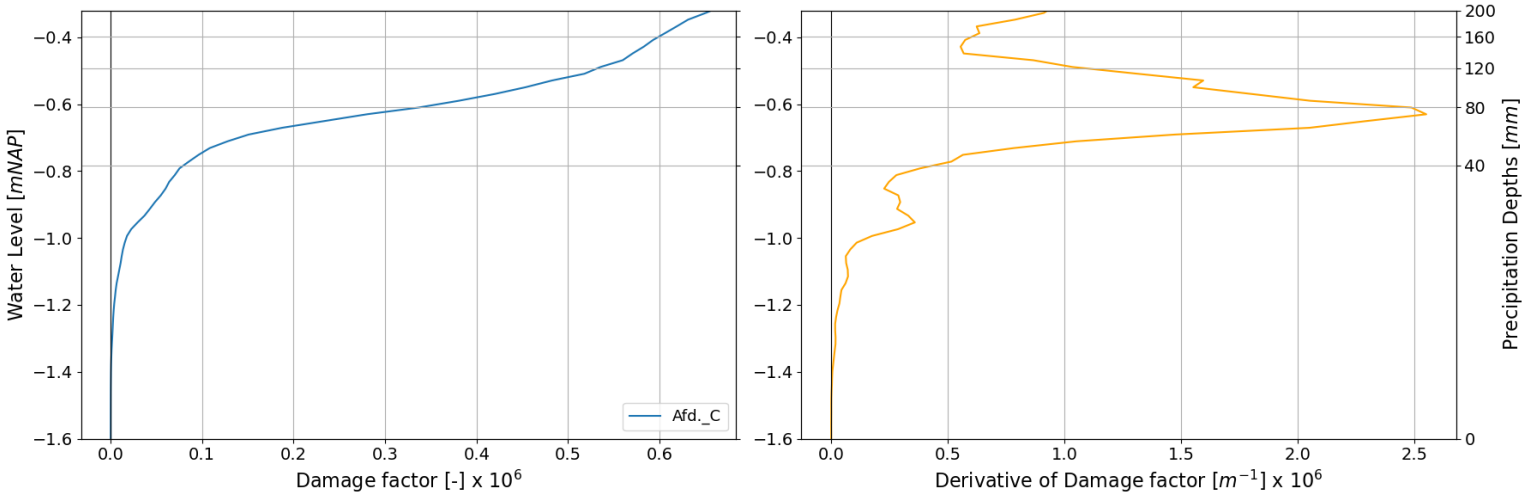
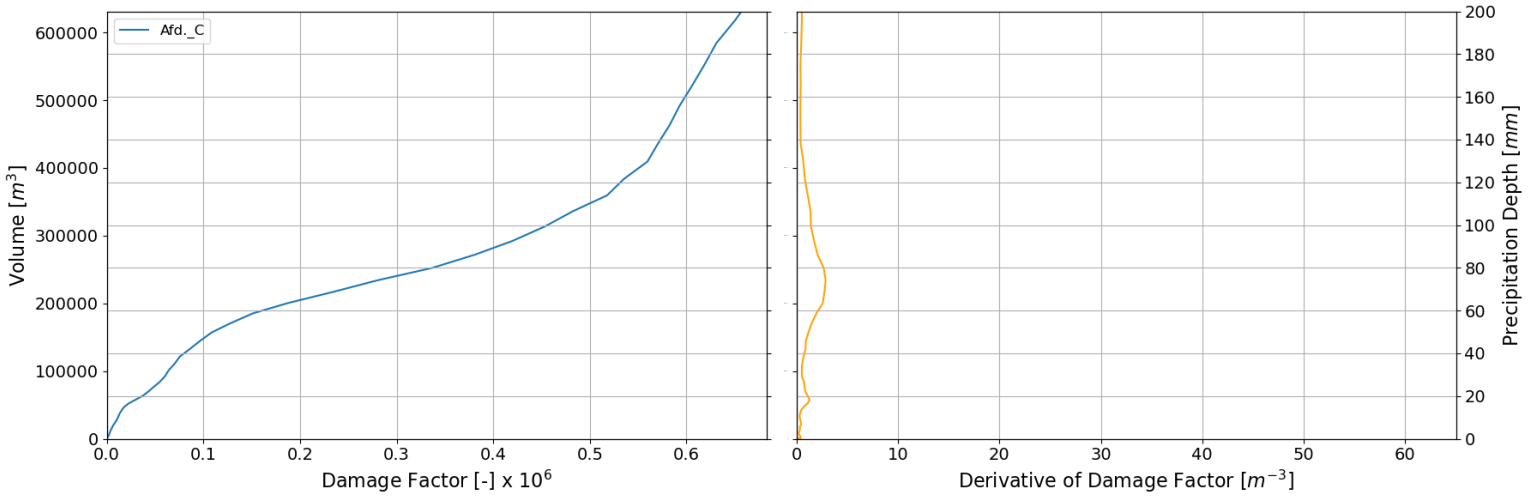


Table L.3: Afd. C characteristics

Afd. C			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
50	14.0		316	124
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	96	0.44	0	0
Optimized	96	0.44		





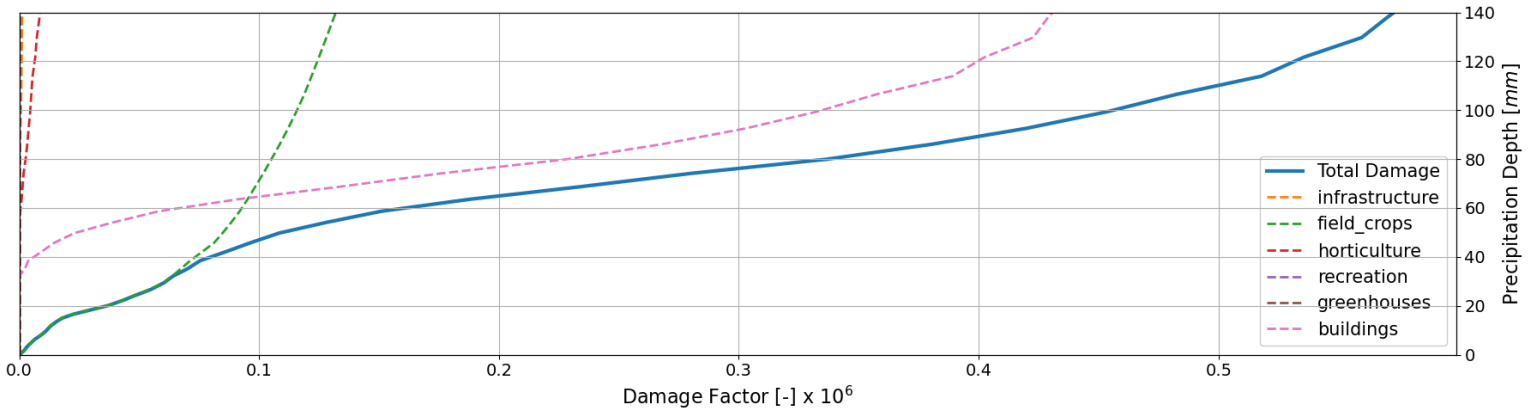
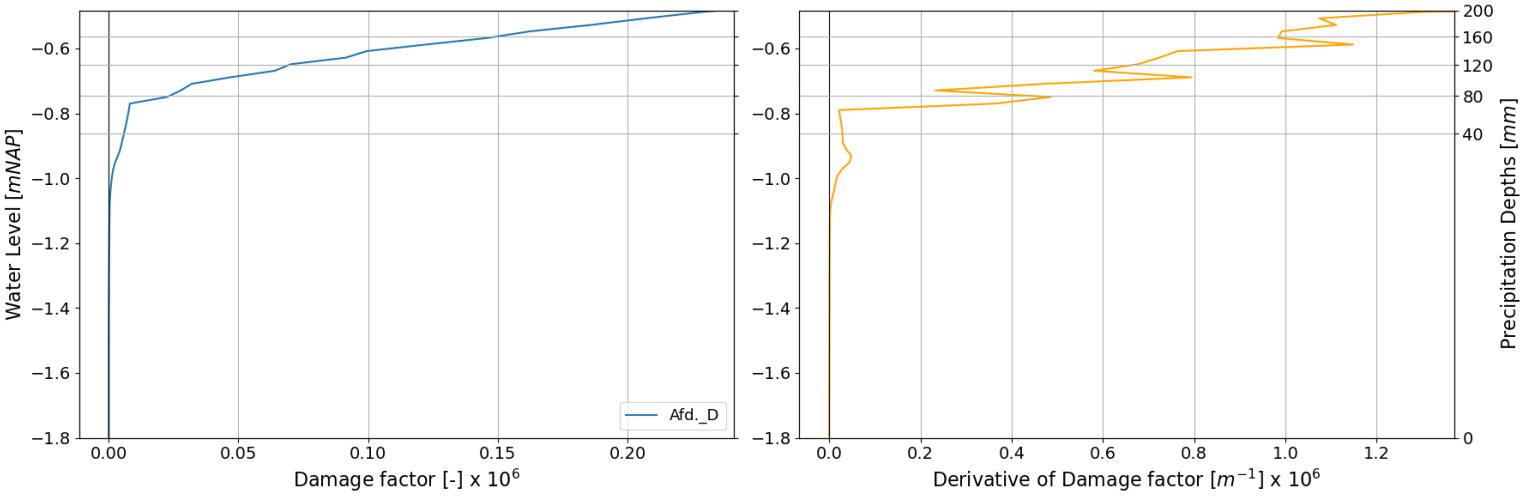
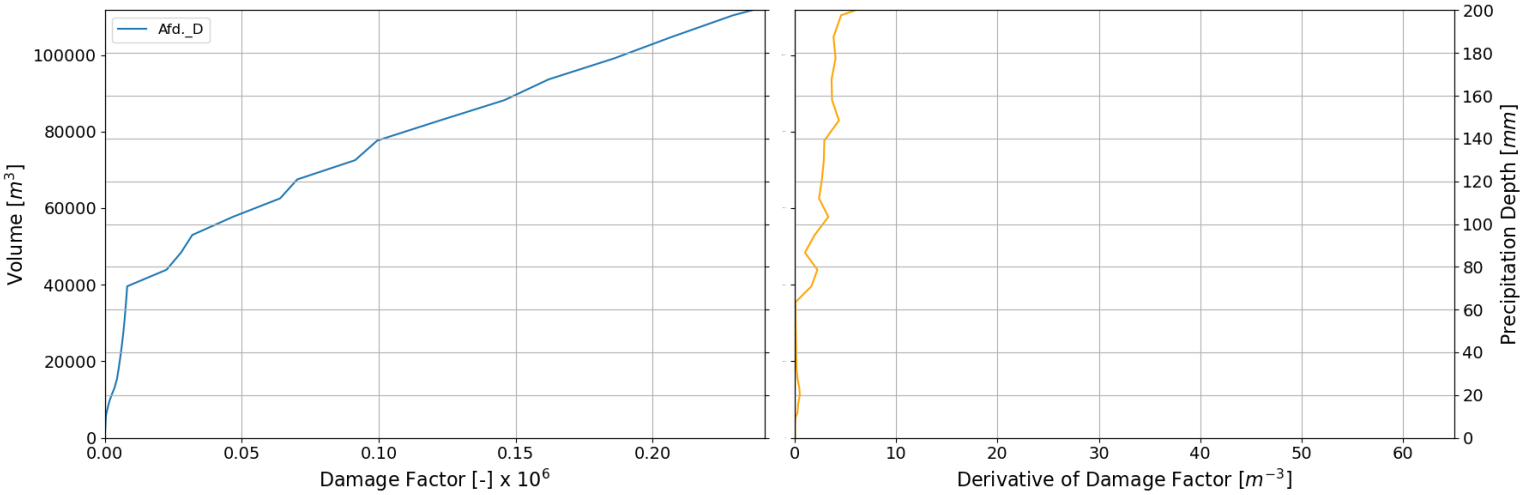


Table L.4: Afd. D characteristics

Afd. D			Type 1	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
65 mm	48.9		56	134
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	59	0.01	0	0
Optimized	59	0.01		





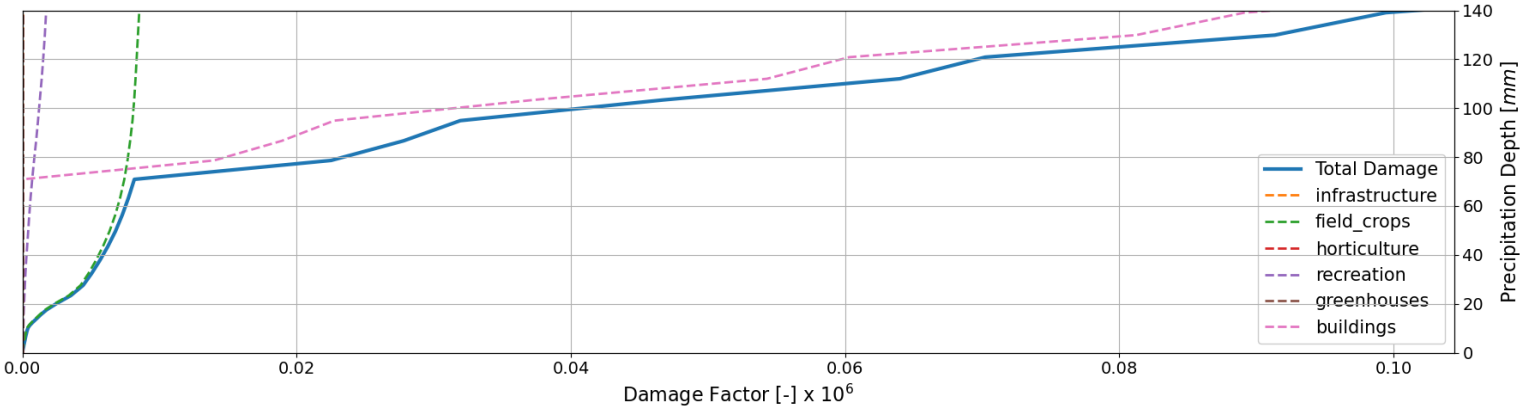
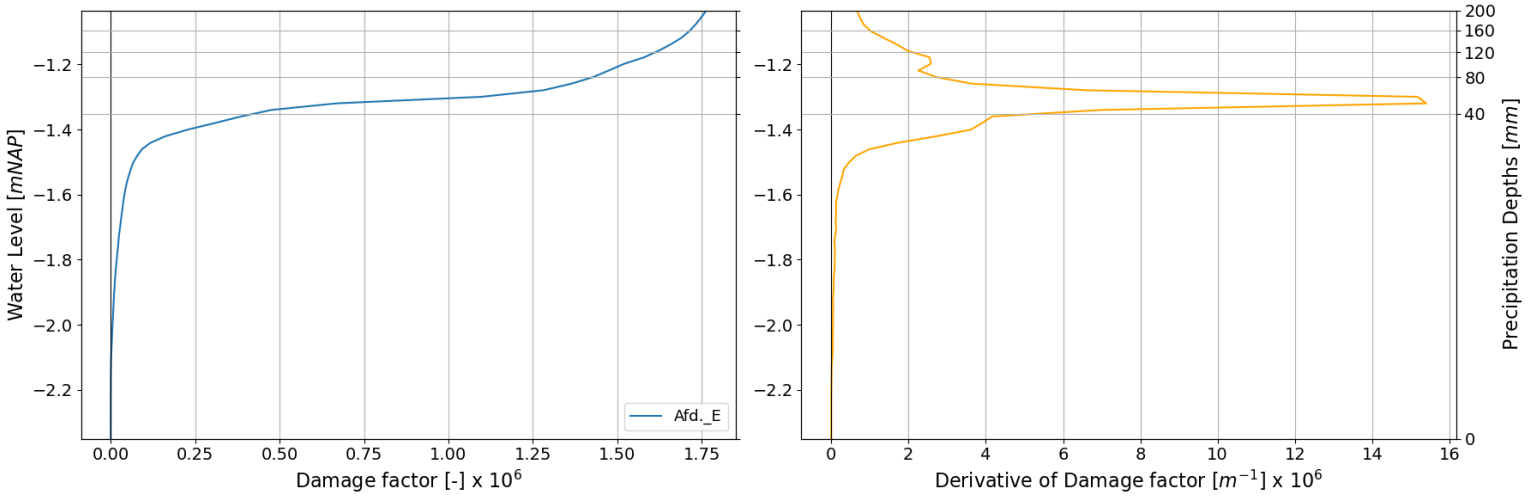
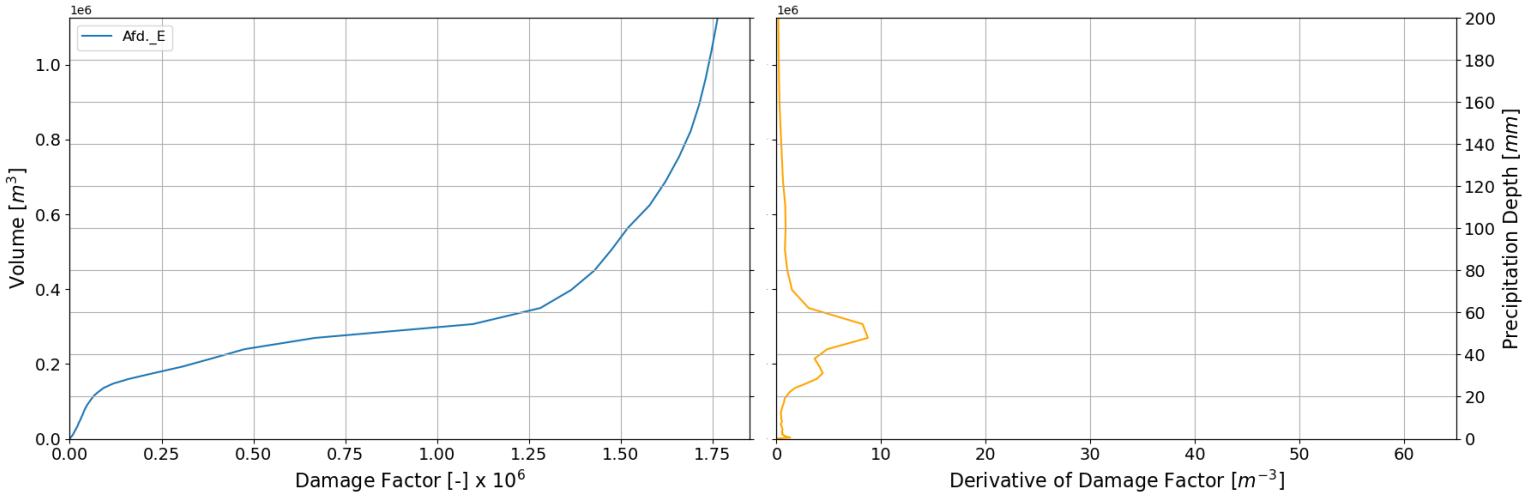


Table L.5: Afd. E characteristics

Afd. E			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	13.3		563	103
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	76	1.4	0	0
Optimized	76	1.4		





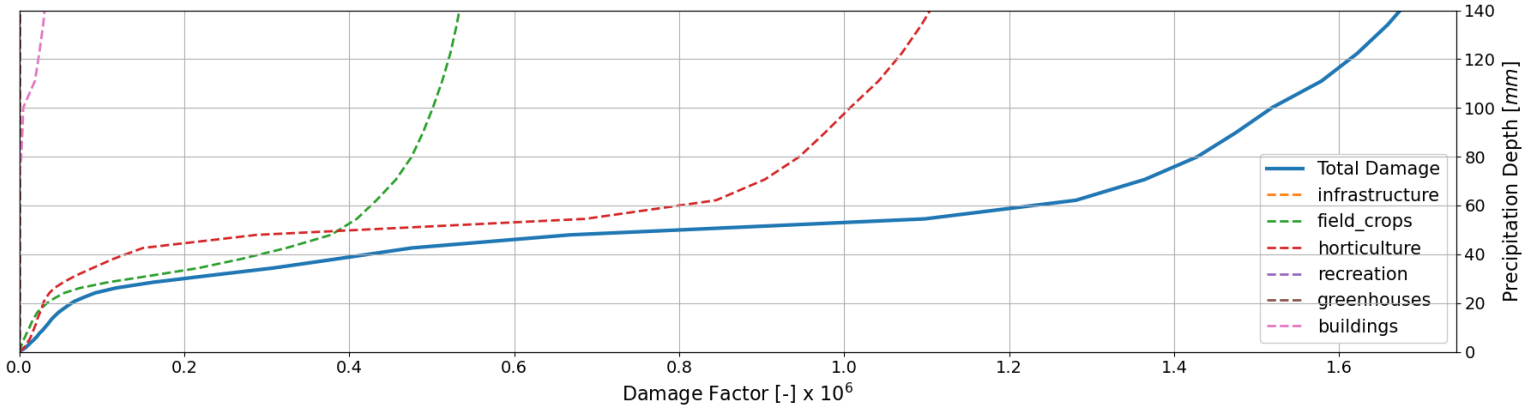
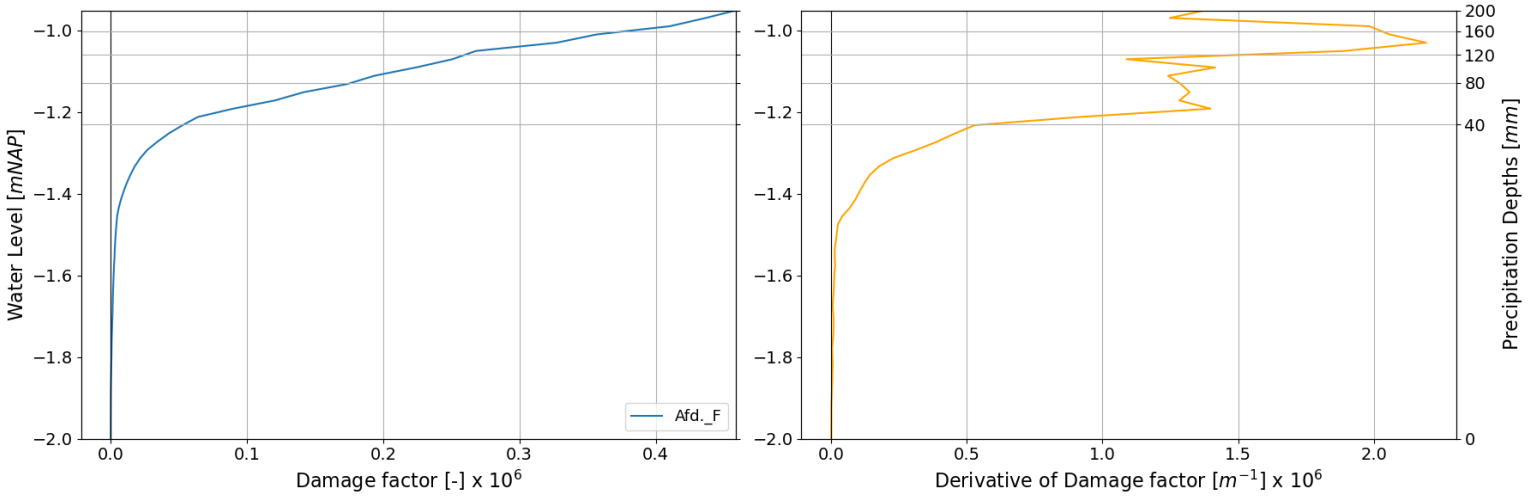
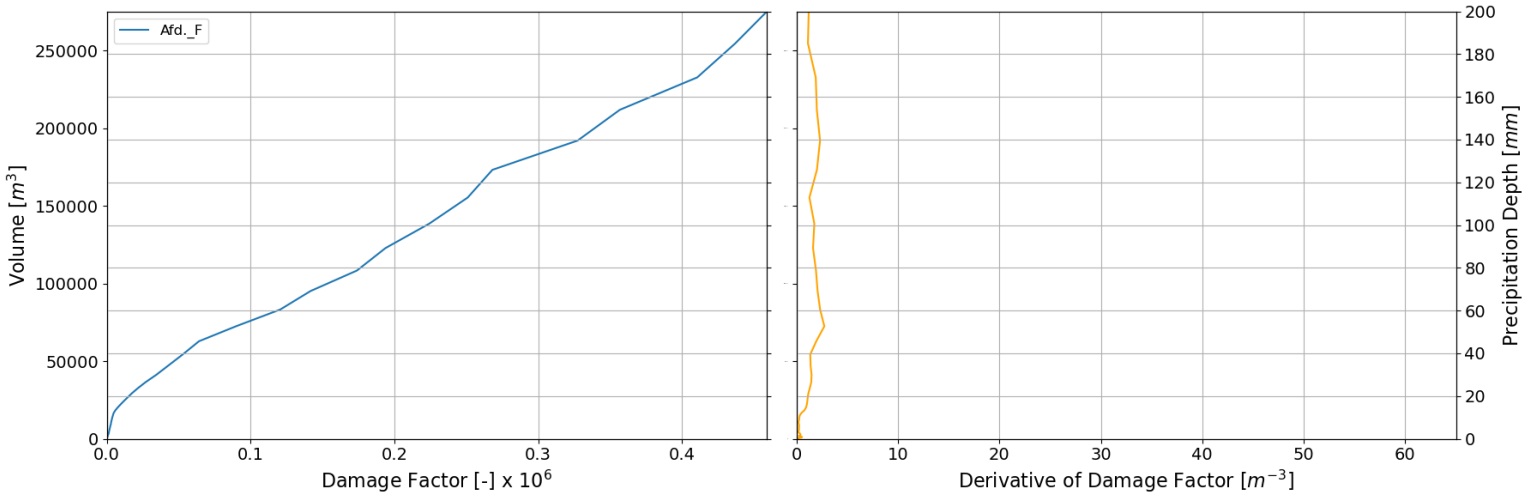


Table L.6: Afd. F characteristics

Afd. F			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	19.8		138	128
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	88	0.19	0	0
Optimized	88	0.19		





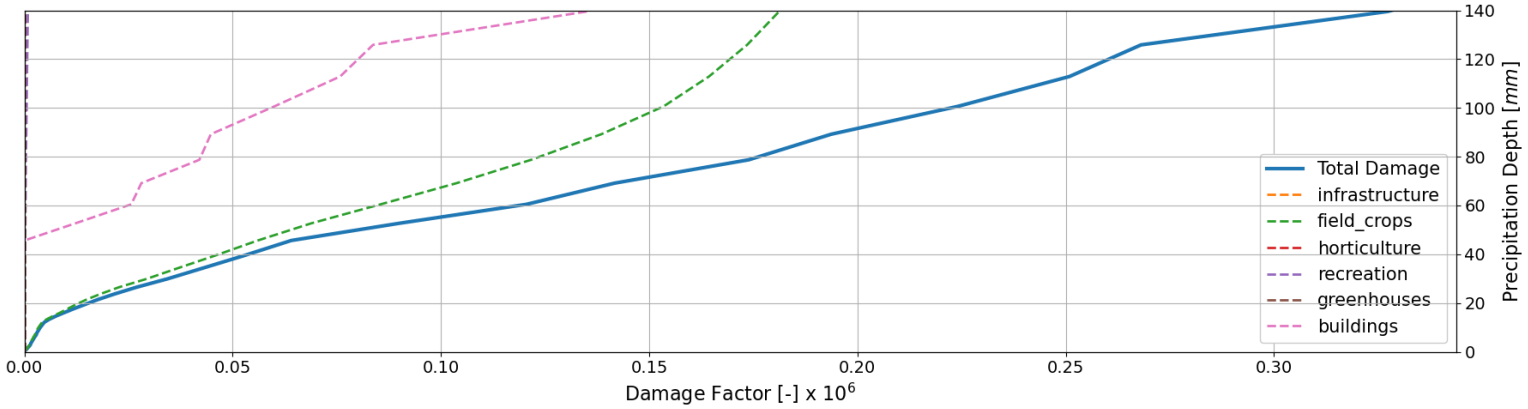
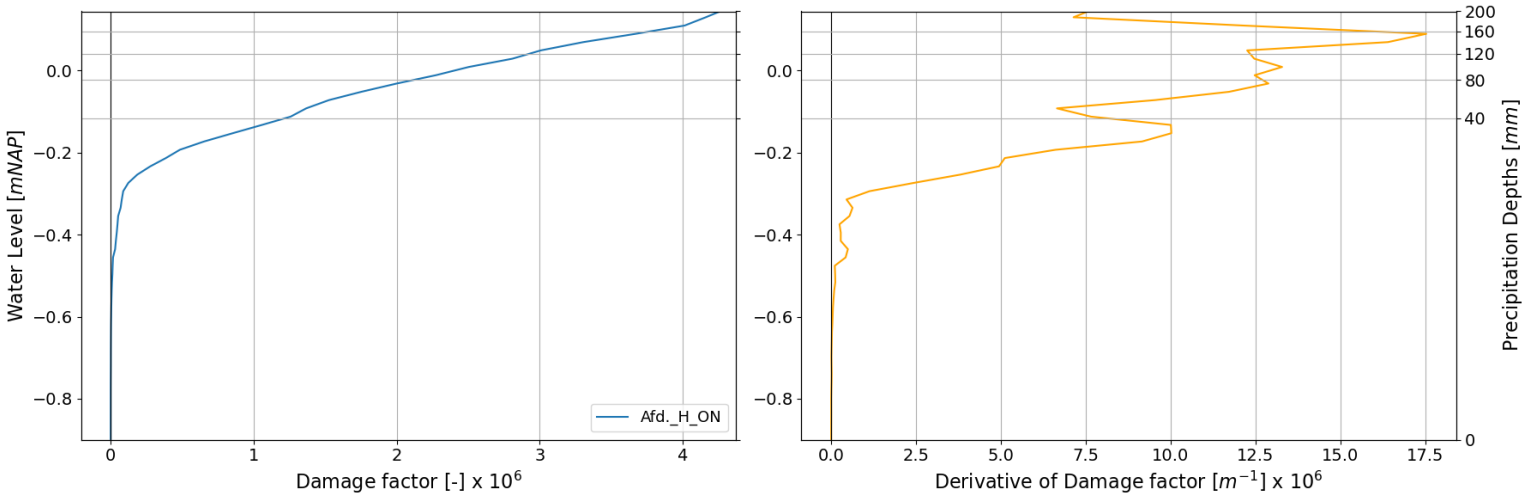
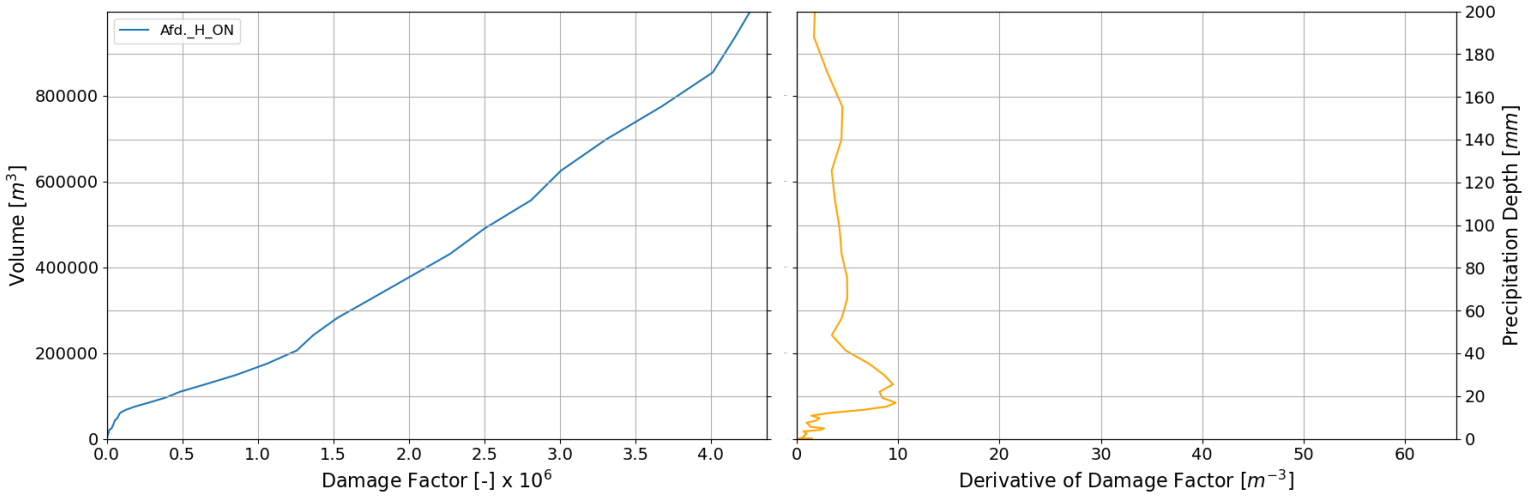


Table L.7: Afd. H-ON characteristics

Afd. H-ON			Type 1	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	20.5		498	124
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	82.0	2.16	0.049	12960
Optimized	80.0	2.11		





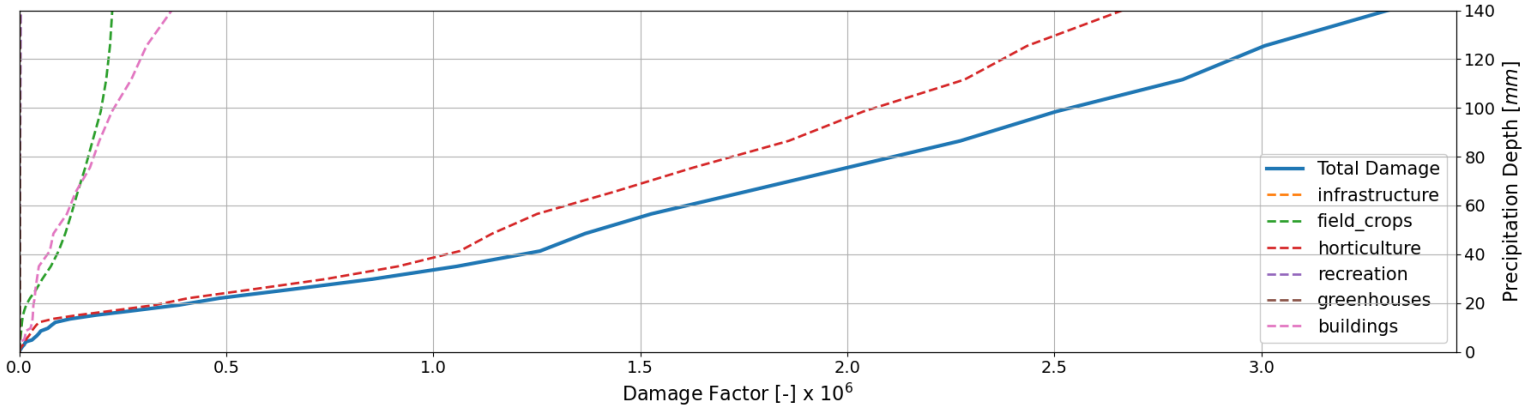
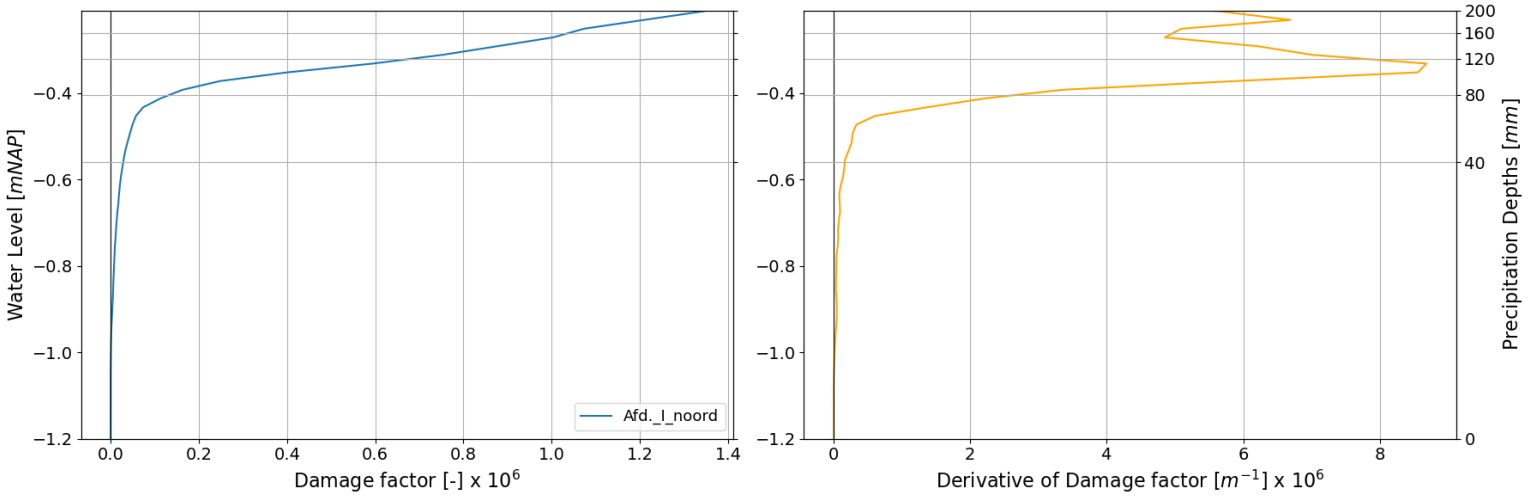
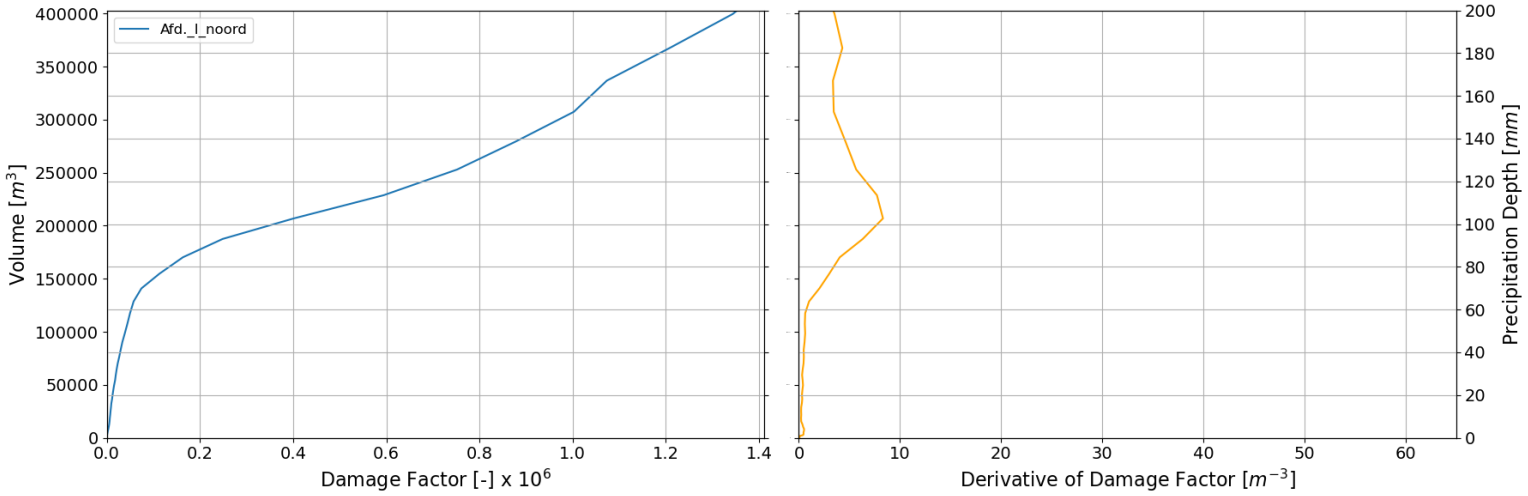


Table L.8: Afd. I-noord characteristics

Afd. I-noord			Type 1	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
70	19.2		202	151
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	113.2	0.59	0.086	12960
Optimized	108.4	0.50		





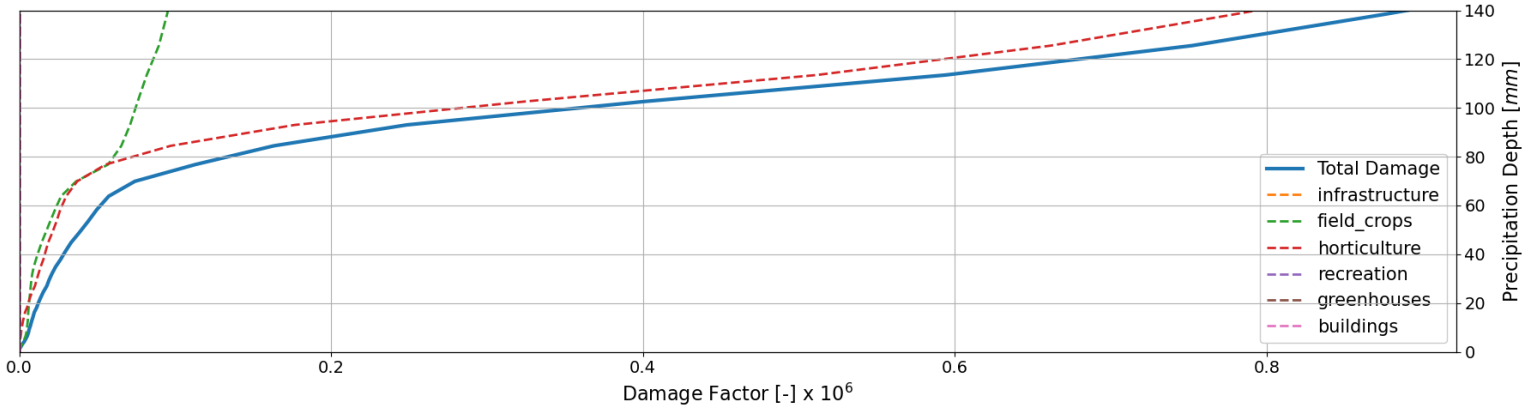
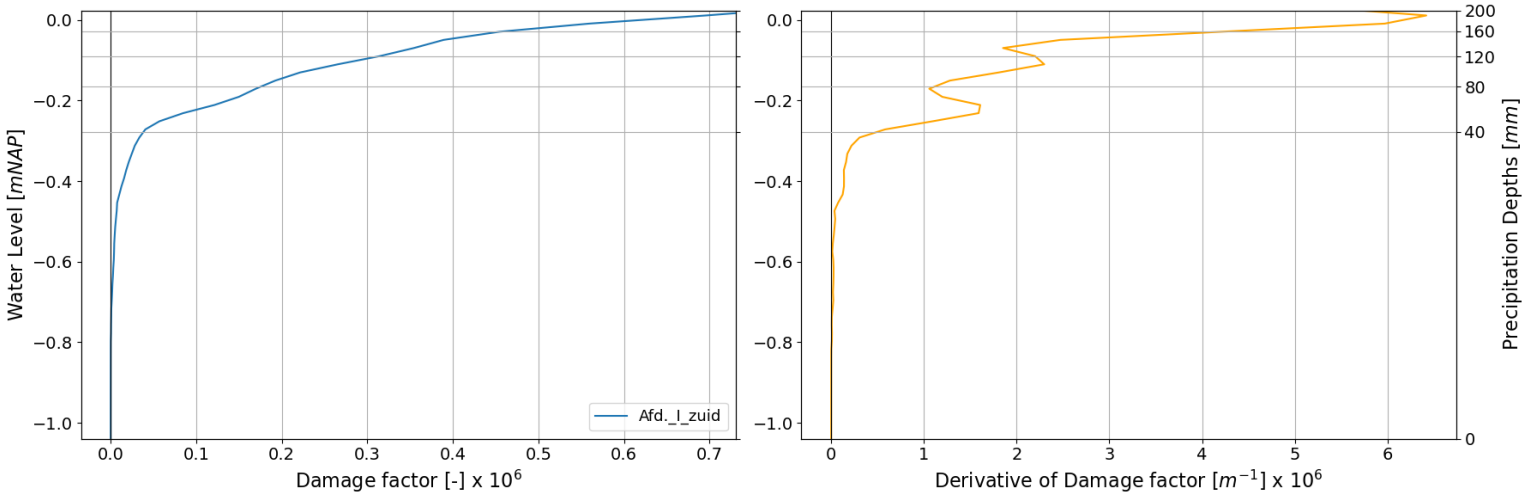
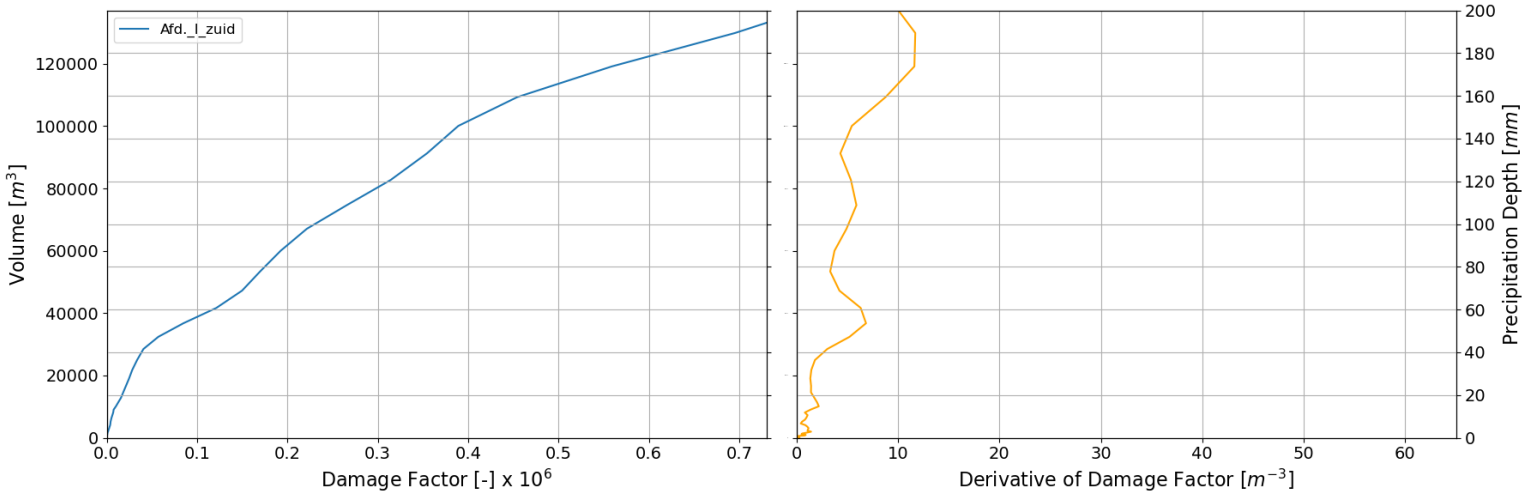


Table L.9: Afd. I-zuid characteristics

Afd. I-zuid			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	16.7		69	137
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	102	0.24	0	0
Optimized	102	0.24		





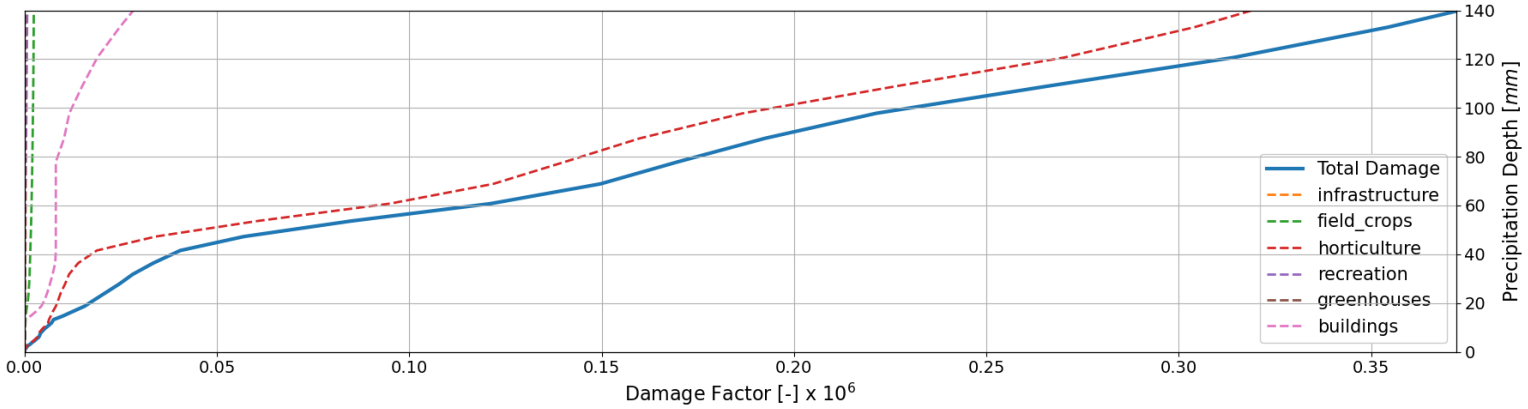
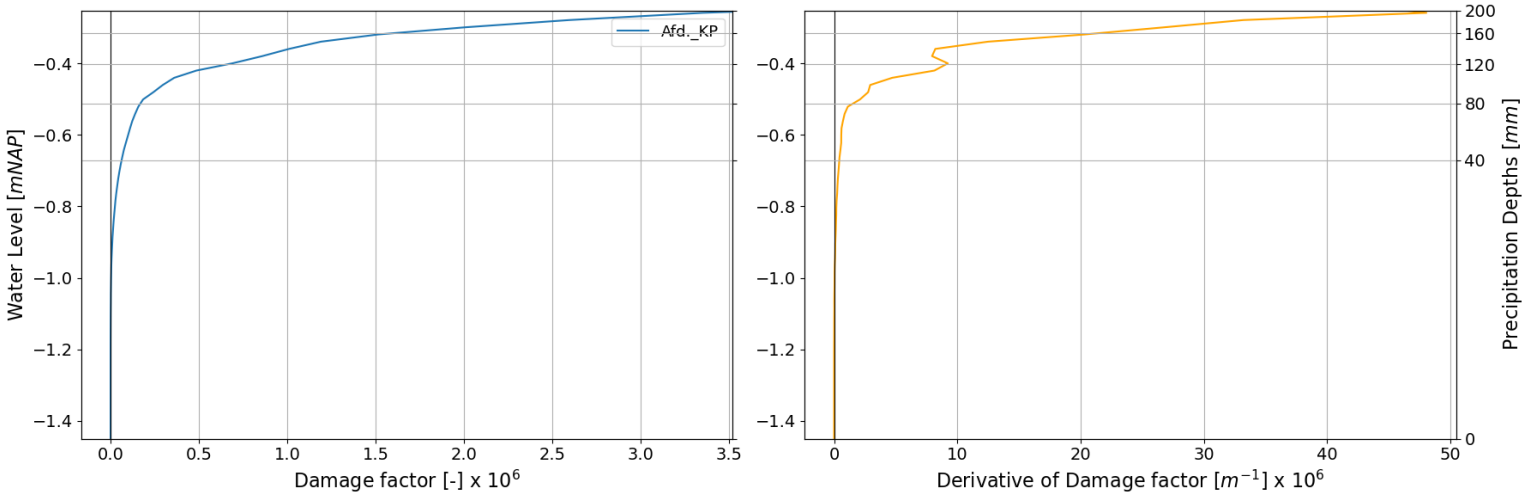
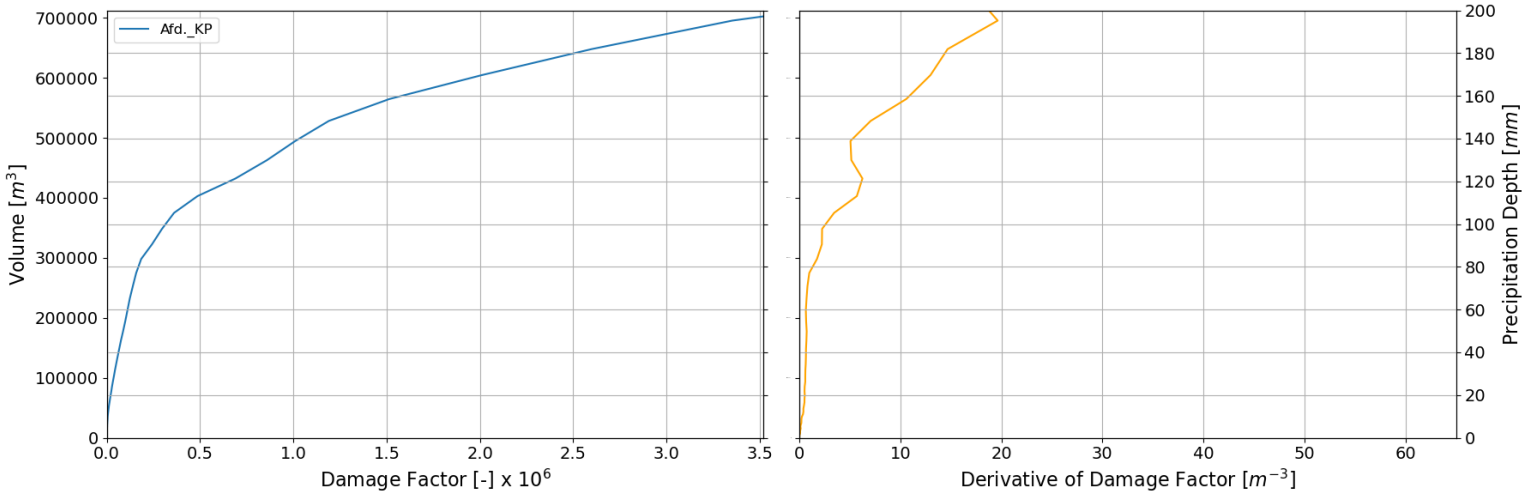


Table L.10: Afd. KP characteristics

Afd. KP			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
80	15.0		356	101
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	71	0.14	0	0
Optimized	71	0.14		





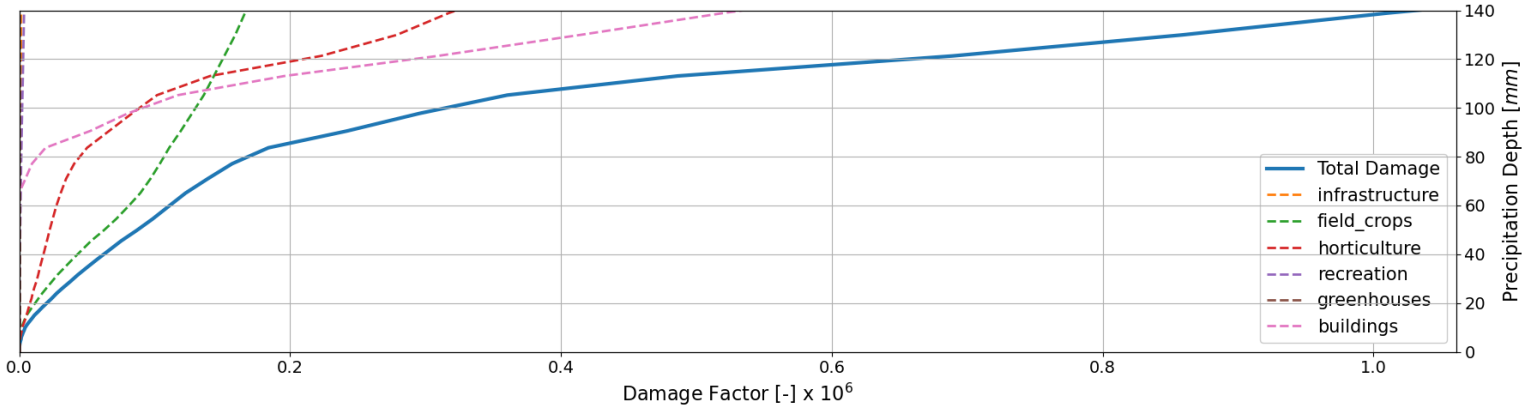
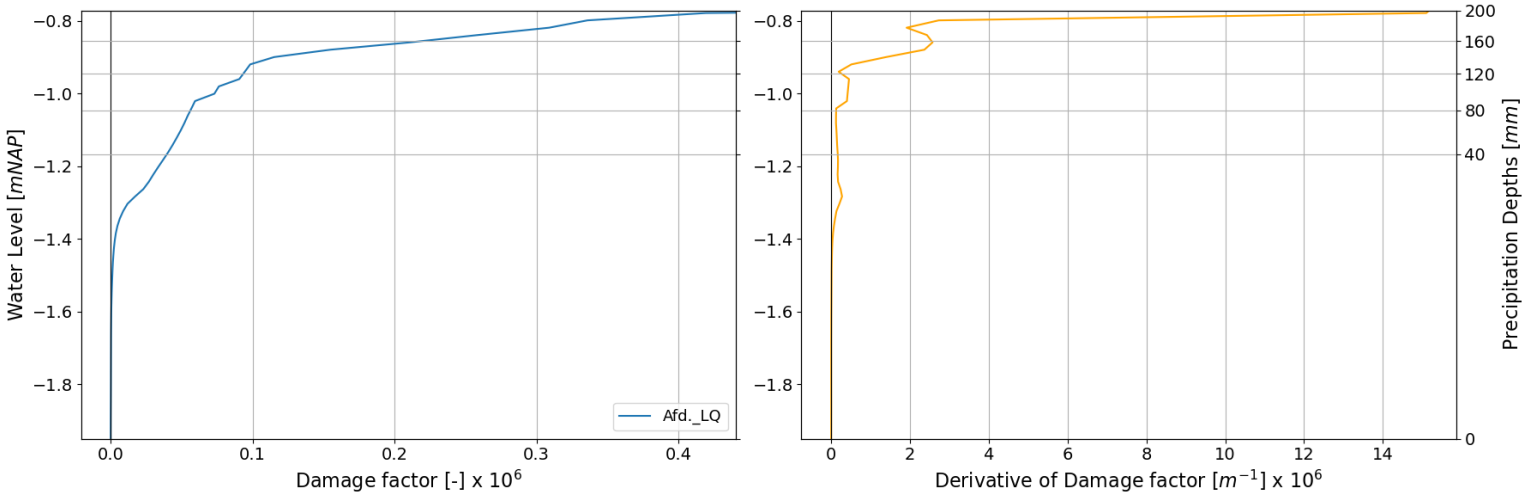
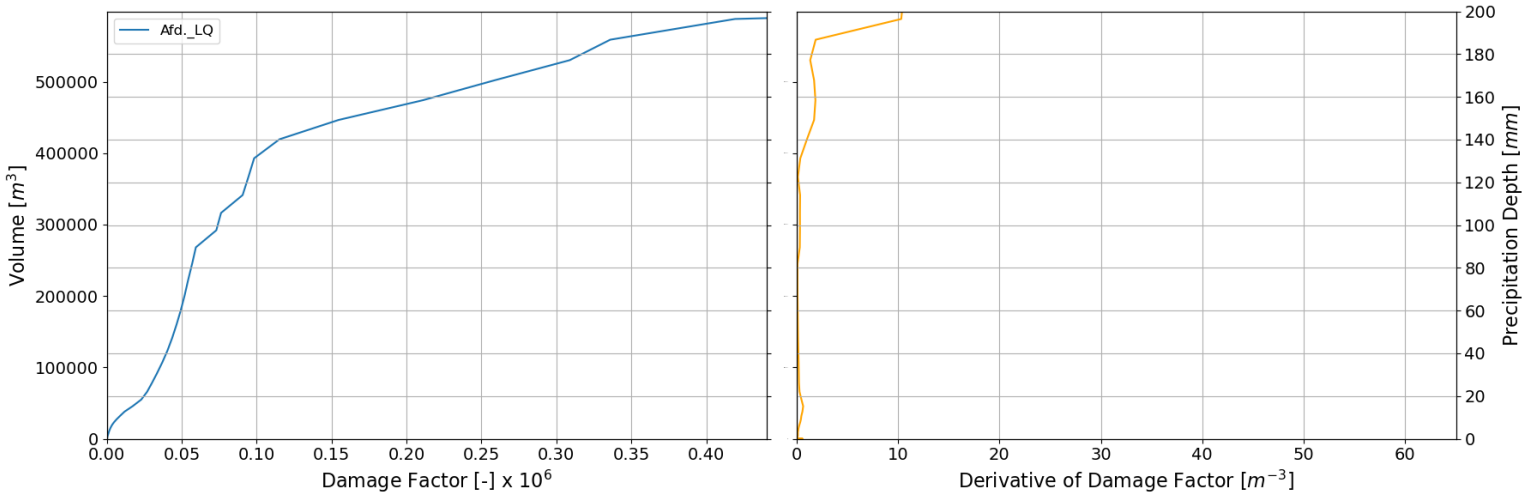


Table L.11: Afd. LQ characteristics

Afd. LQ			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	15.4		299	121
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	114	0.09	0	0
Optimized	114	0.09		





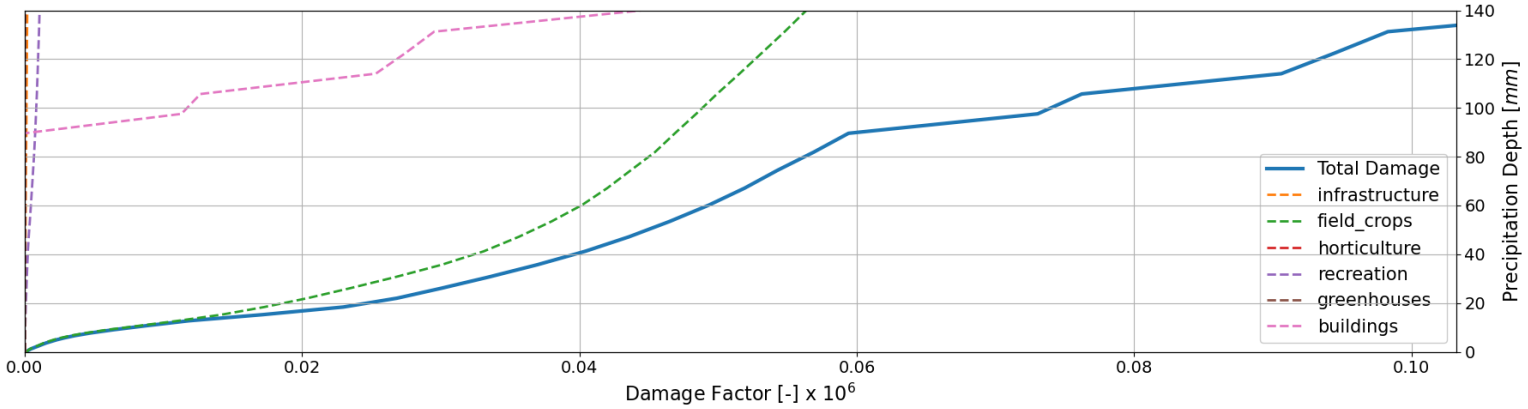
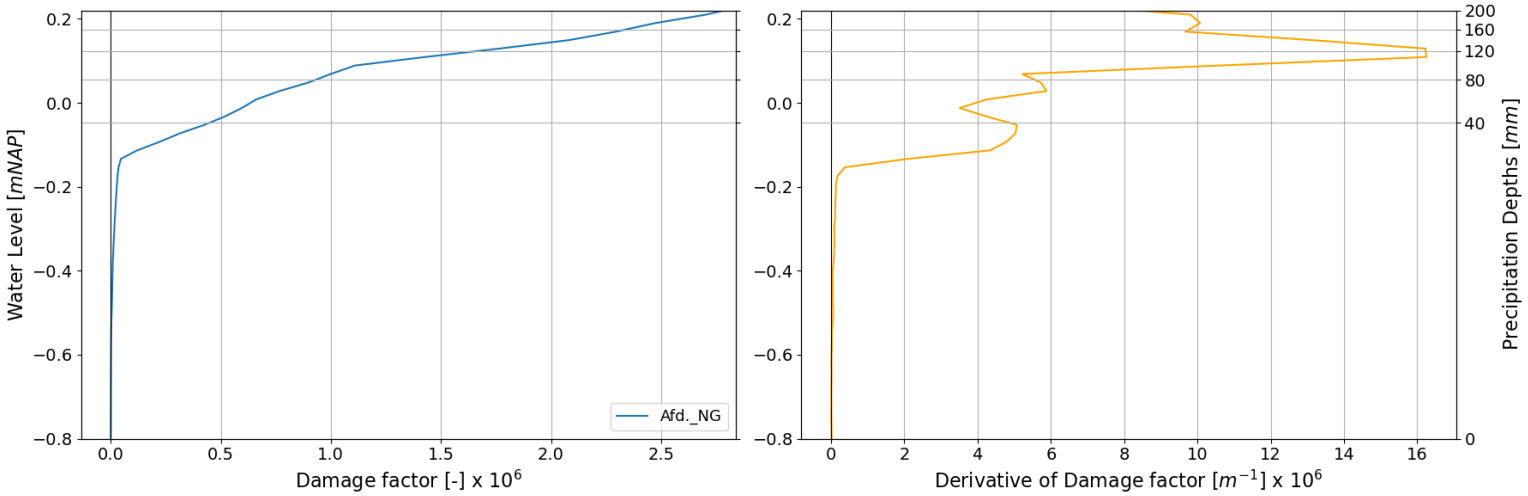
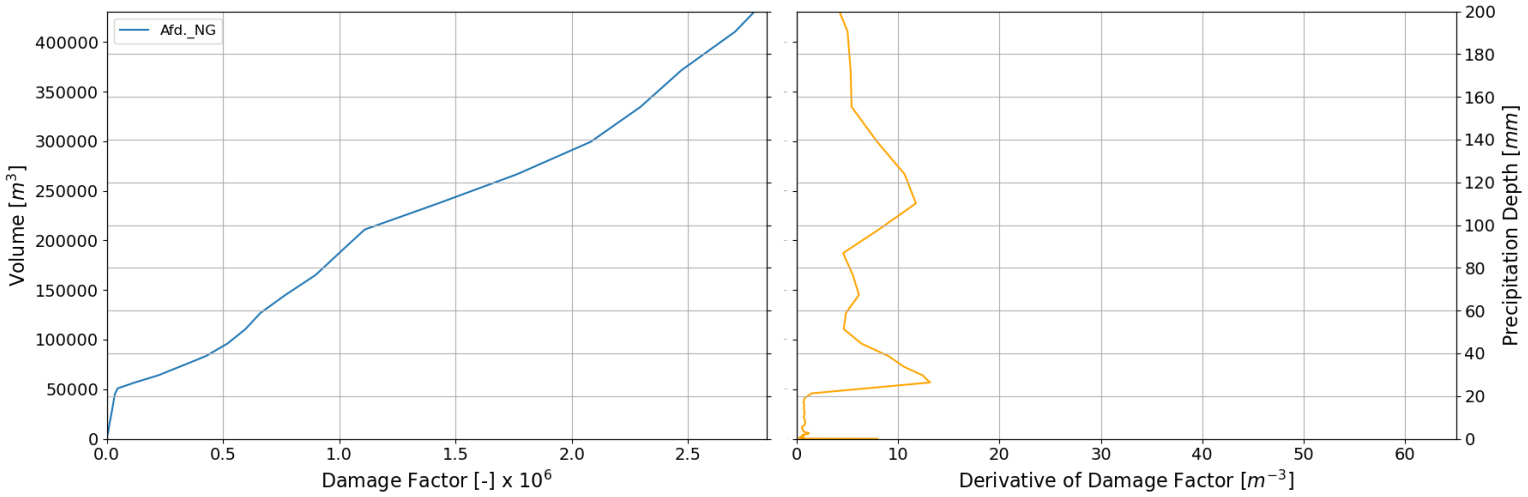


Table L.12: Afd. NG characteristics

Afd. NG			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	18.1		215	123
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	85.1	0.98	0.033	10800
Optimized	81.7	0.95		





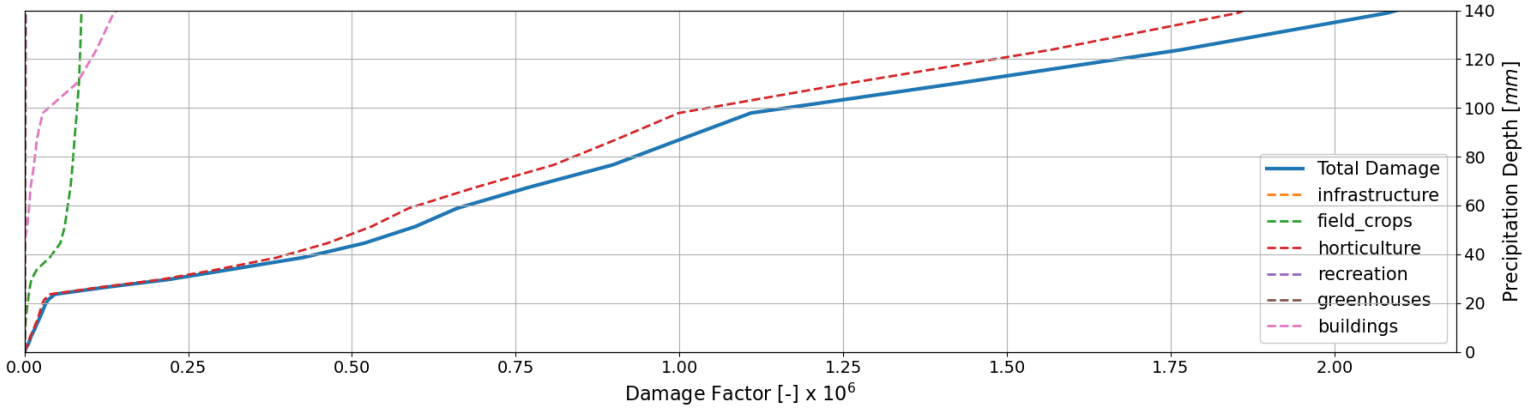
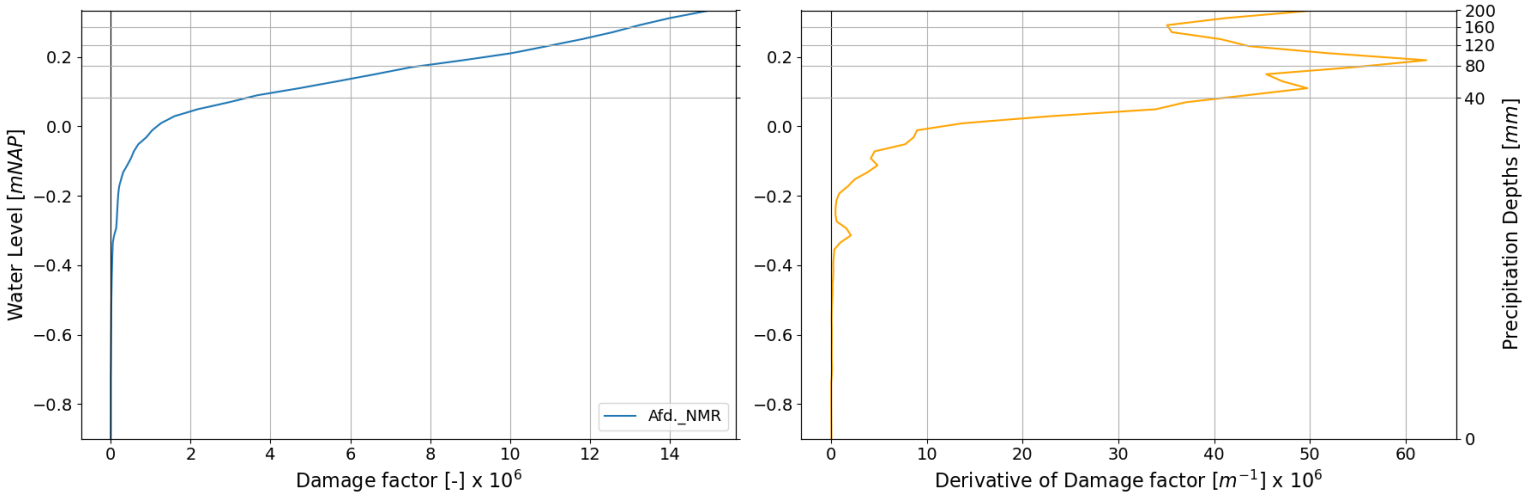
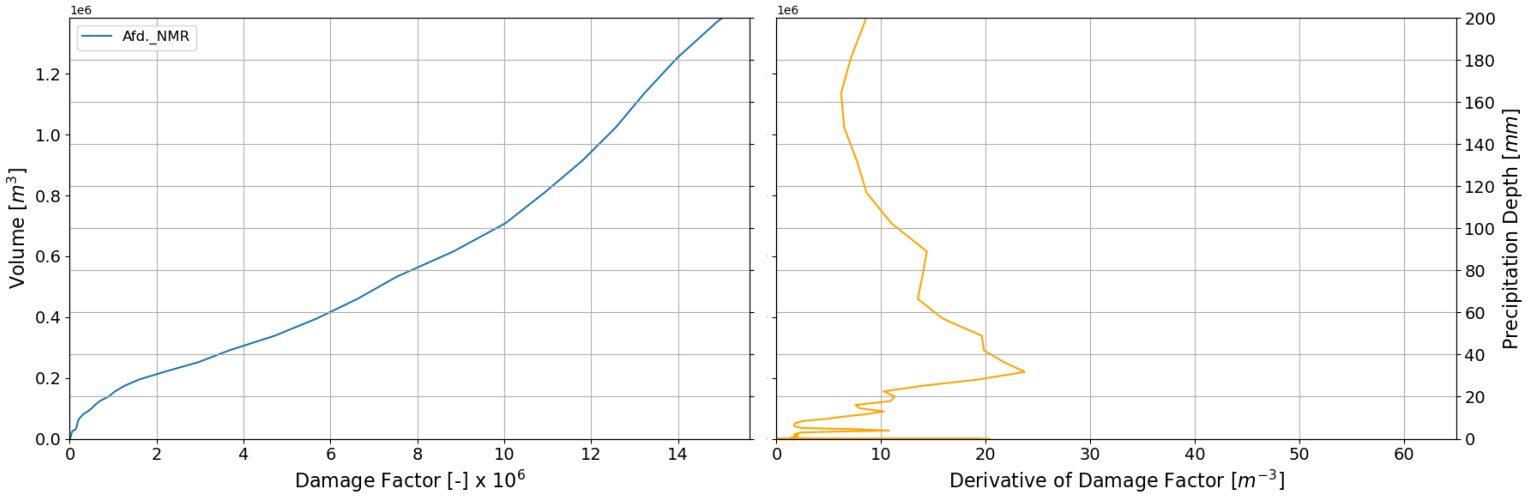


Table L.13: Afd. NMR characteristics

Afd. NMR			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	25.0		692	123
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	70.0	6.92	0.751	74520
Optimized	62.0	6.17		





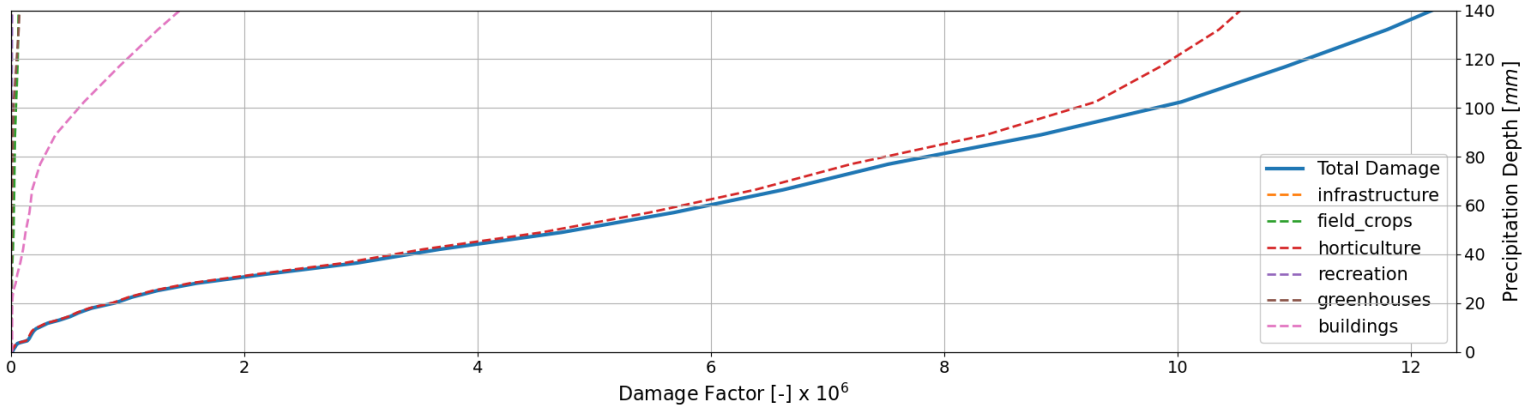
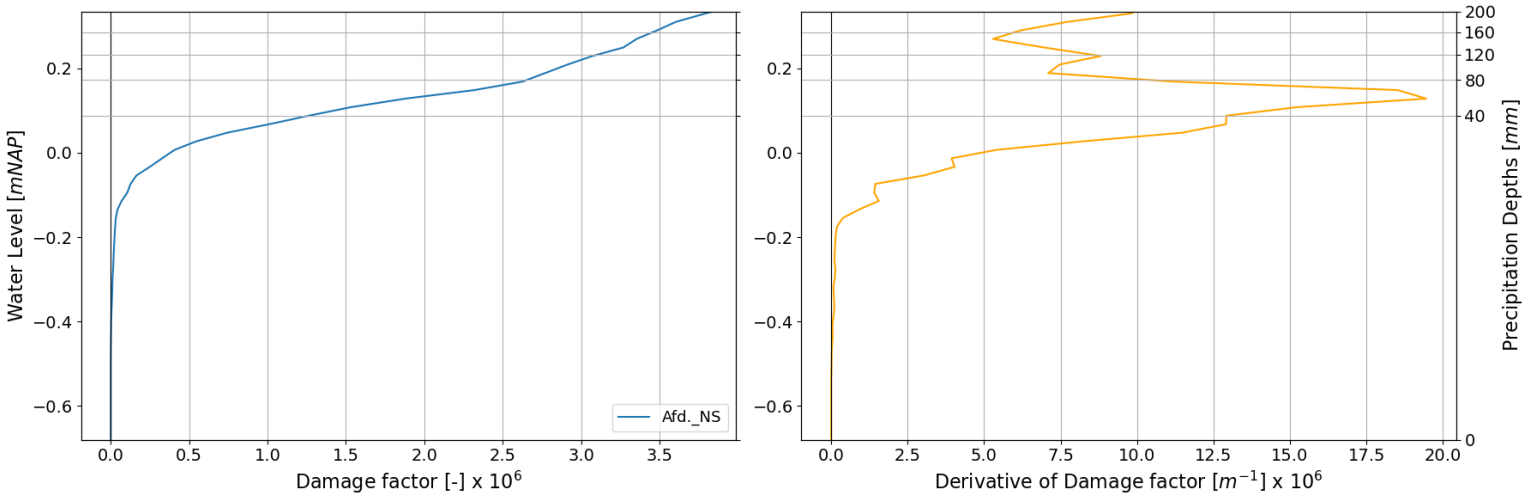
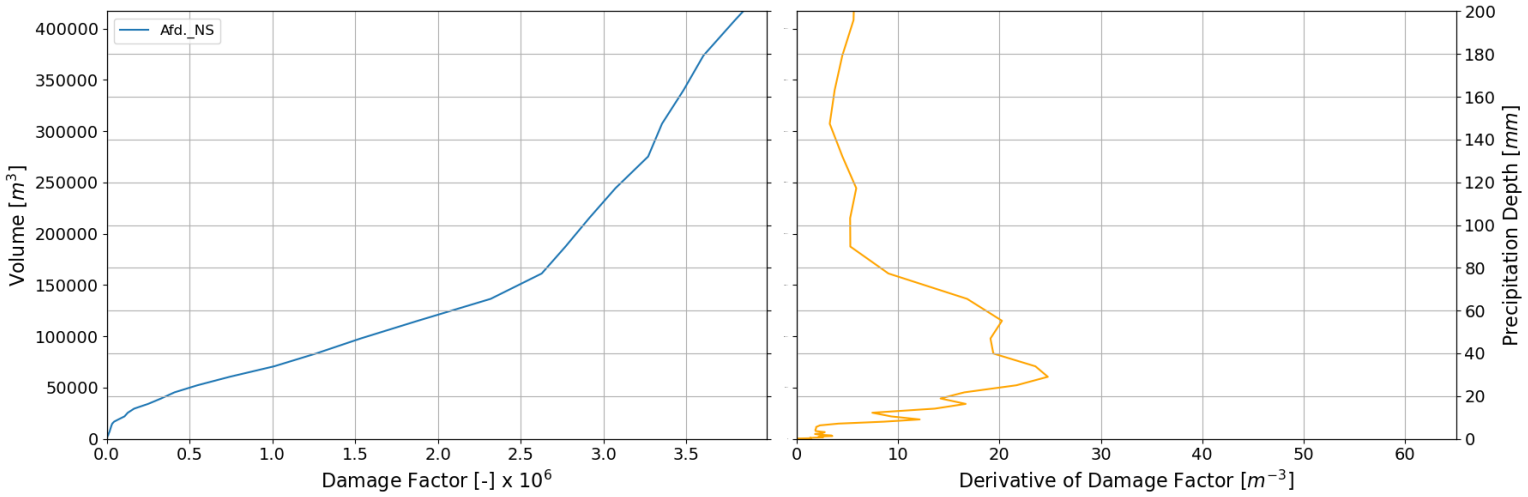


Table L.14: Afd. NS characteristics

Afd. NS			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	16.6		208	108
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	75.2	2.57	0.927	61560
Optimized	49.6	1.64		





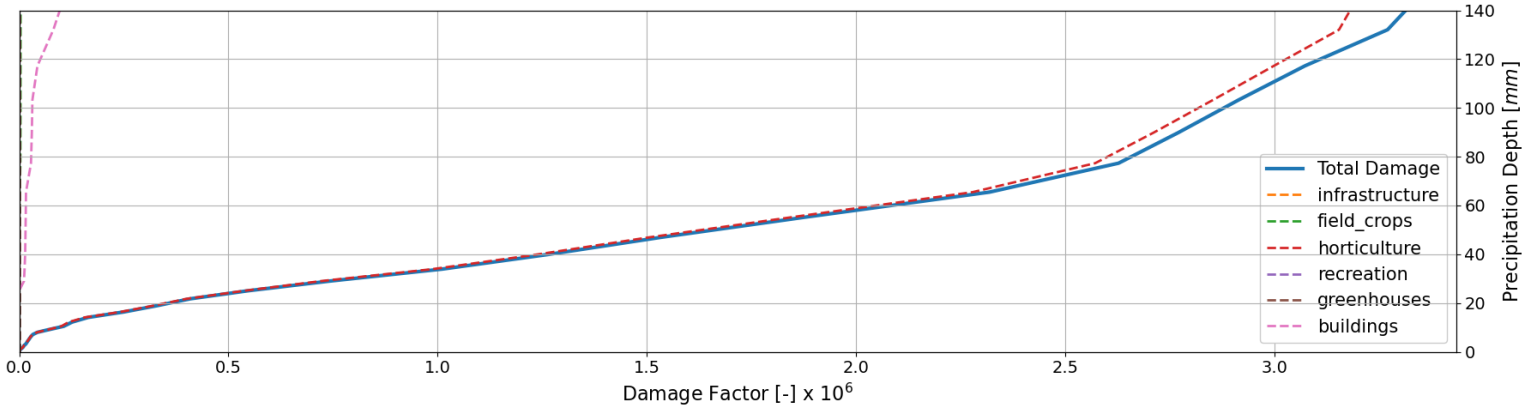
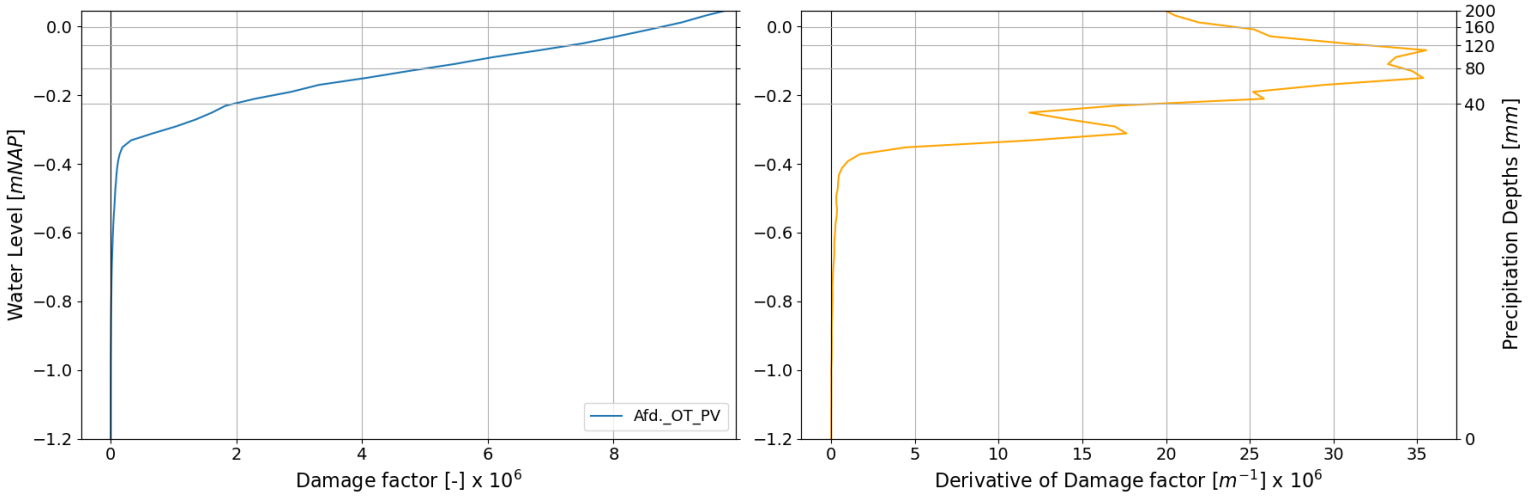
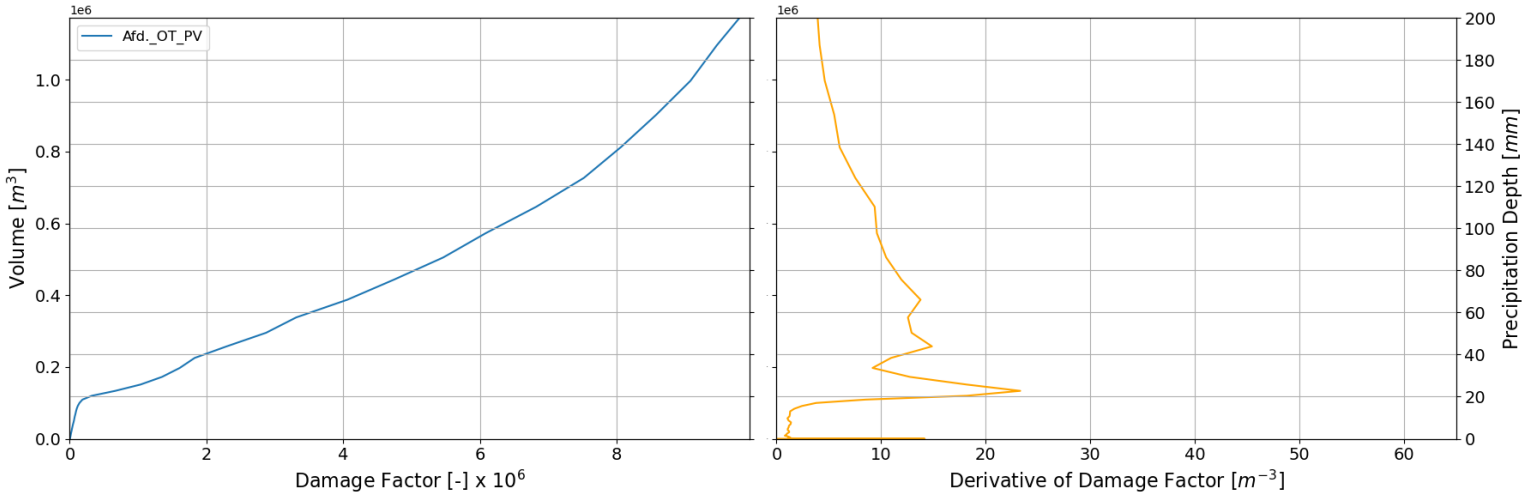


Table L.15: Afd. OT-PV characteristics

Afd. OT-PV			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	14.5		586	115
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	86.7	5.49	2.226	190080
Optimized	56.9	3.26		





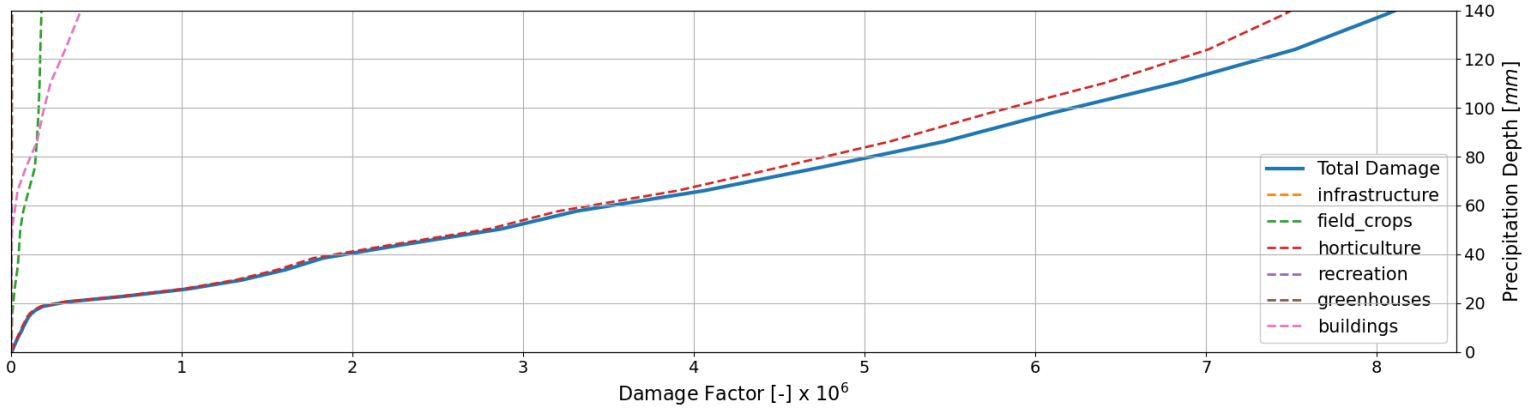
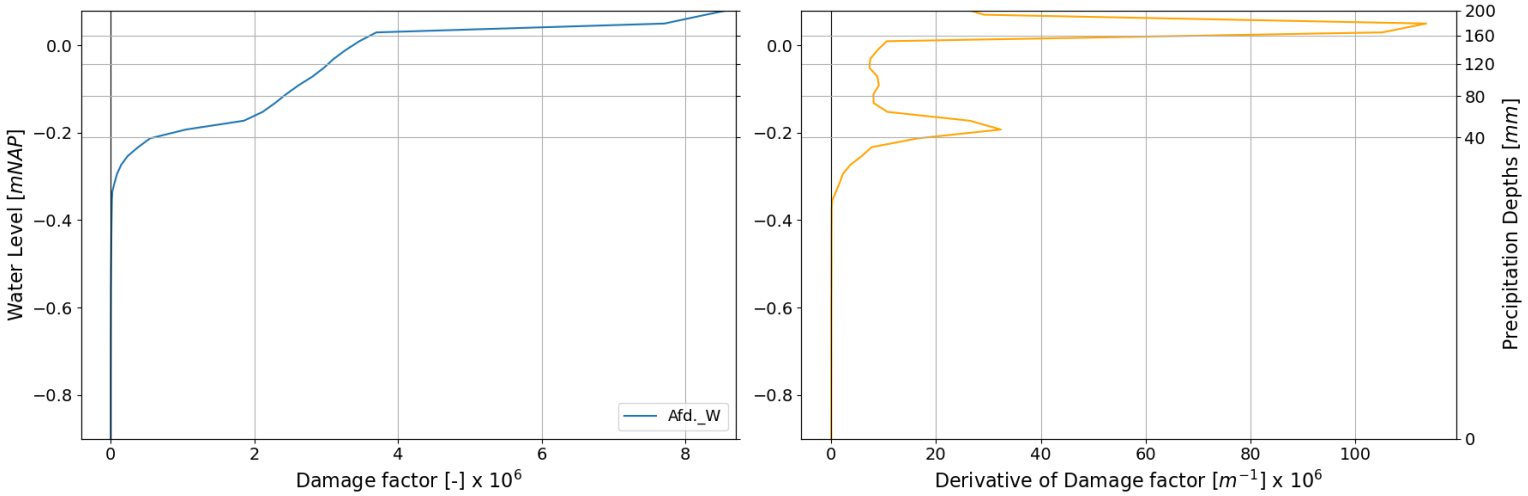
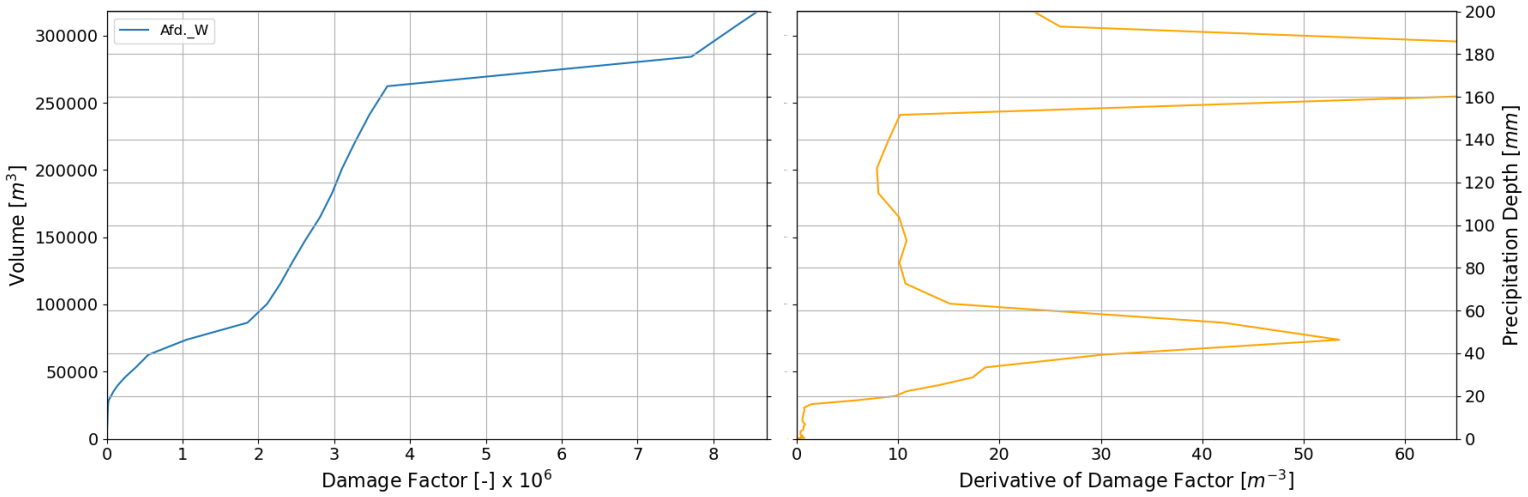


Table L.16: Afd. W characteristics

Afd. W			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	18.1		159	102
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	64.7	2.14	1.642	48600
Optimized	37.7	0.50		





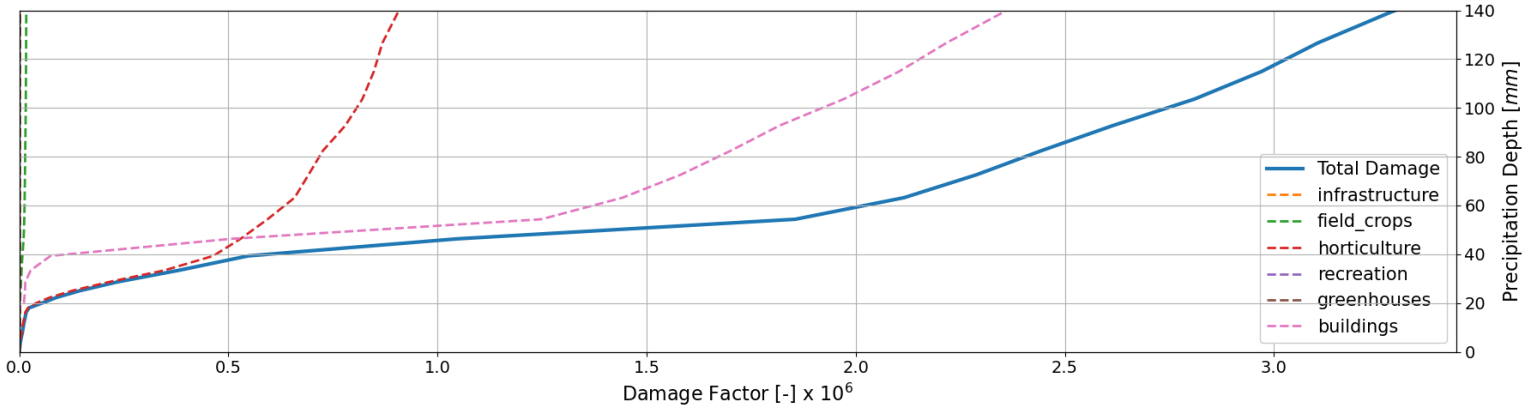
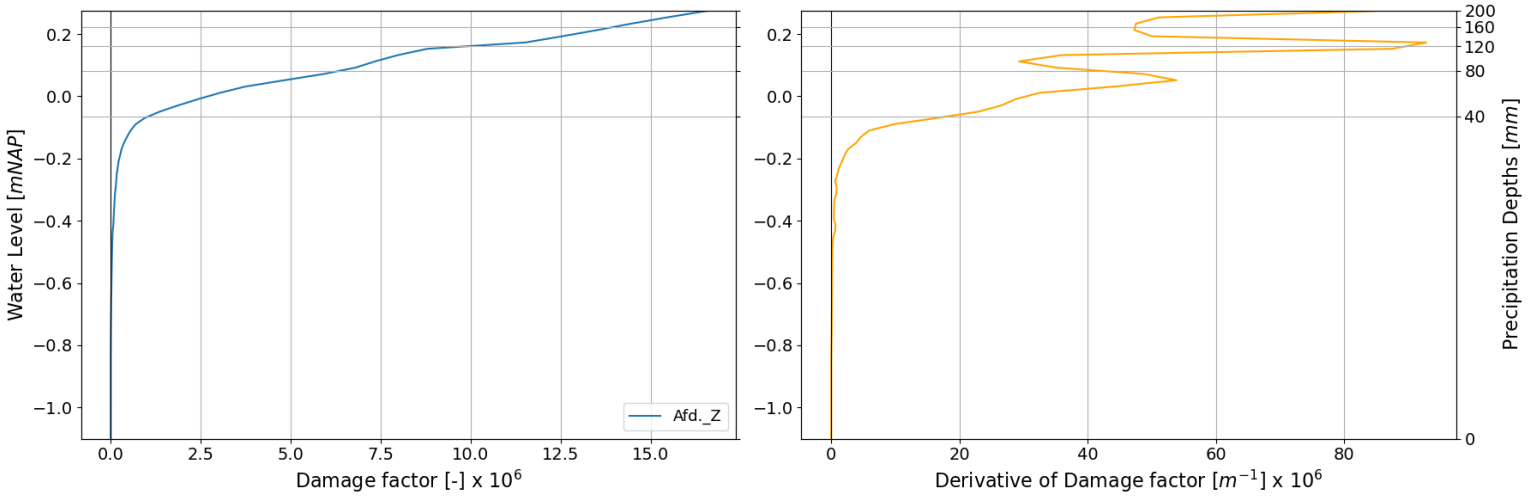
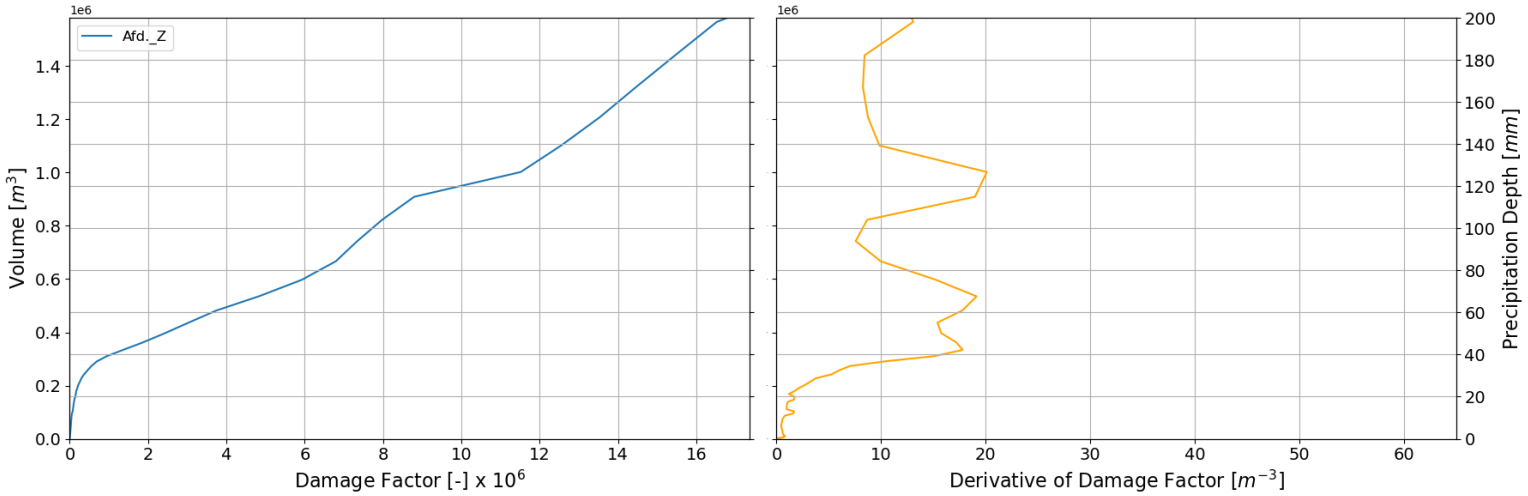


Table L.17: Afd. Z characteristics

Afd. Z			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
30	27.1		791	107
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	54.4	2.94	0.699	48600
Optimized	48.7	2.24		





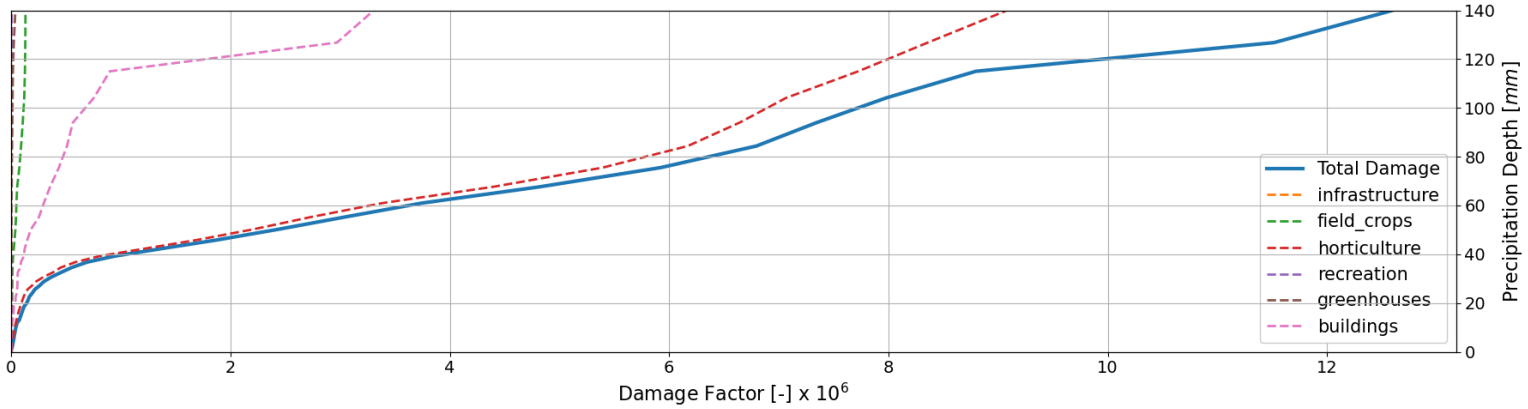
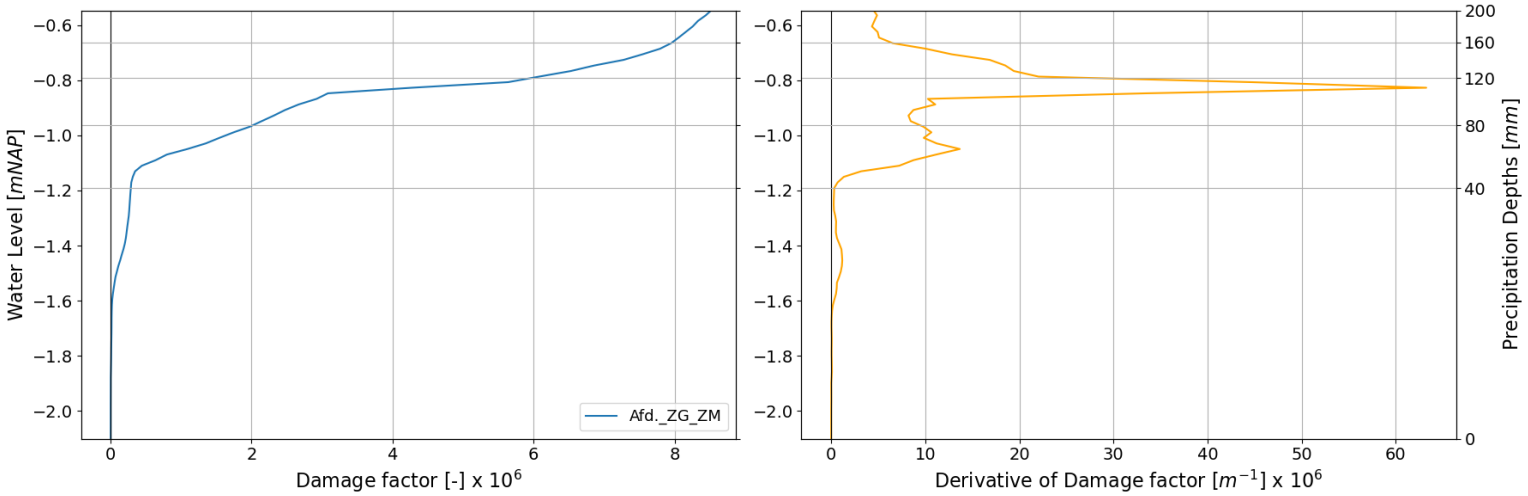
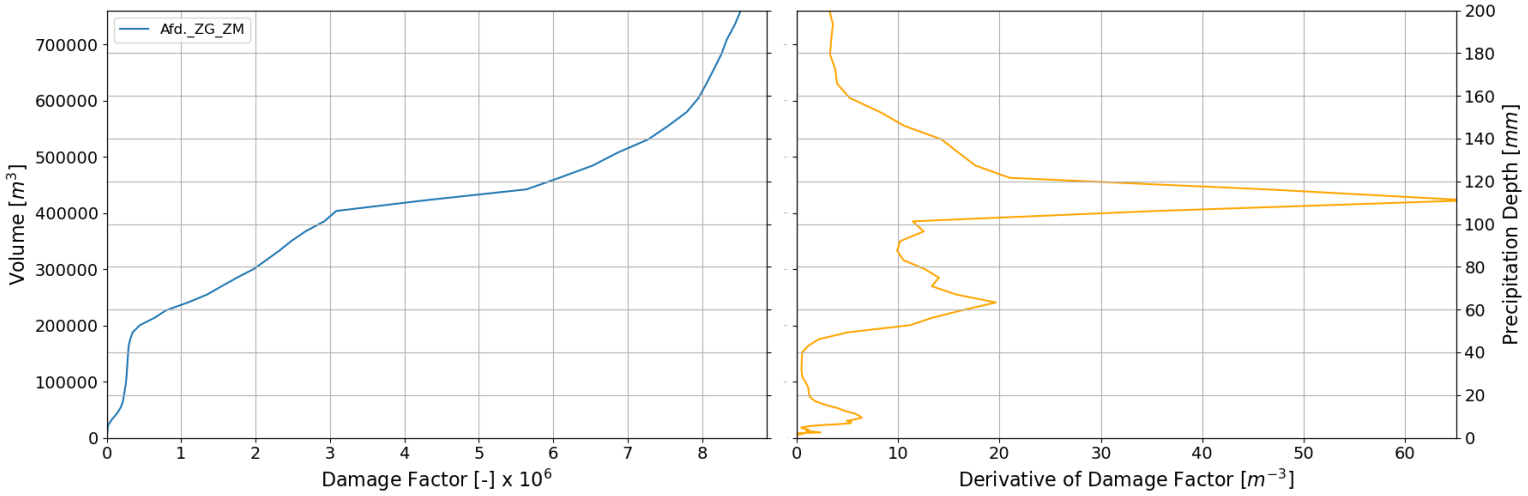


Table L.18: Afd. ZG-ZM characteristics

Afd. ZG-ZM			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
40	16.6		381	125
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	94.3	2.57	1.57	136440
Optimized	62.2	1.00		





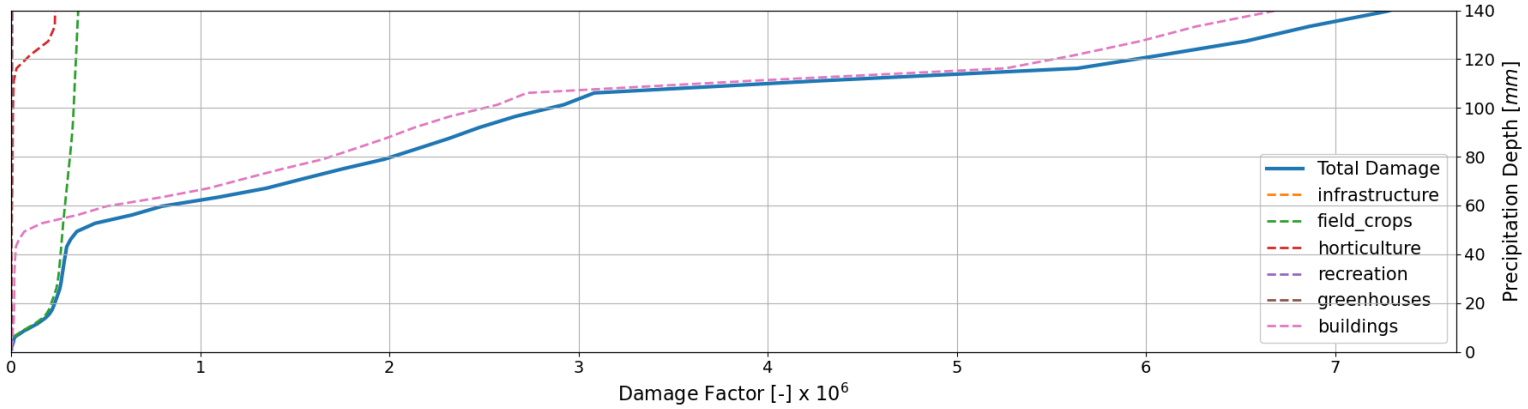
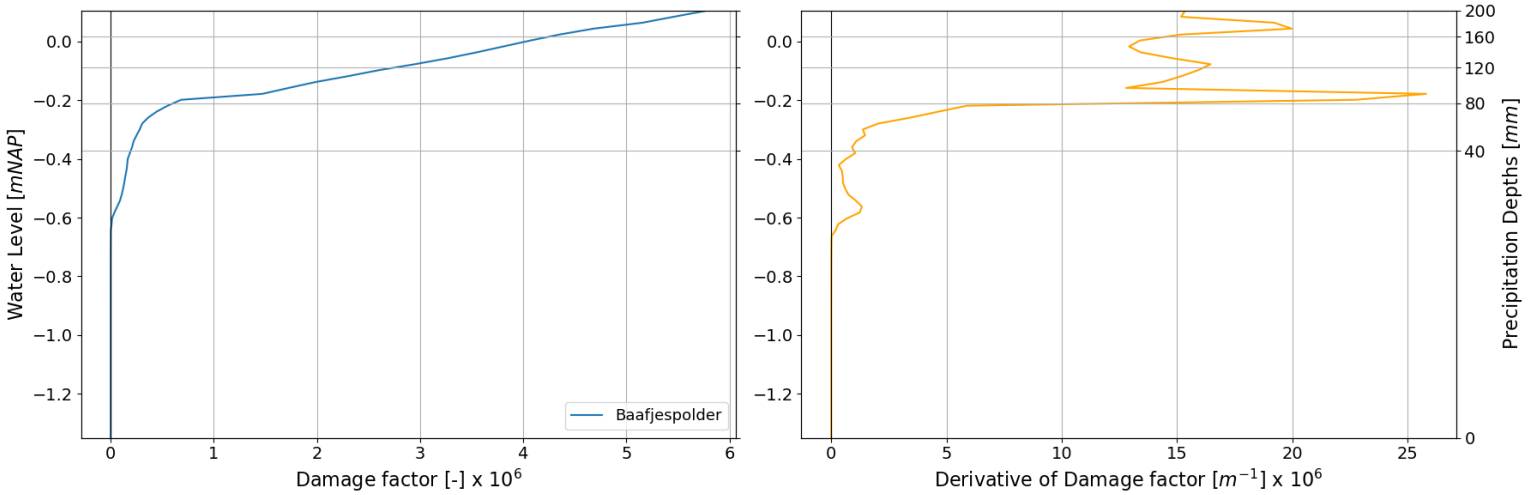
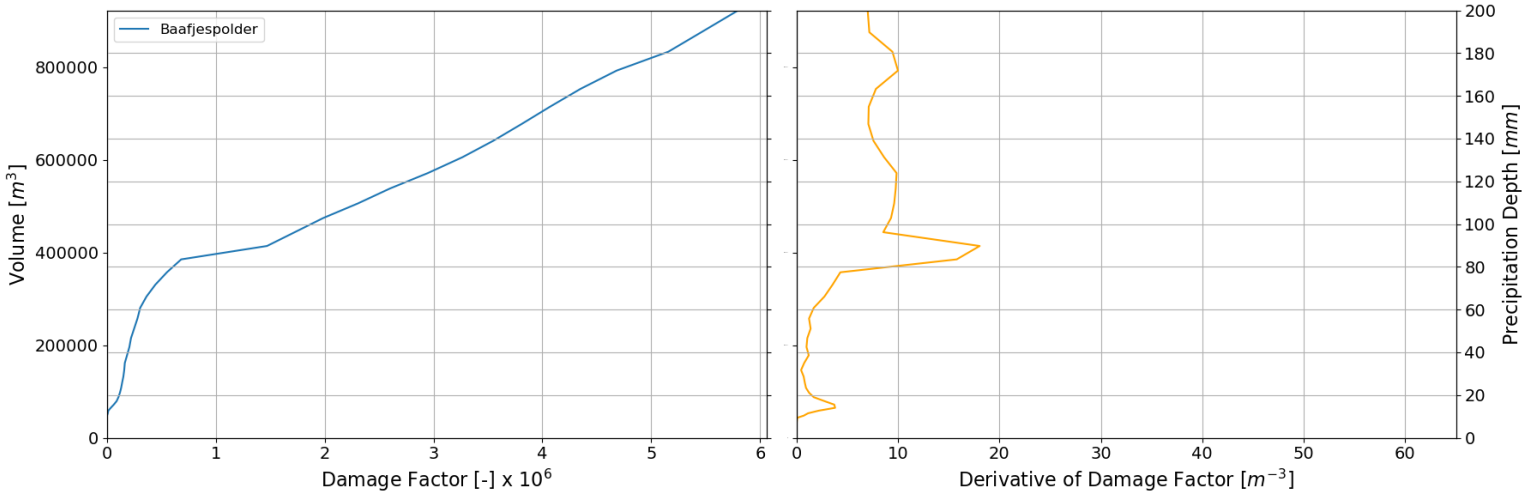


Table L.19: Baafjespolder characteristics

Baafjespolder			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
60	17.2		461	121
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	82.6	0.66	0.088	30600
Optimized	78.6	0.57		





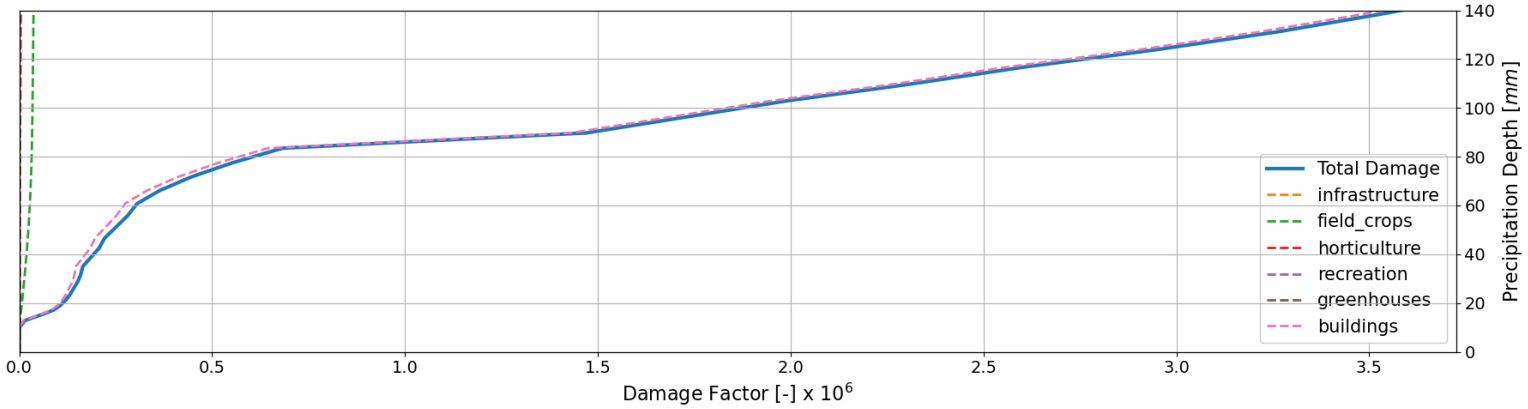
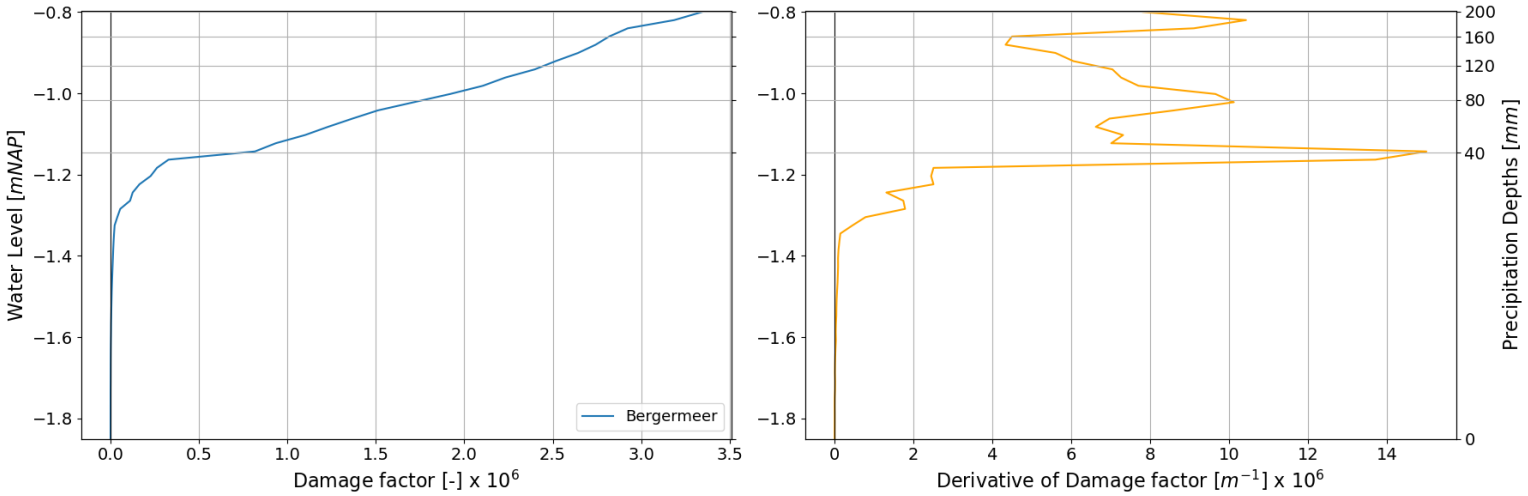
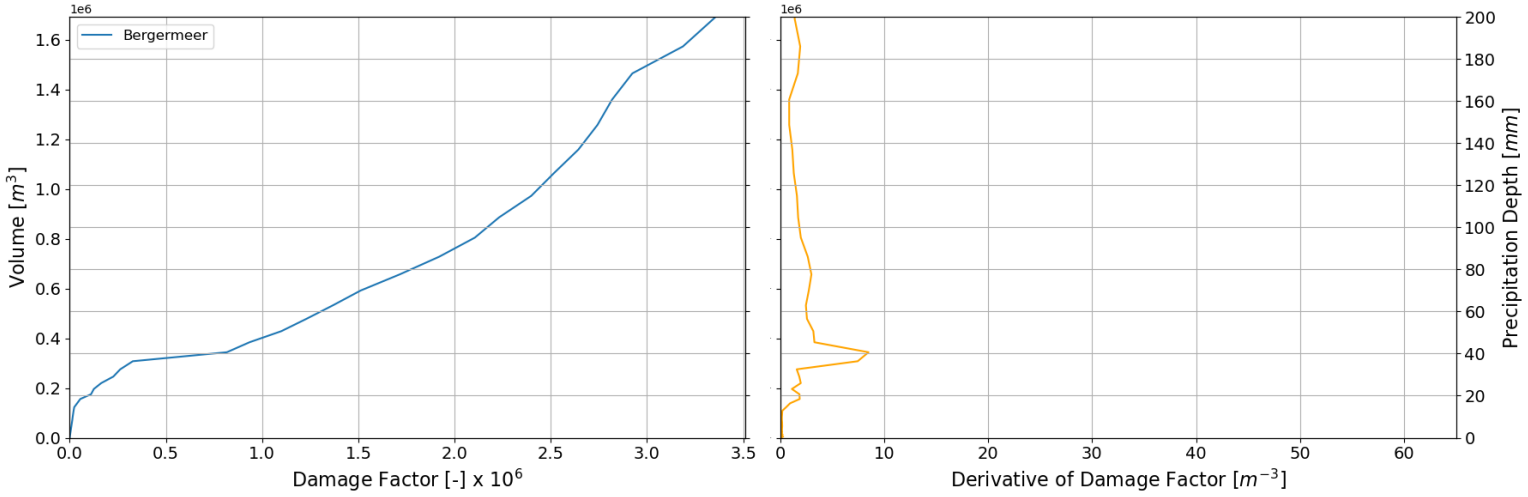


Table L.20: Bergermeer characteristics

Bergermeer			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	23.1		846	124
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	75.4	1.65	0	0
Optimized	75.4	1.65		





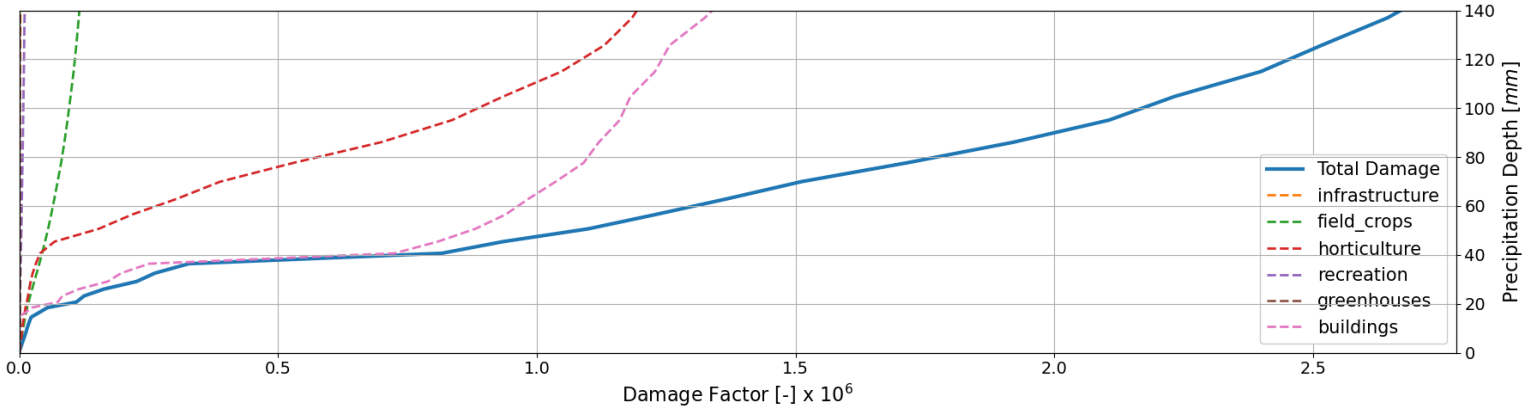
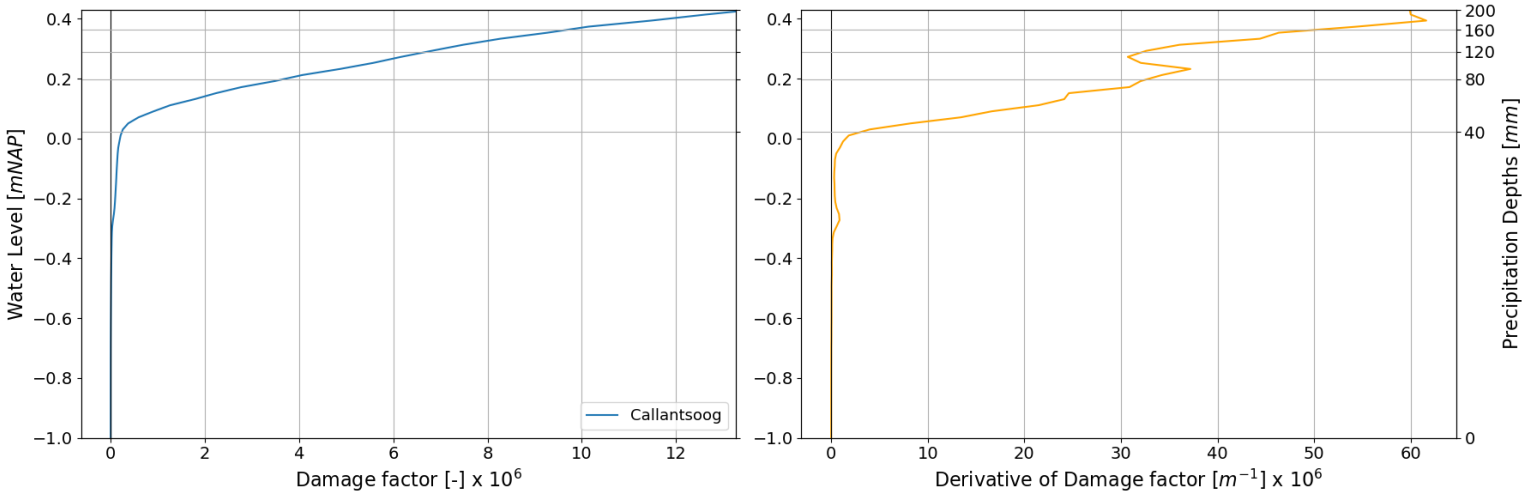
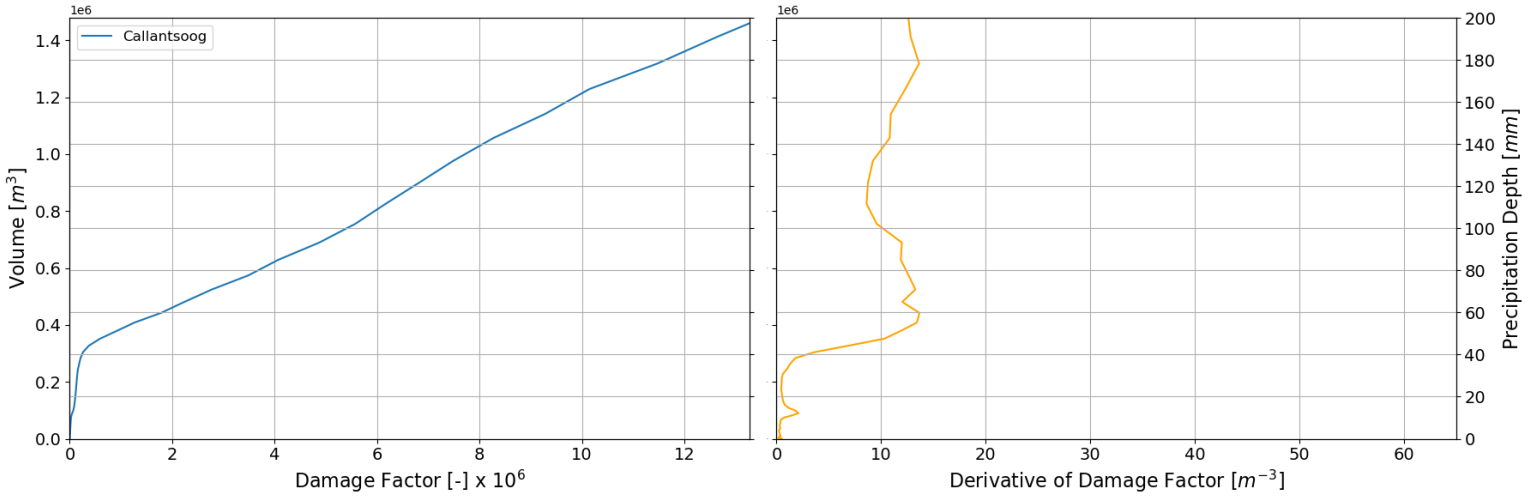


Table L.21: Callantsoog characteristics

Callantsoog			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
40	17.5		739	99
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	65.0	2.23	1.782	158760
Optimized	45.4	0.45		





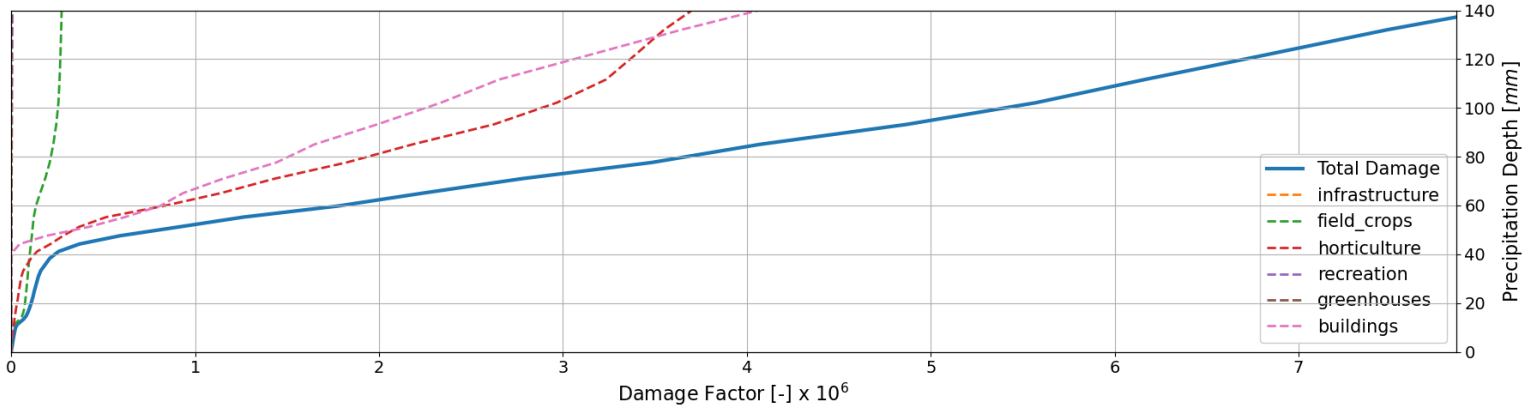
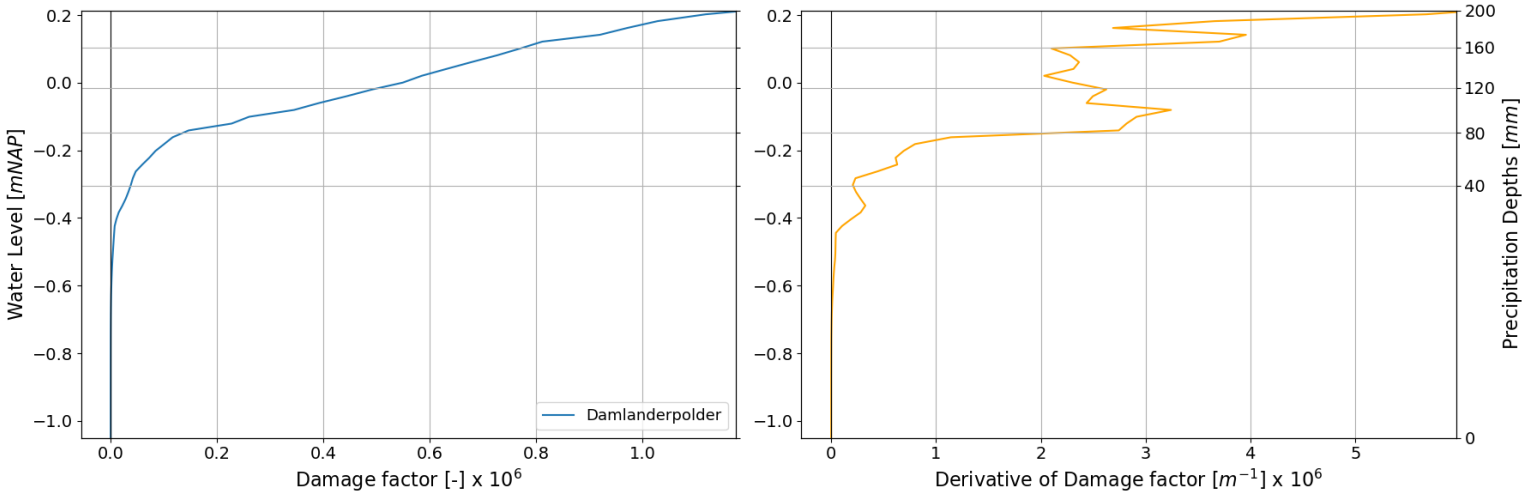
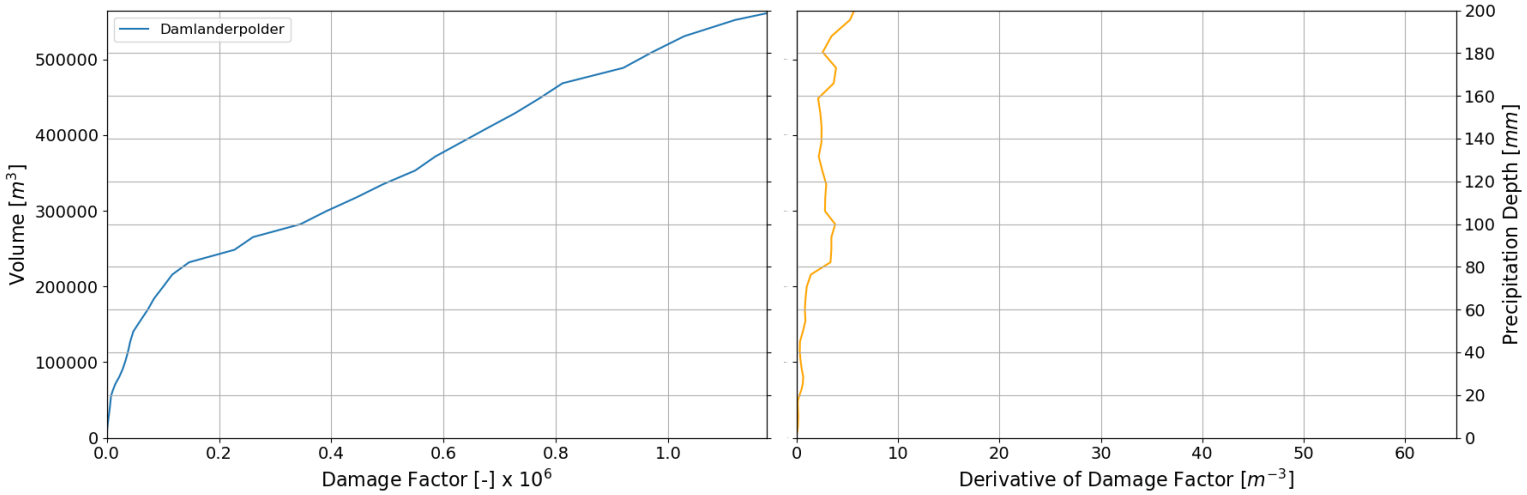


Table L.22: Damlanderpolder characteristics

Damlanderpolder			Type 1	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
80	10.7		282	152
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	128.2	0.57	0	0
Optimized	128.2	0.57		





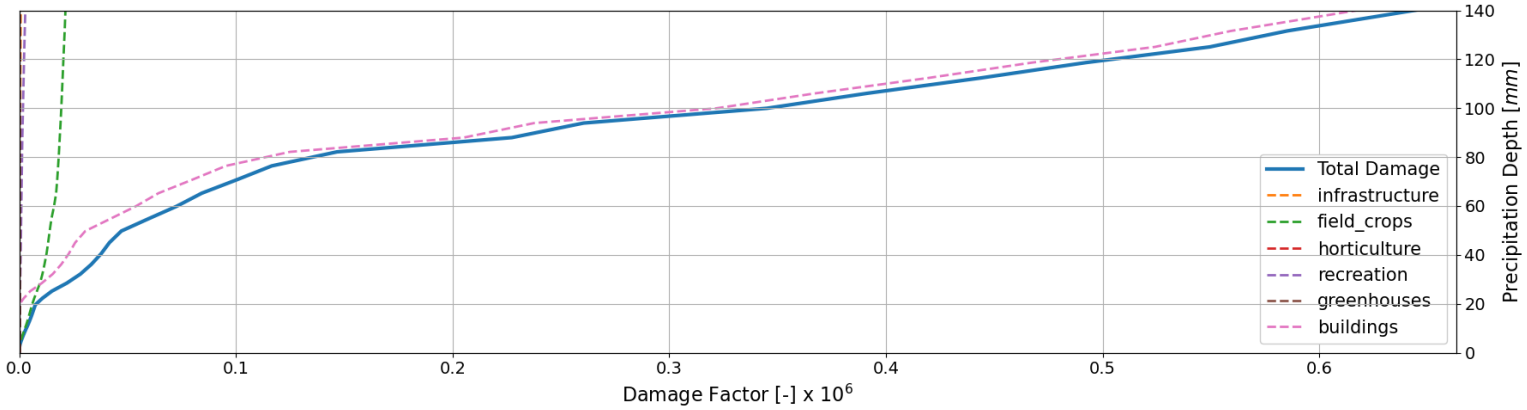
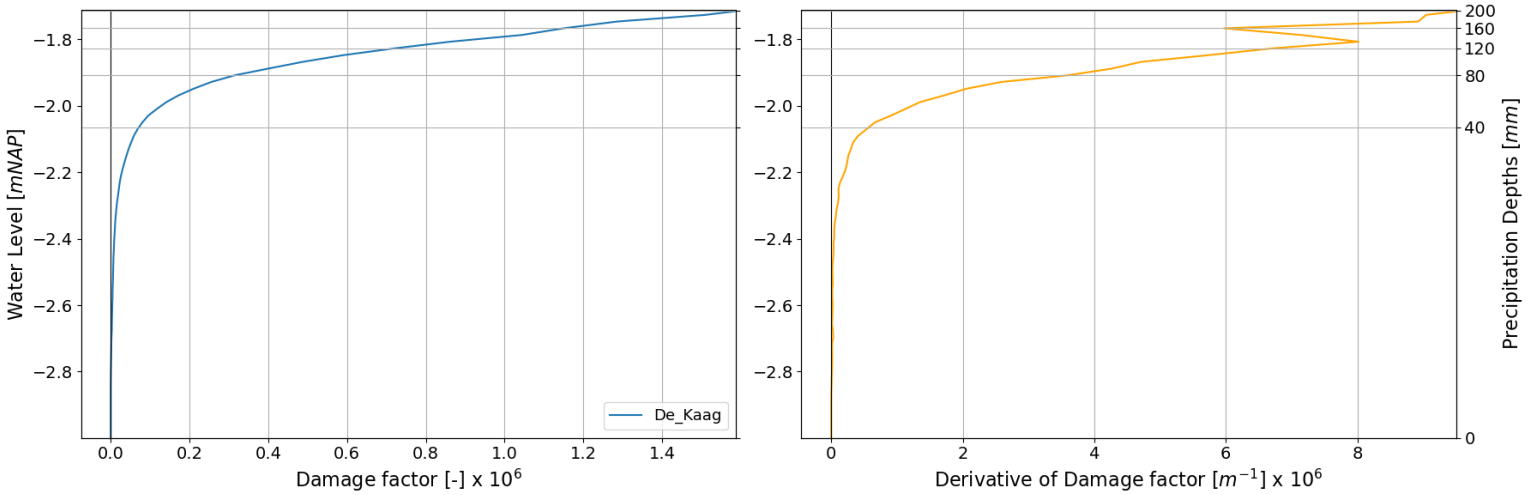
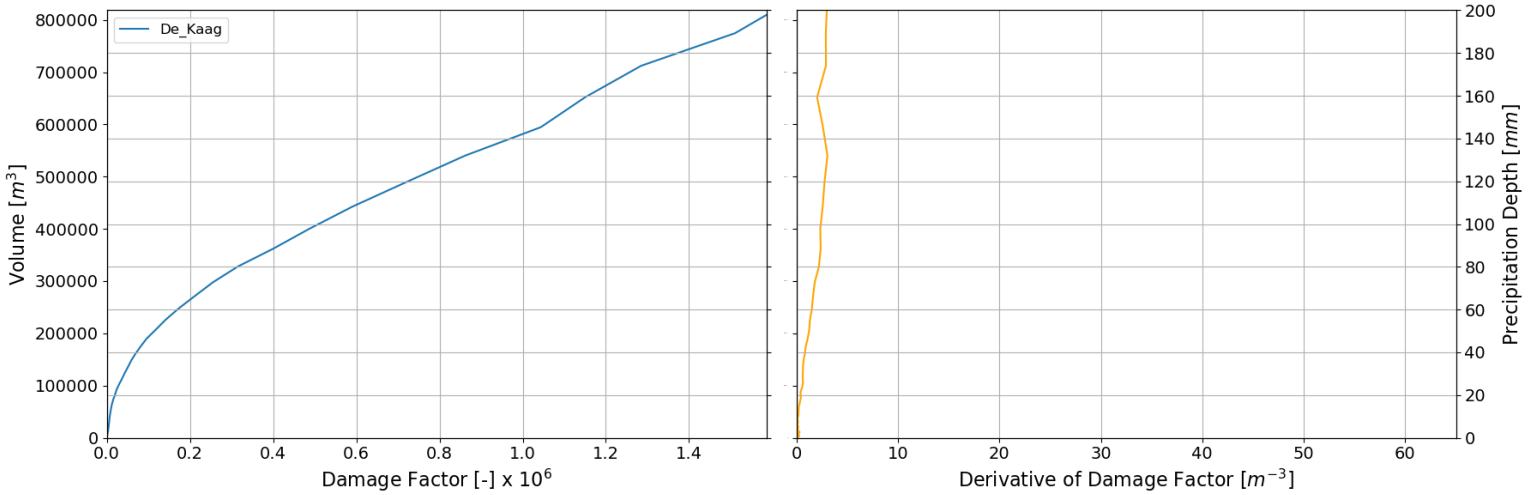


Table L.23: De Kaag characteristics

De Kaag			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	14.1		409	76
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	46	0.09	0	0
Optimized	46	0.09		





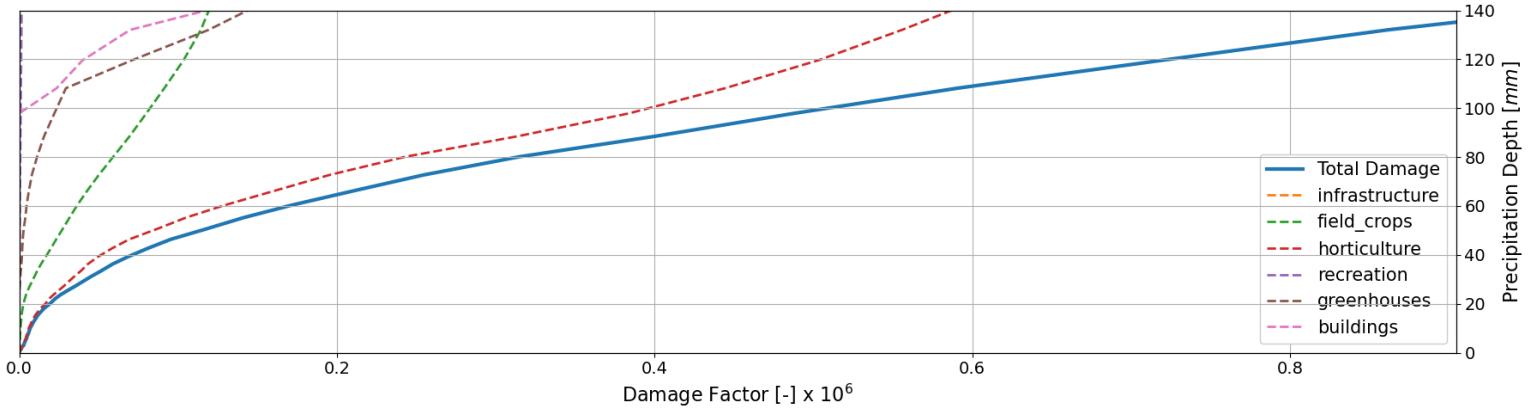
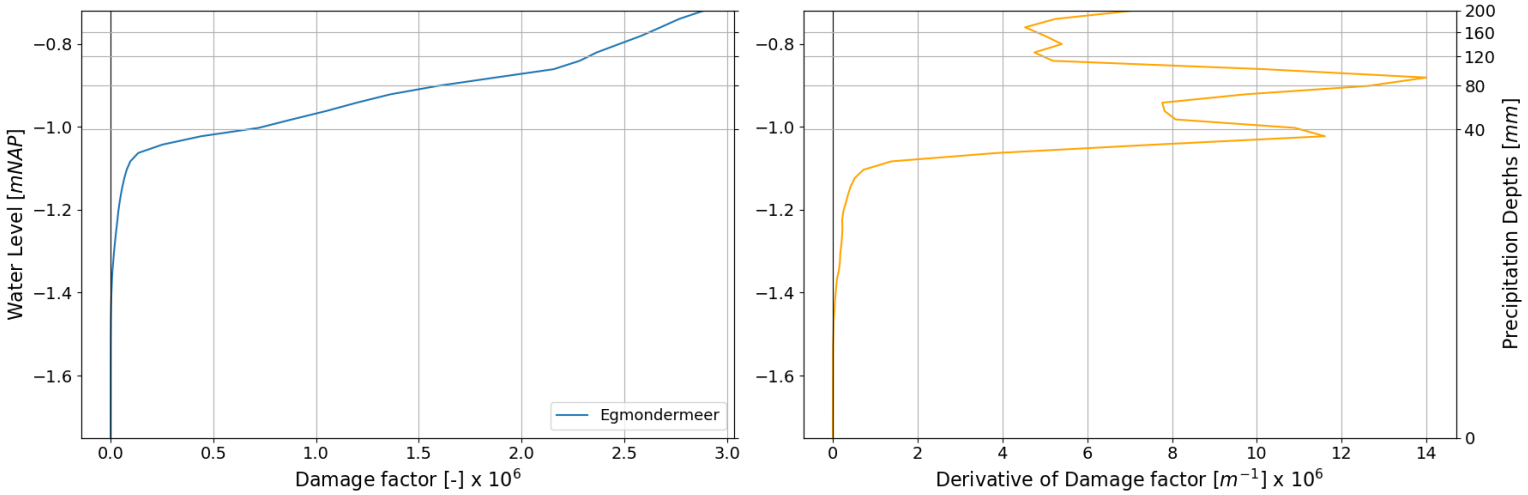
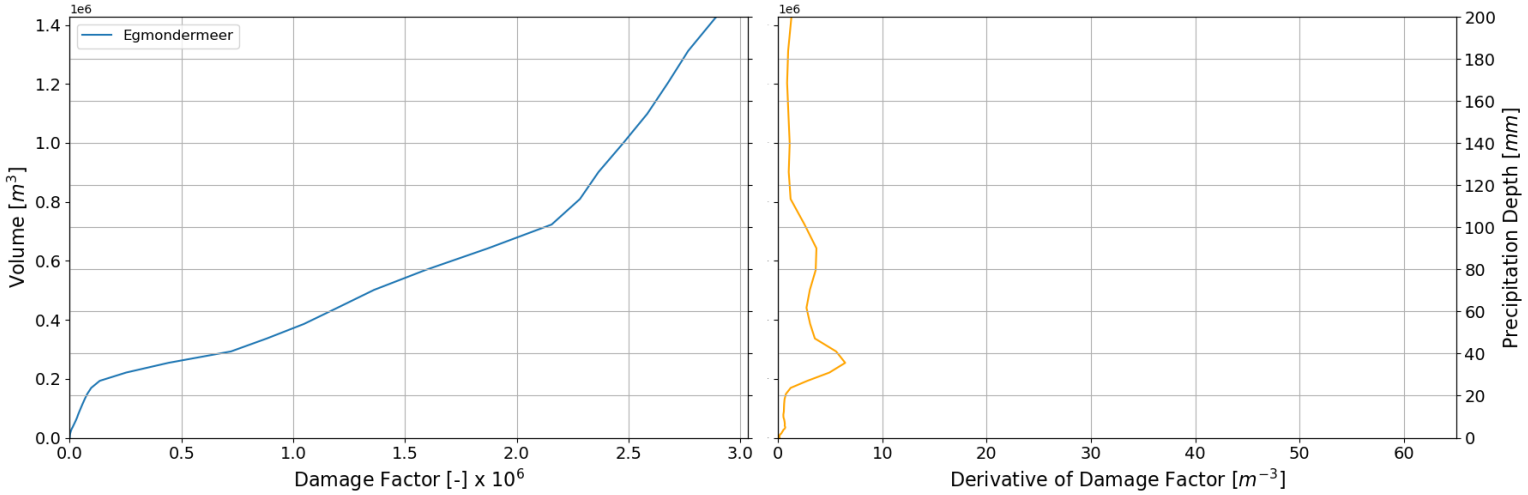


Table L.24: Egmondermeer characteristics

Egmondermeer			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	16.1		714	130
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	94.5	1.98	0.011	6480
Optimized	94.0	1.97		





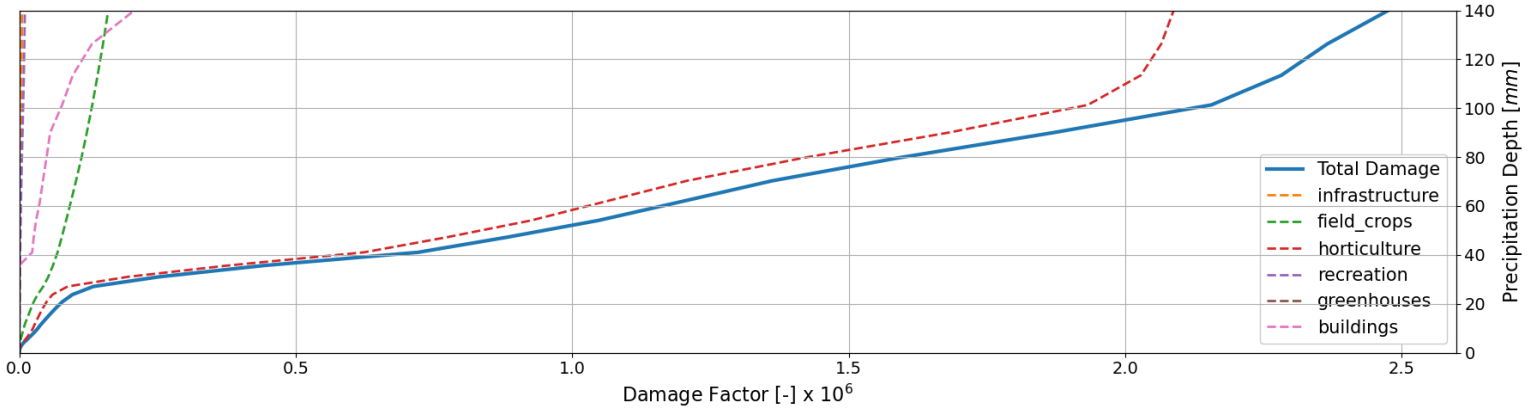
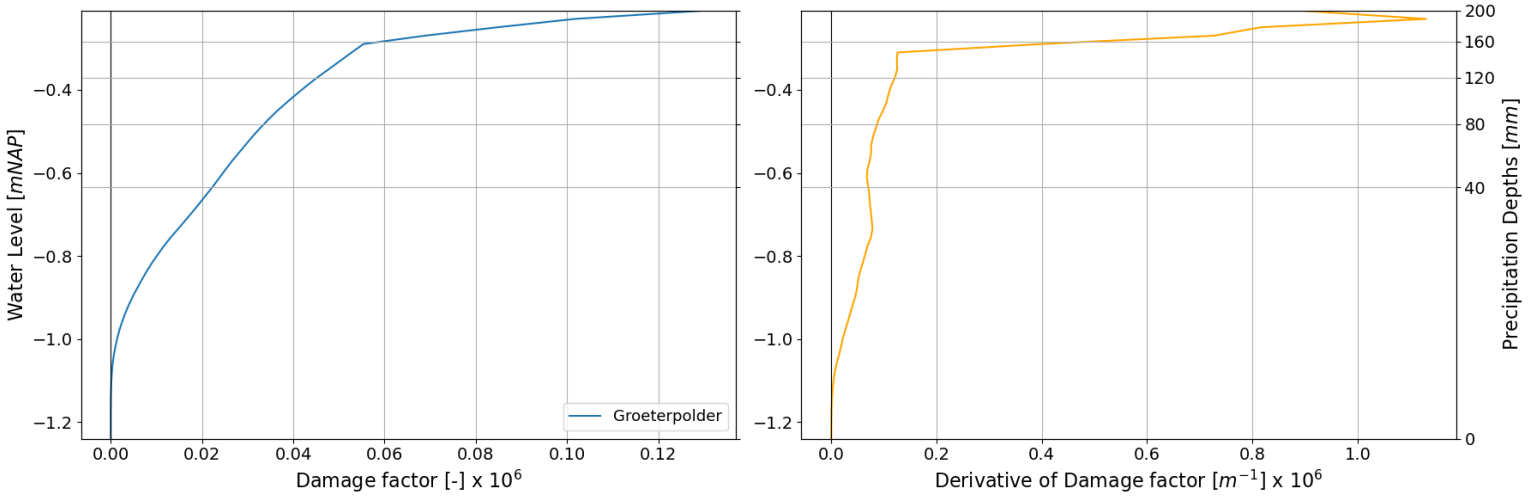
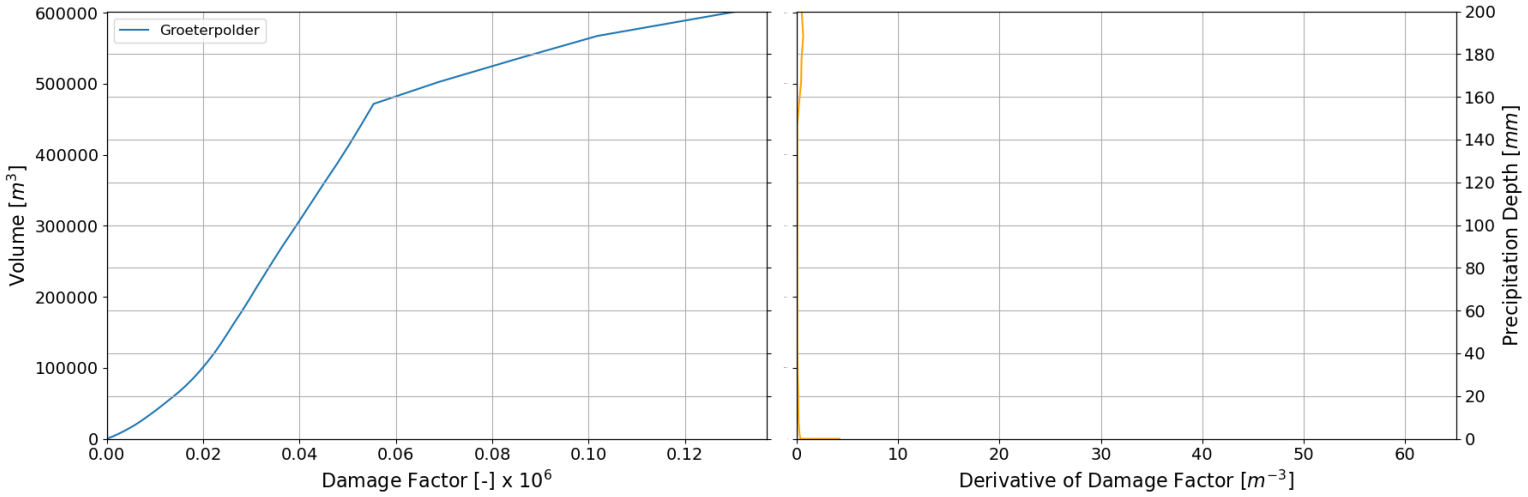


Table L.25: Groeterpolder characteristics

Groeterpolder			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	11.5		301	138
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	113	0.04	0	0
Optimized	113	0.04		





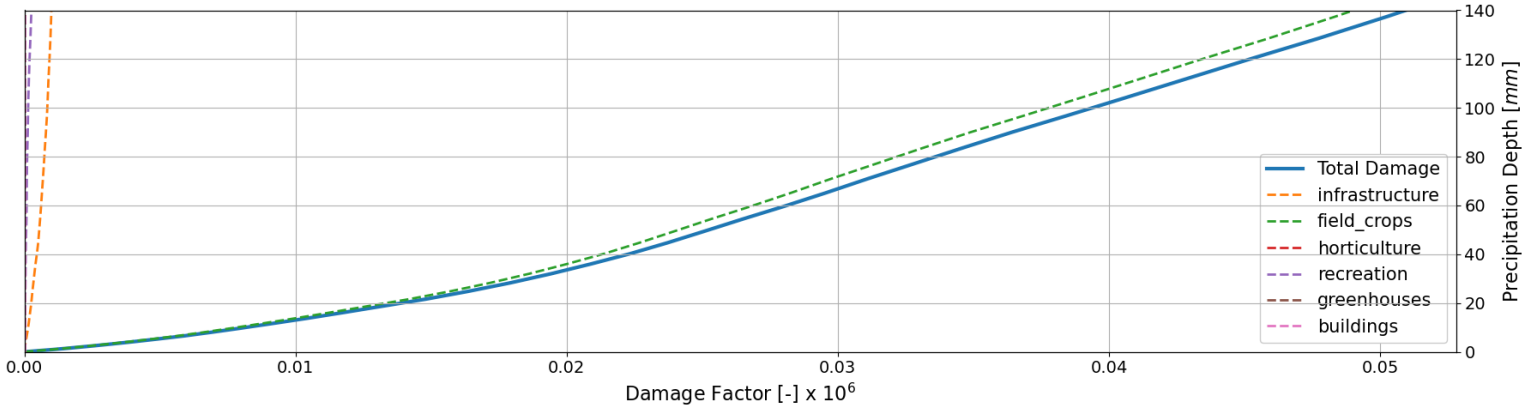
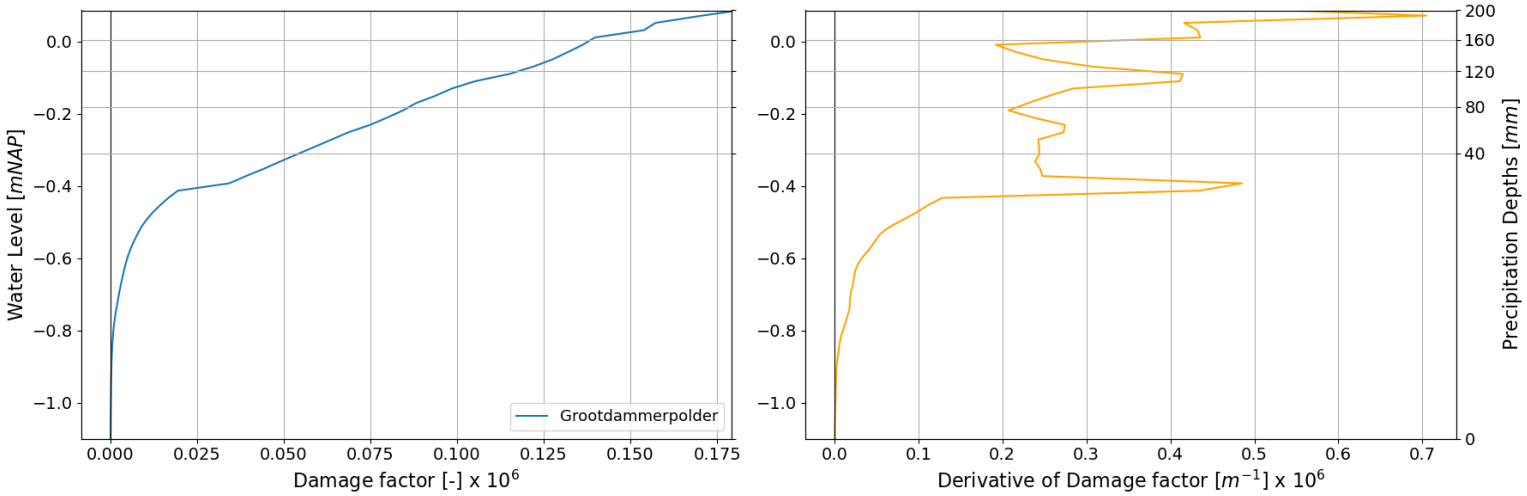
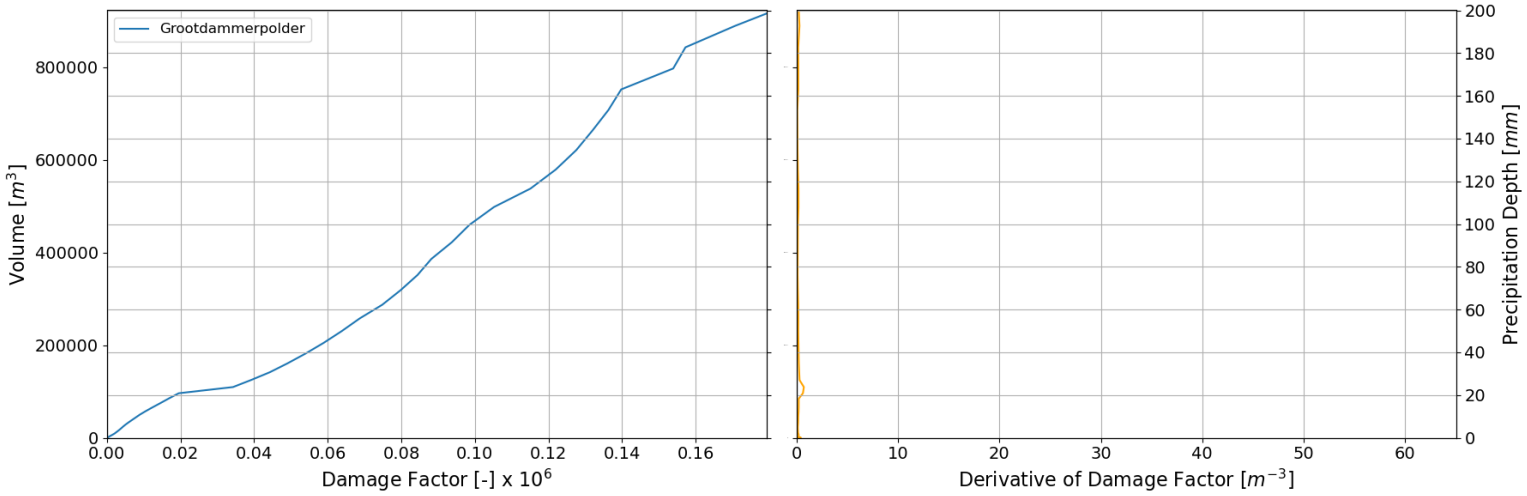


Table L.26: Grootdammerpolder characteristics

Grootdammerpolder			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	10.3		461	152
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	129	0.12	0	0
Optimized	129	0.12		





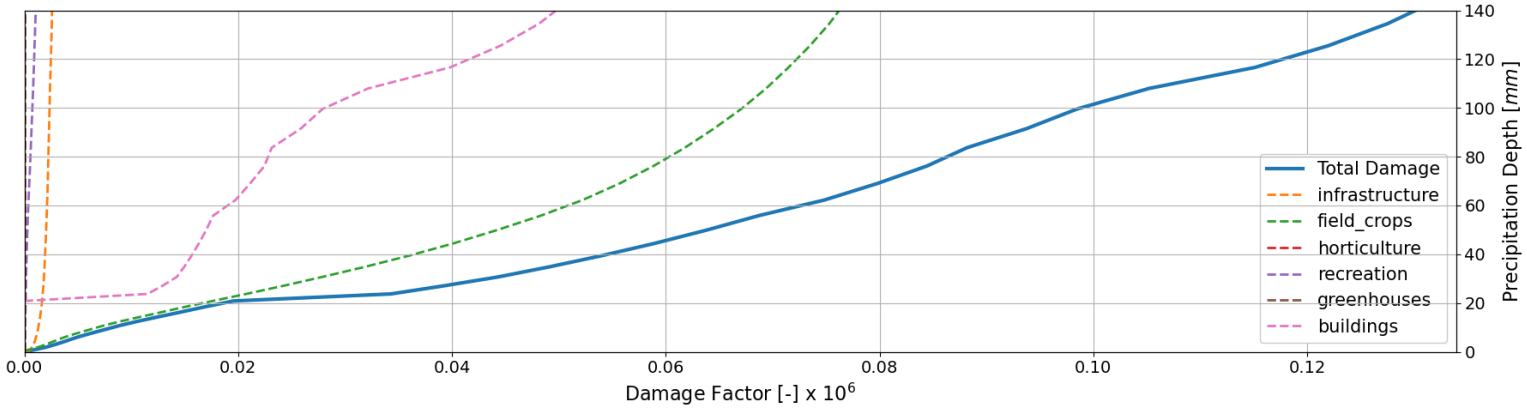
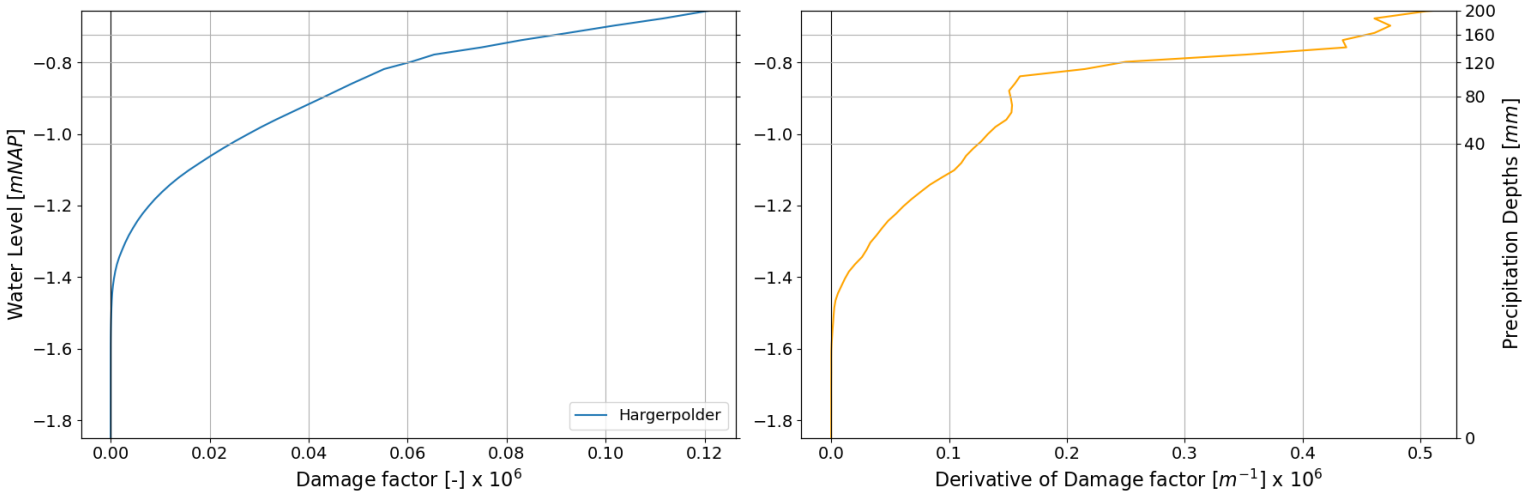
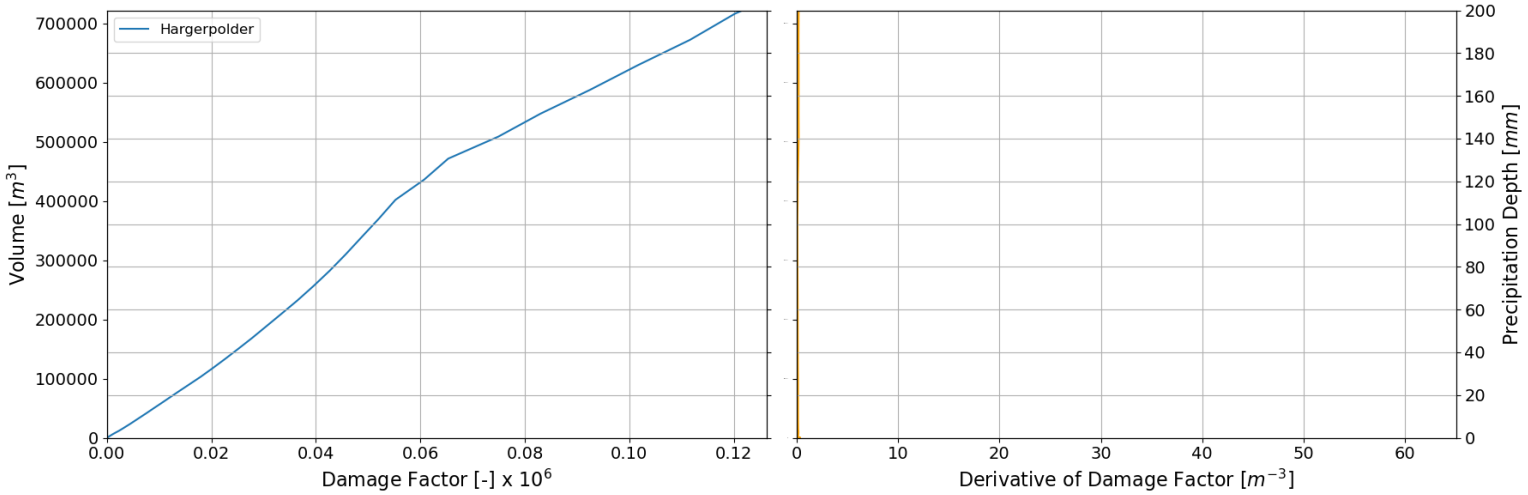


Table L.27: Hargerpolder characteristics

Hargerpolder			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	15.3		361	114
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	80	0.04	0	0
Optimized	80	0.04		





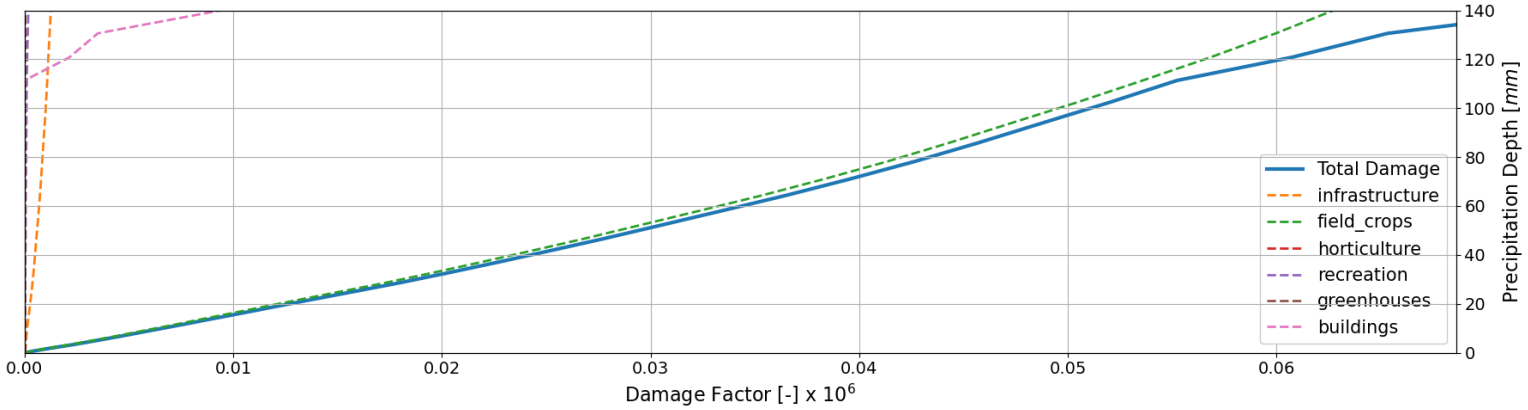
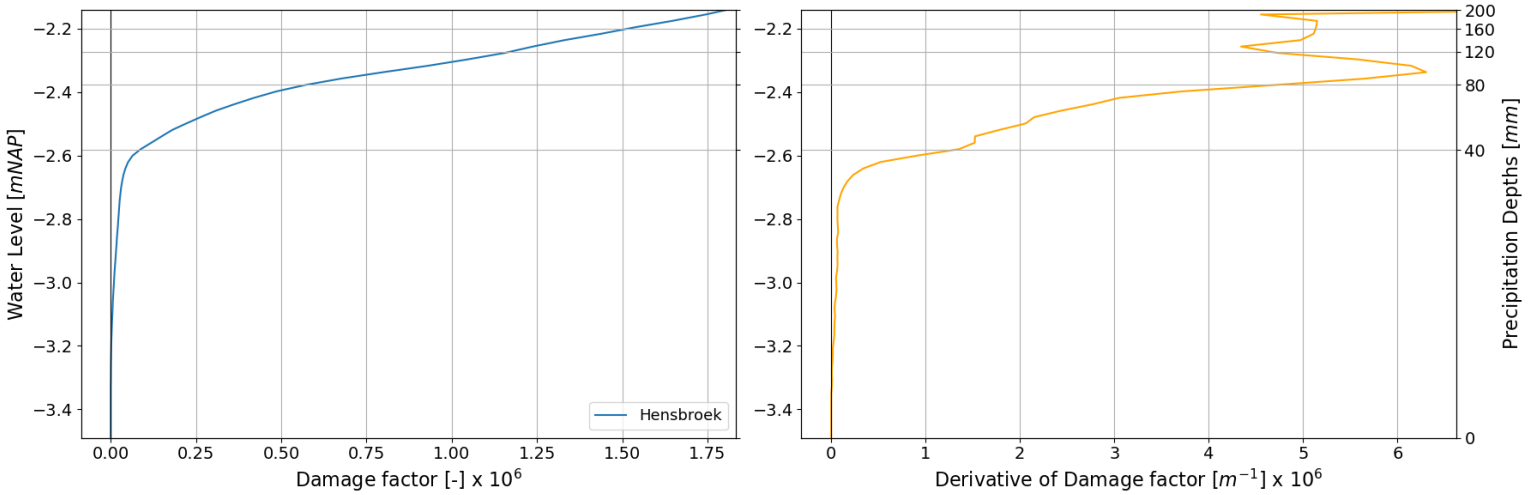
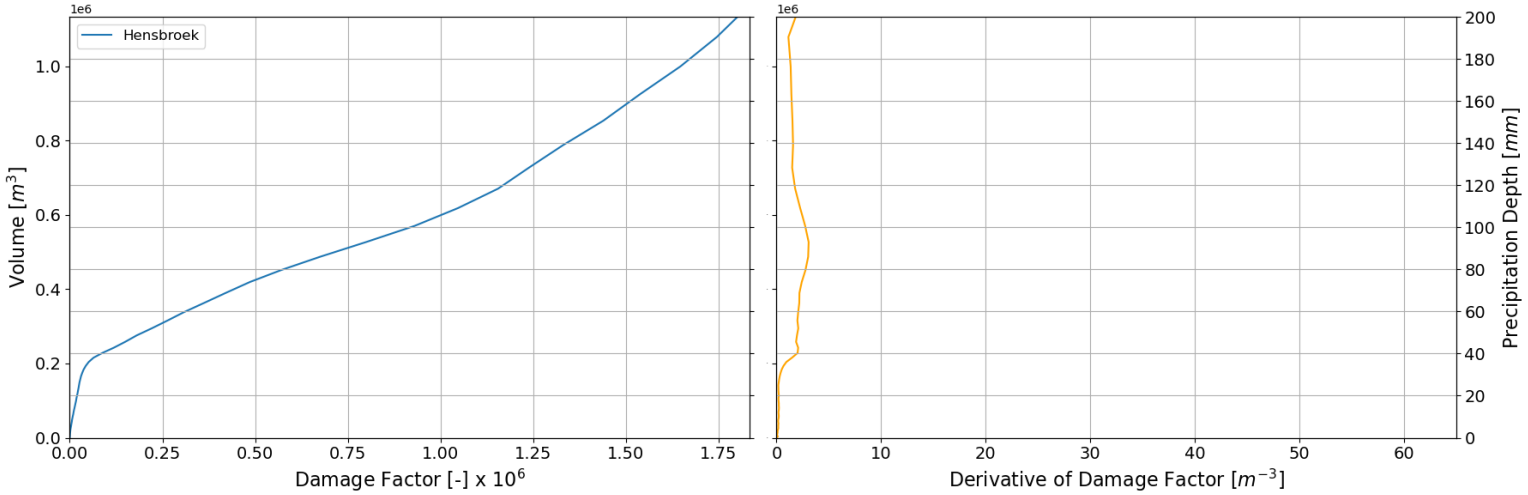


Table L.28: Hensbroek characteristics

Hensbroek			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	15.2		567	66
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	34	0.04	0	0
Optimized	34	0.04		





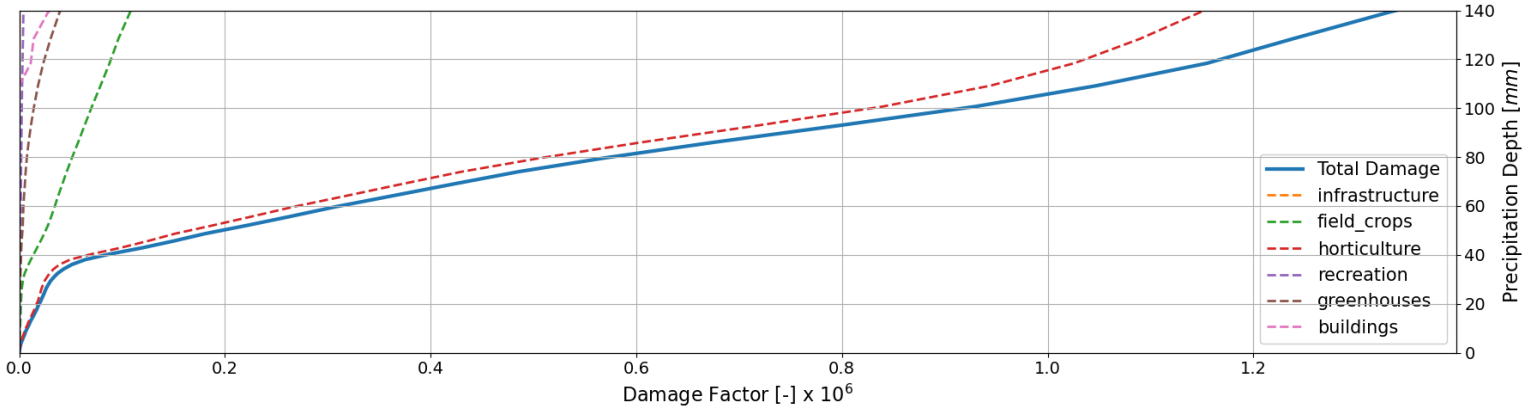
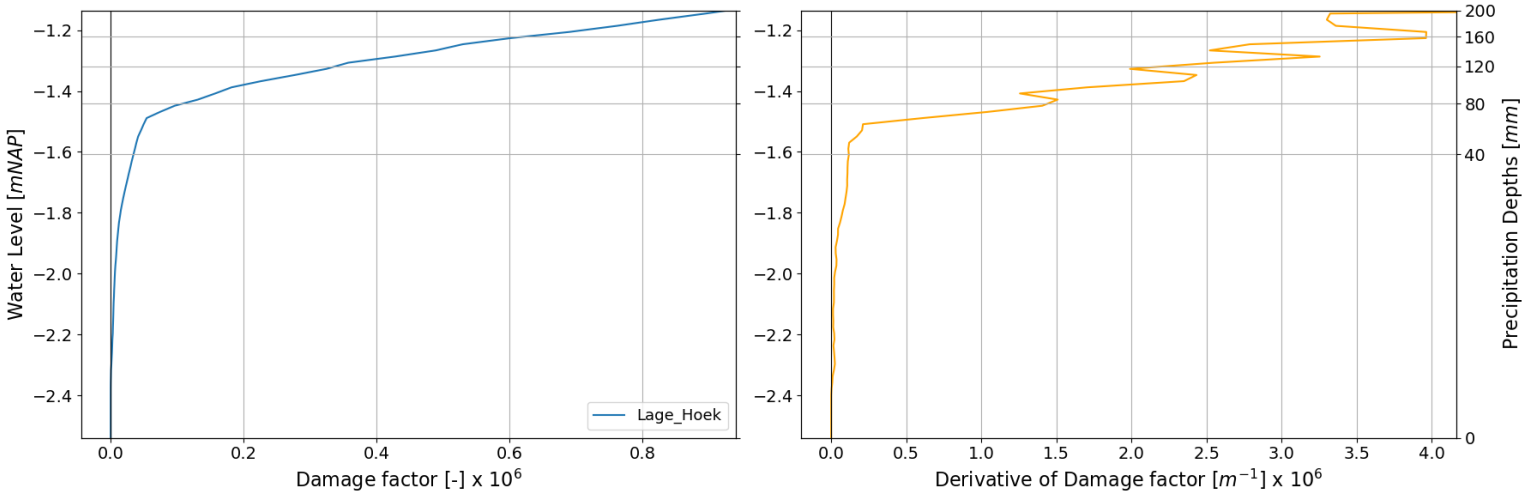
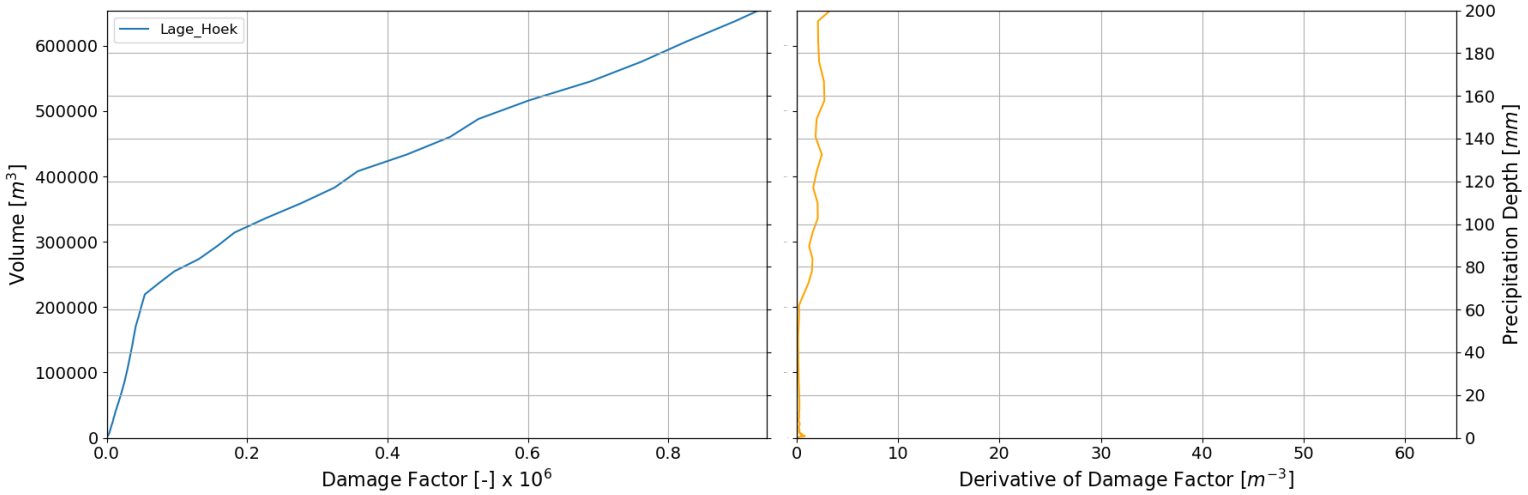


Table L.29: Lage Hoek characteristics

Lage Hoek			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	20.9		327	78
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	38	0.03	0	0
Optimized	38	0.03		





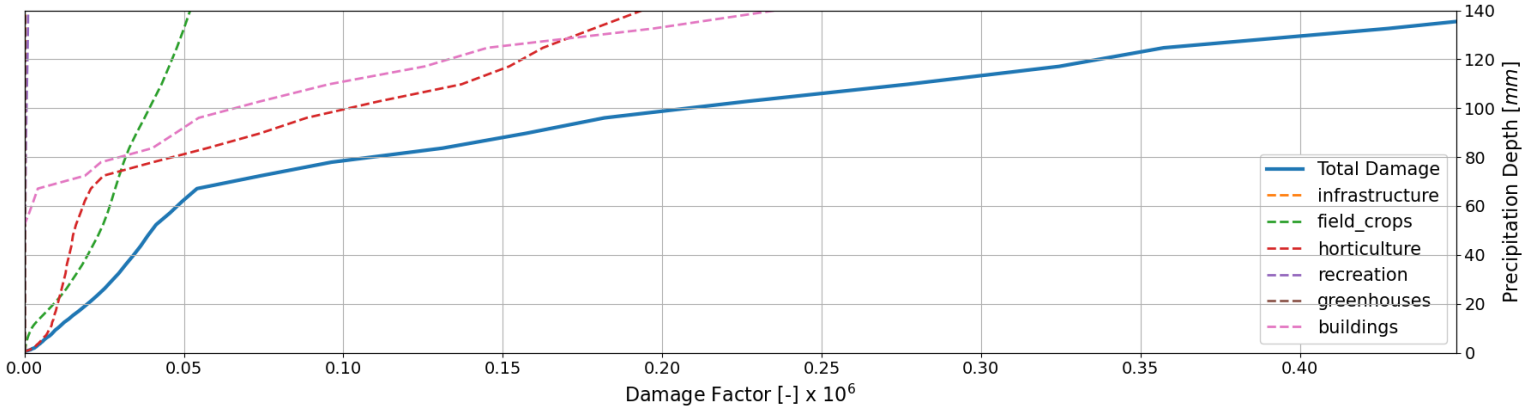
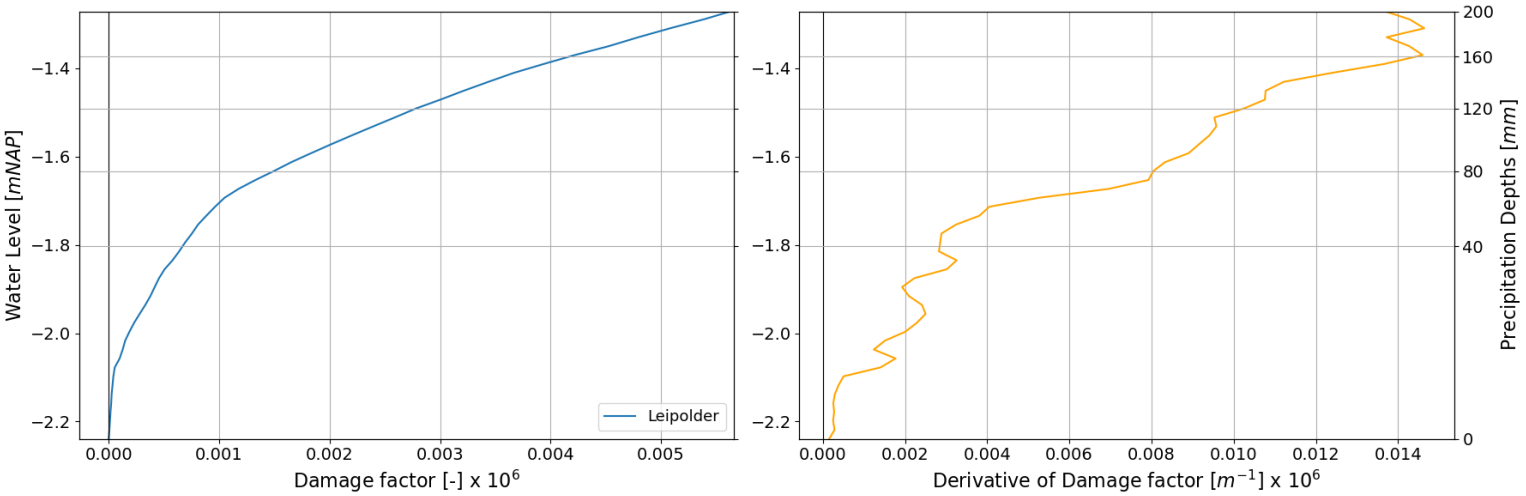
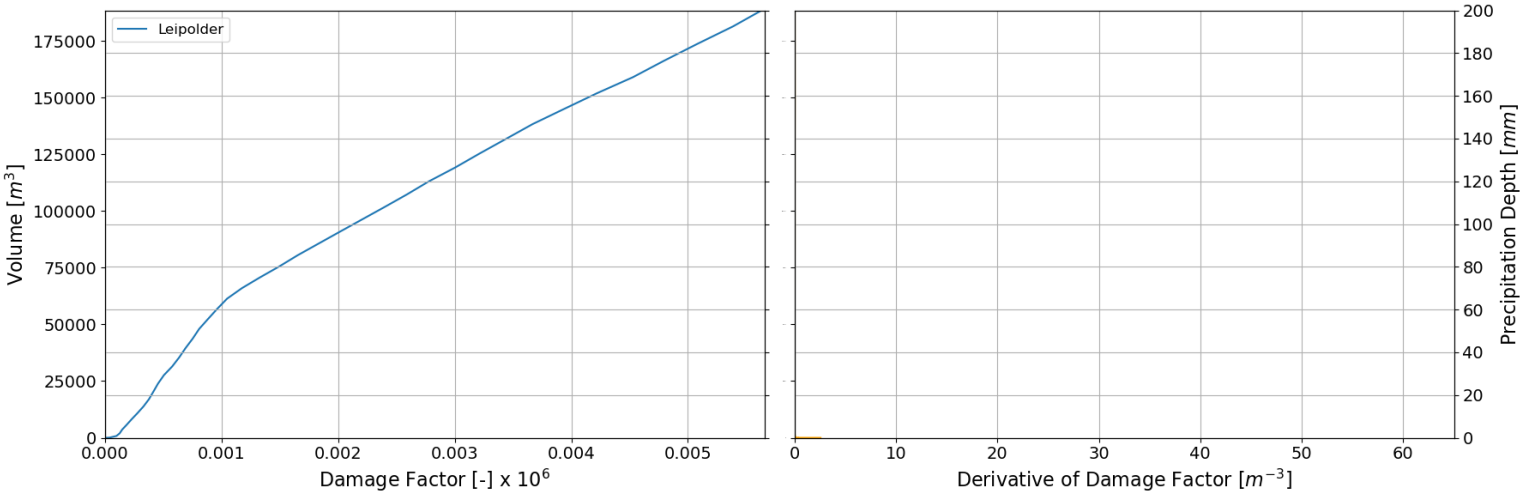


Table L.30: Leipolder characteristics

Leipolder			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	14.7		94	104
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	72	0	0	0
Optimized	72	0		





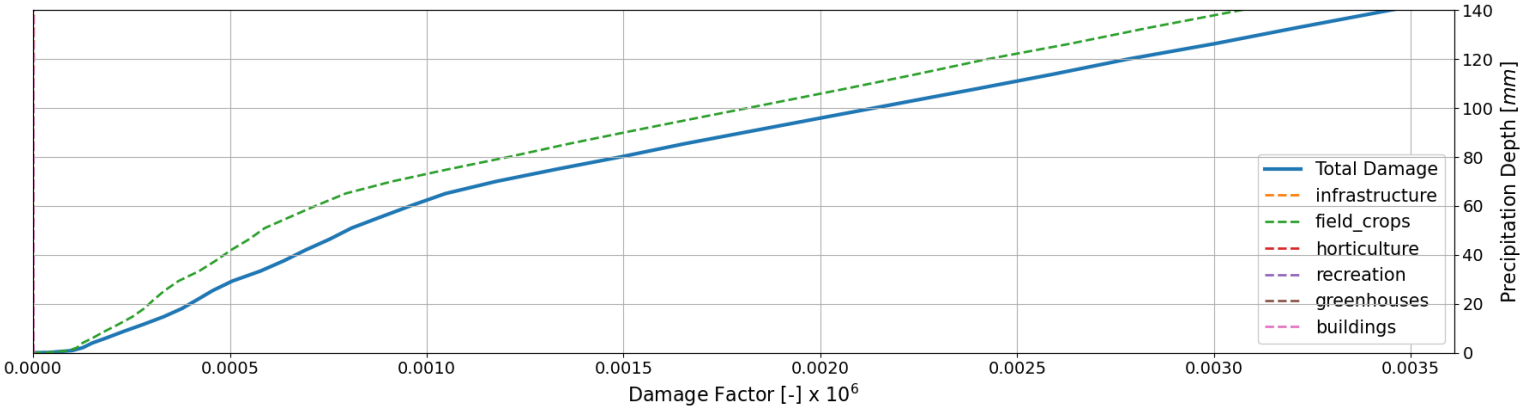
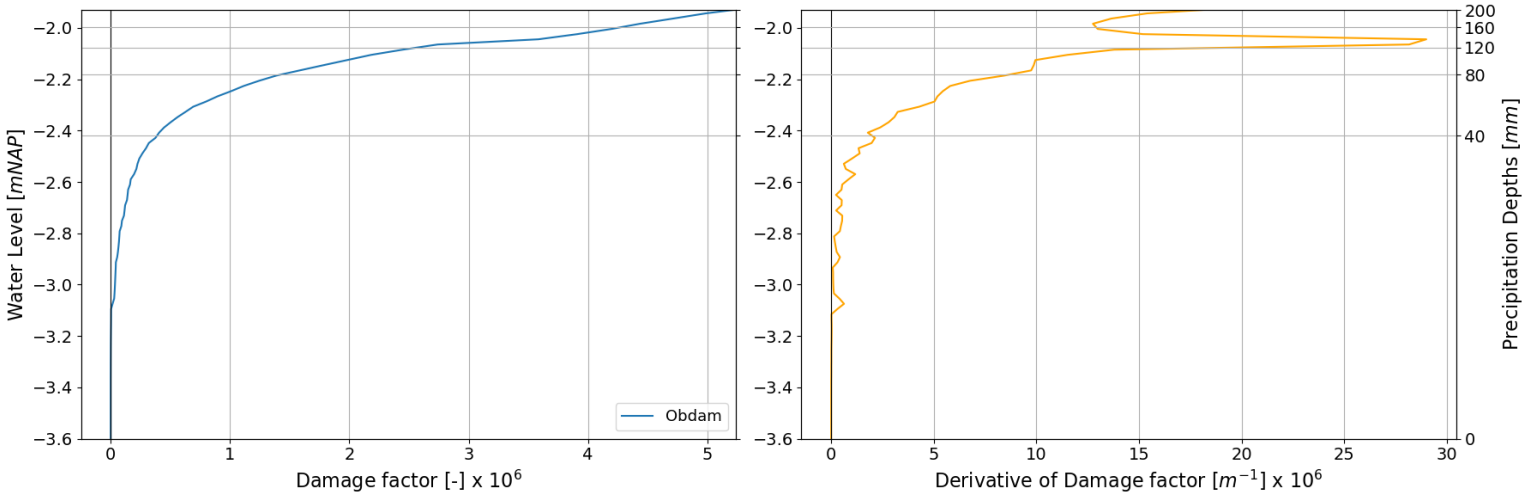
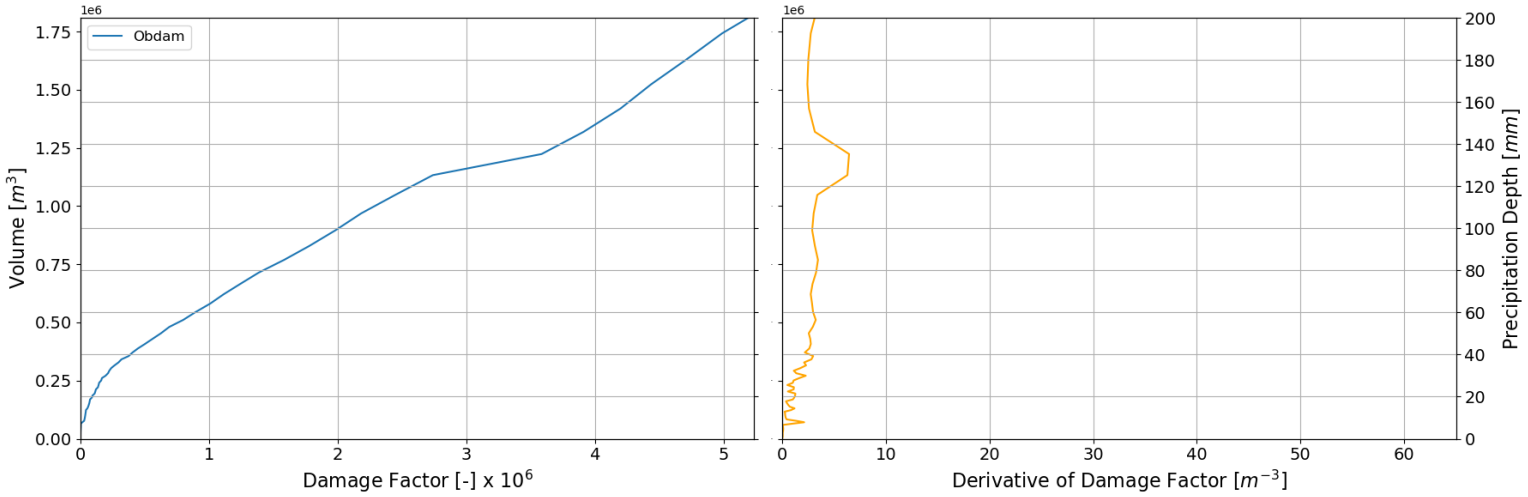


Table L.31: Obdam characteristics

Obdam			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	42.9		905	71
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	38	0.34	0	0
Optimized	38	0.34		





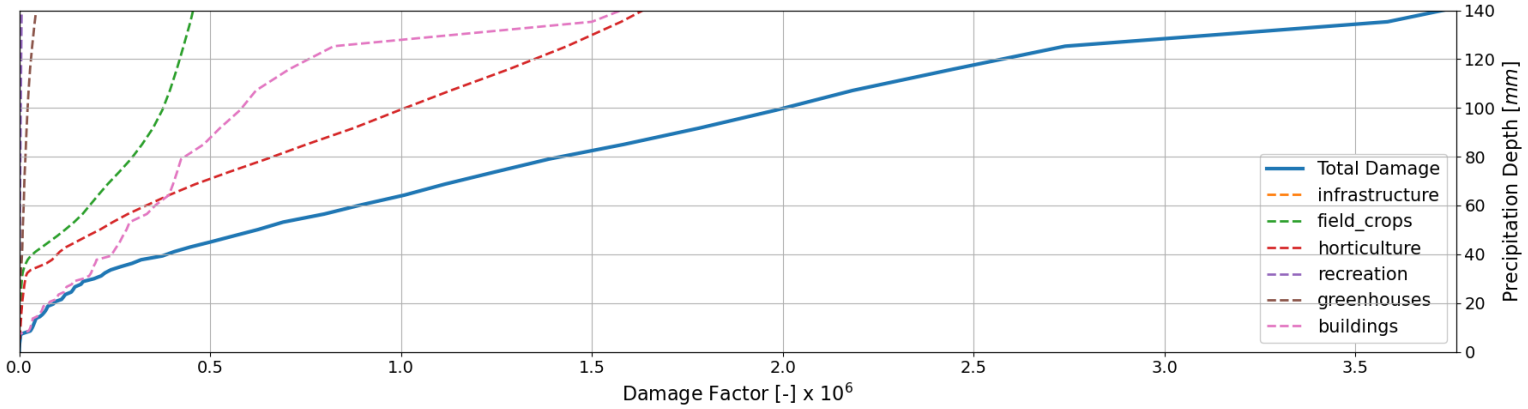
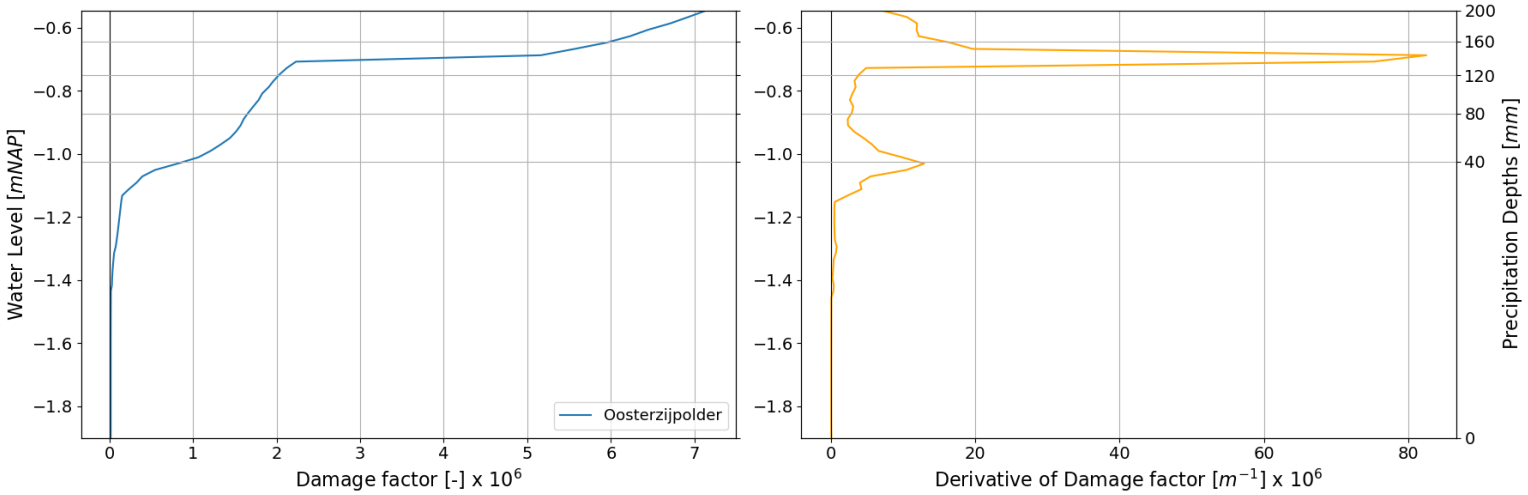
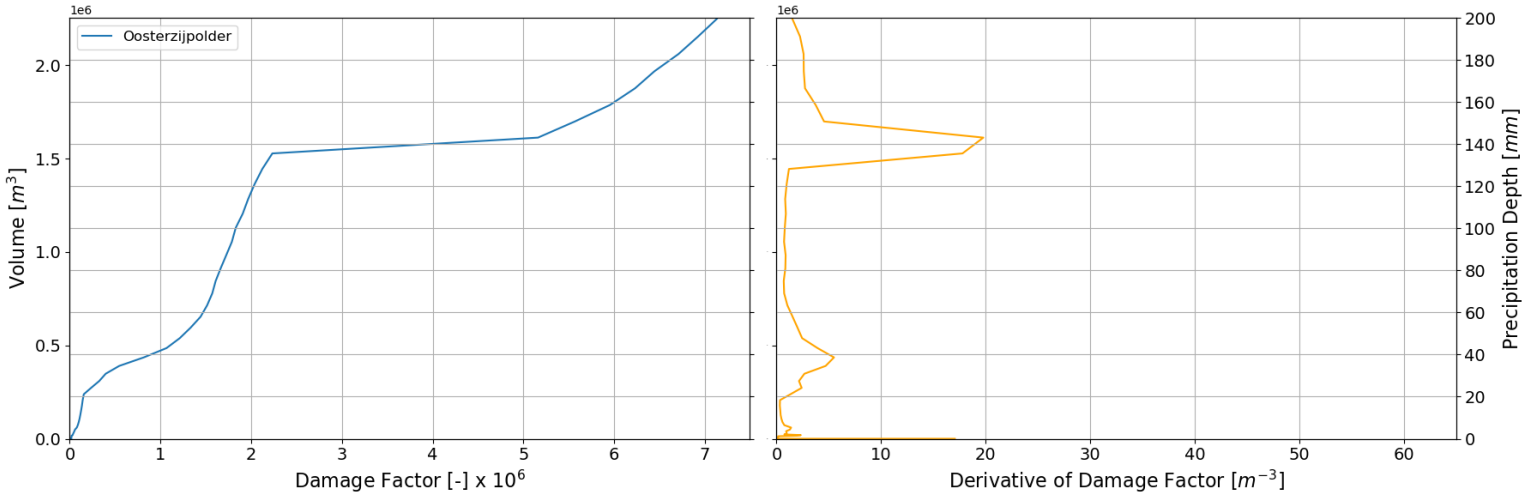


Table L.32: Oosterzijpolder characteristics

Oosterzijpolder			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
130	12.5		1127	106
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	78	1.64	0	0
Optimized	78	1.64		





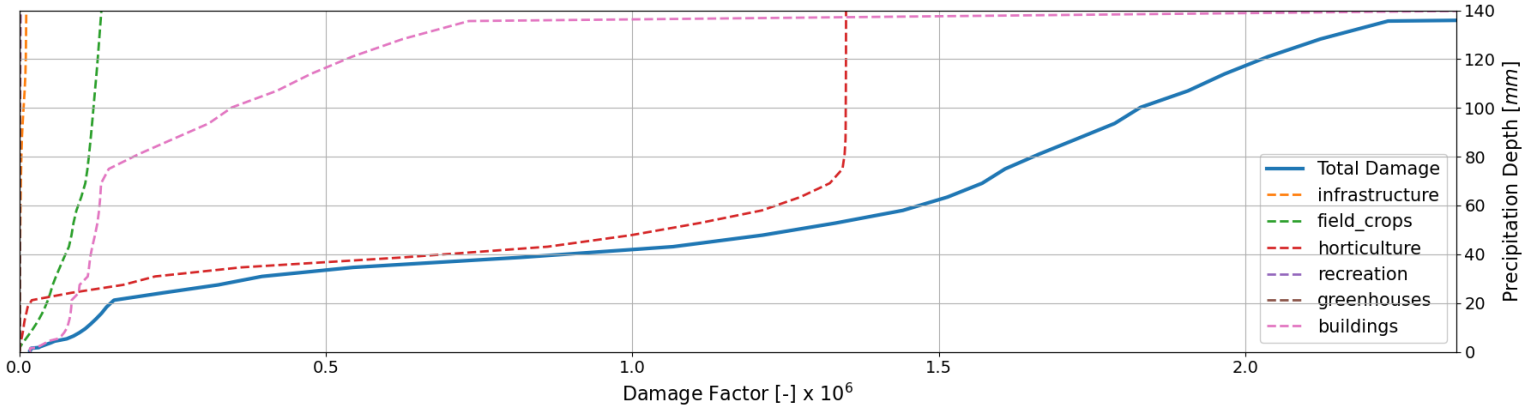
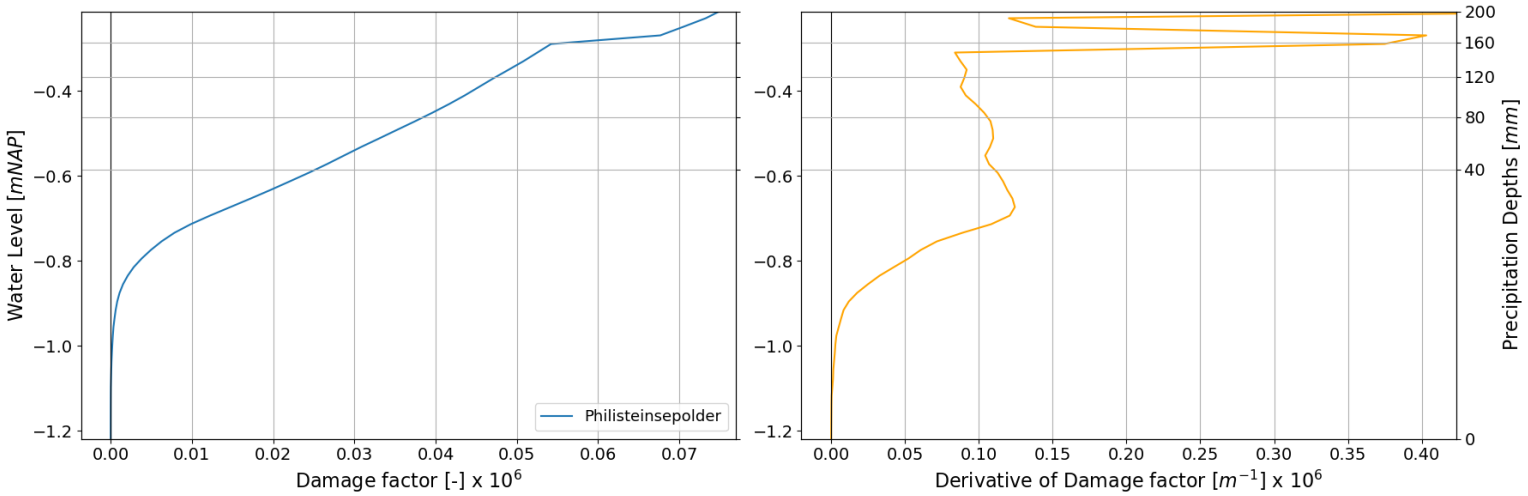
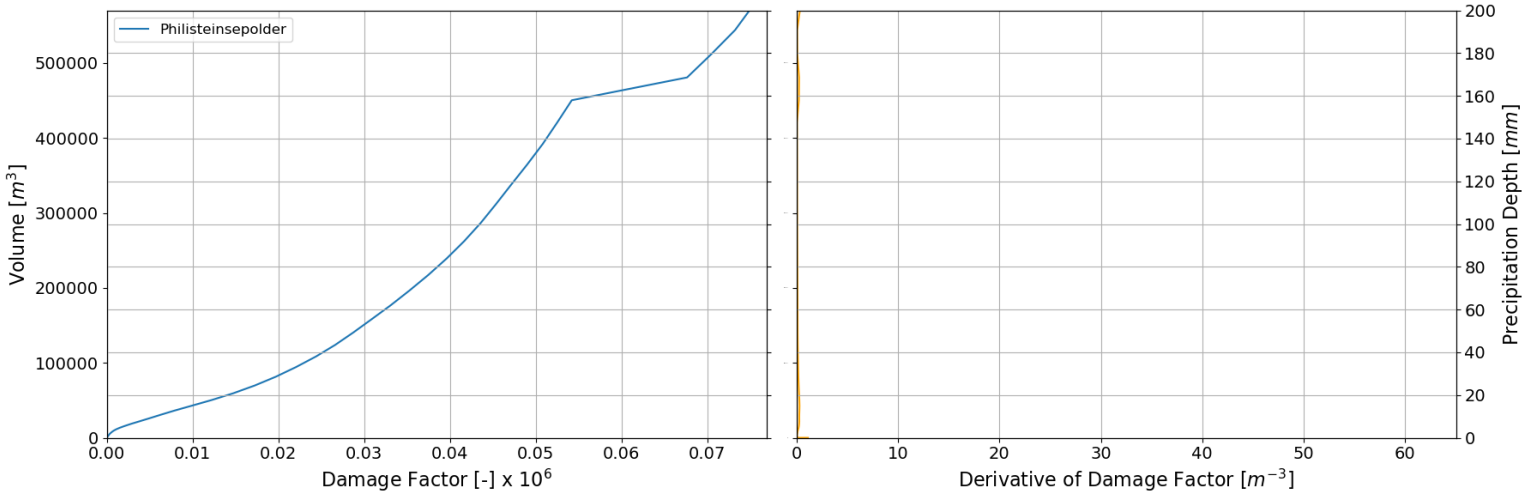


Table L.33: Philisteinsepolder characteristics

Philisteinsepolder			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	10.1		285	159
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	136	0.05	0	0
Optimized	136	0.05		





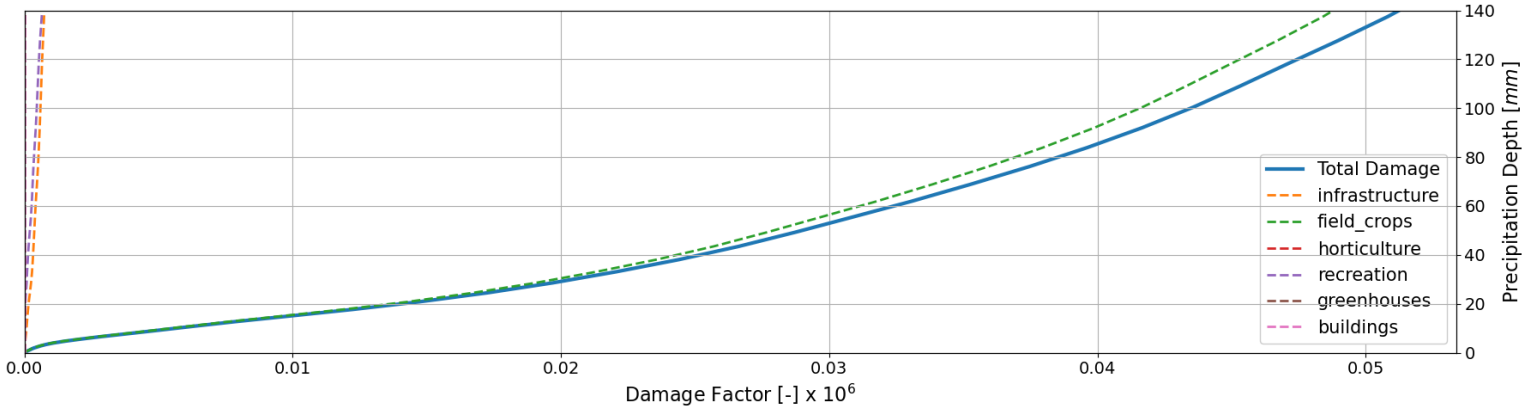
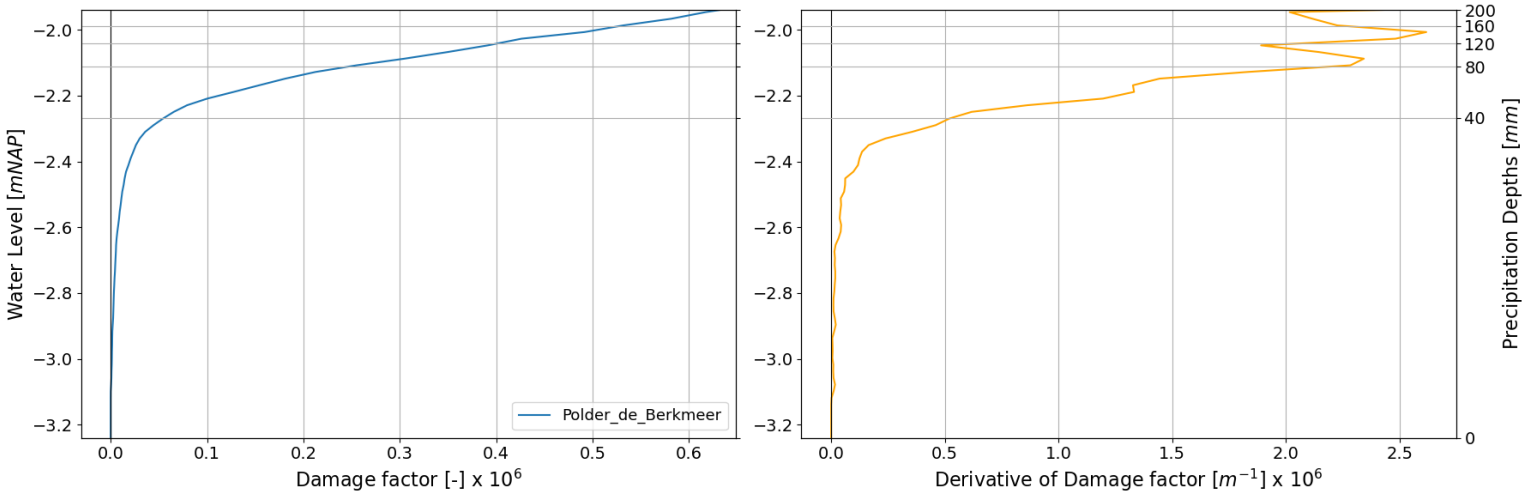
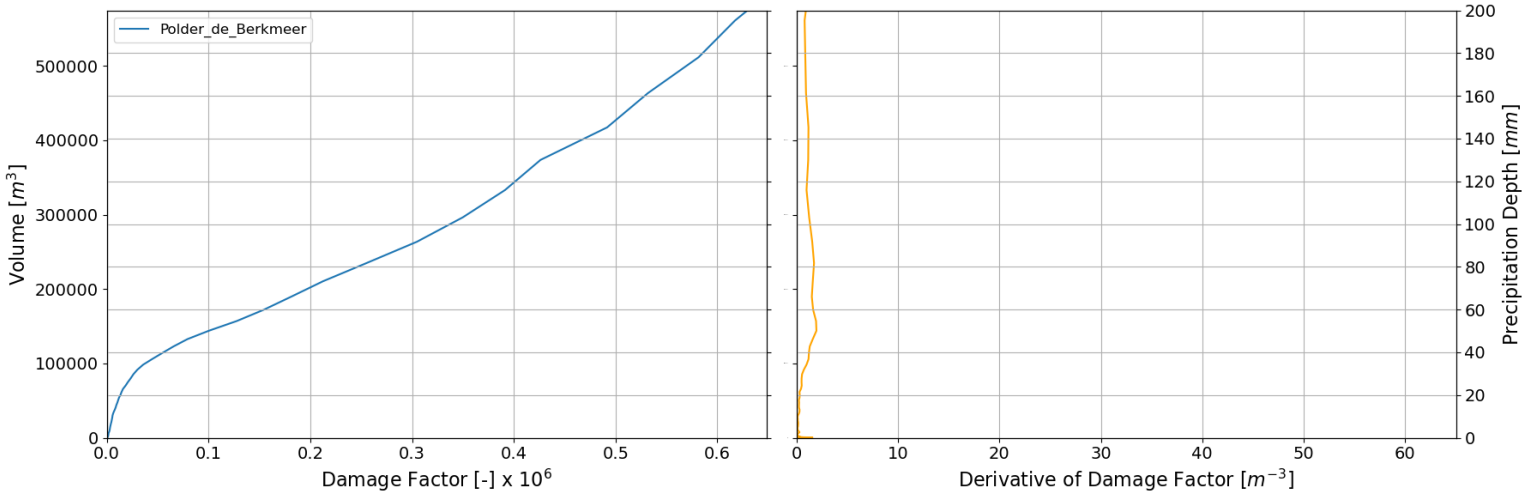


Table L.34: Polder de Berkmeer characteristics

Polder de Berkmeer			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	15.1		287	73
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	41	0.06	0	0
Optimized	41	0.06		





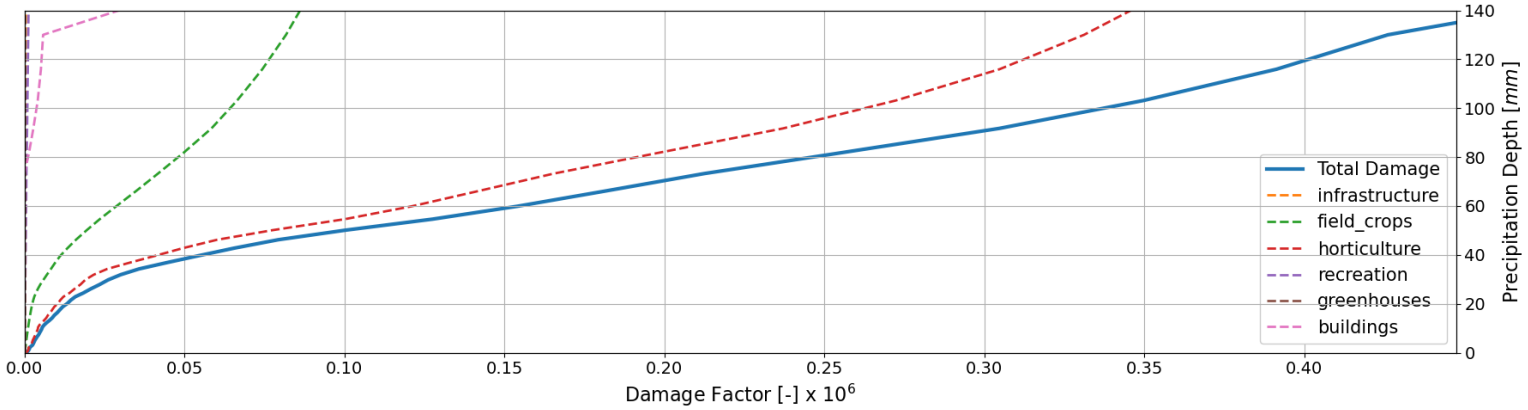
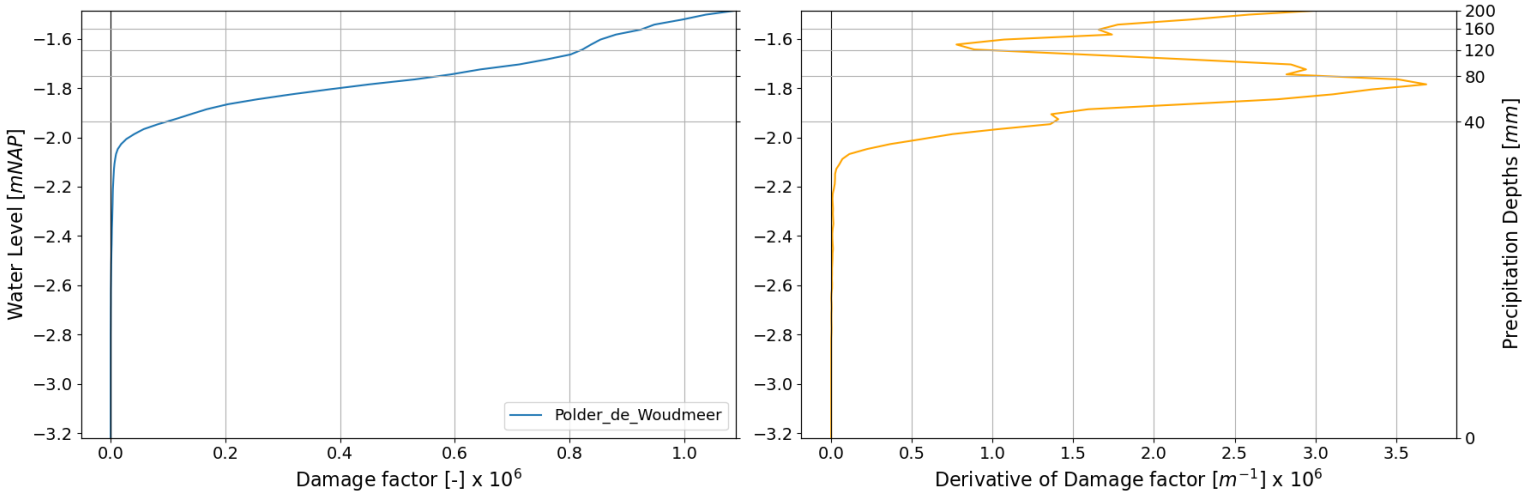
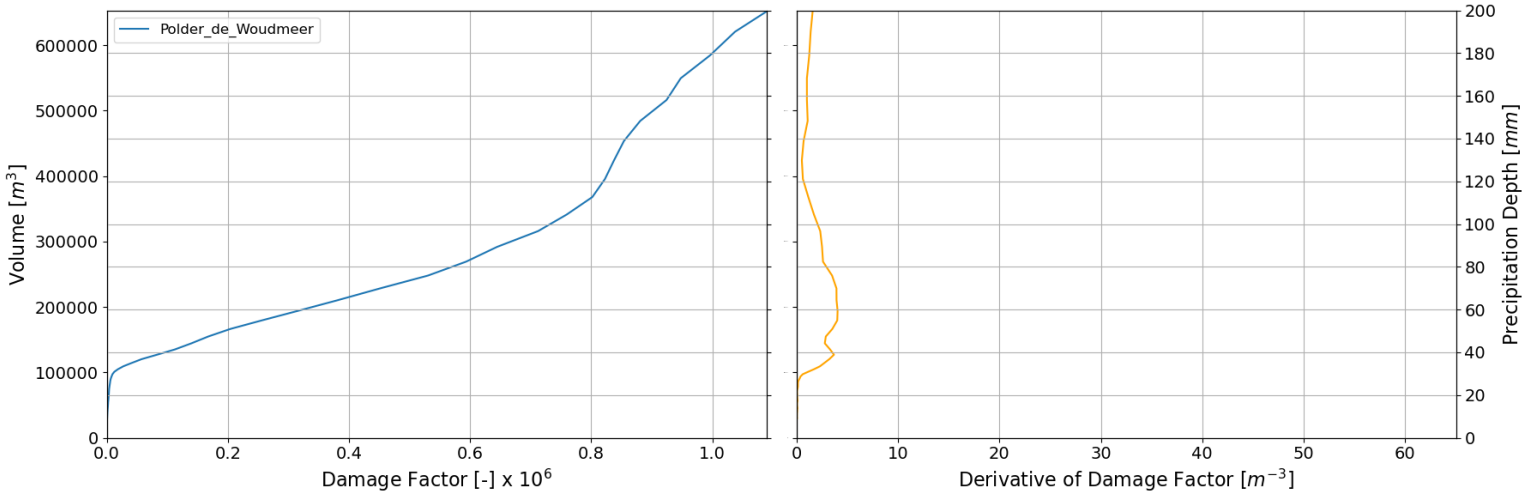


Table L.35: Polder de Woudmeer characteristics

Polder de Woudmeer			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	17.6		327	88
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	51	0.21	0	0
Optimized	51	0.21		





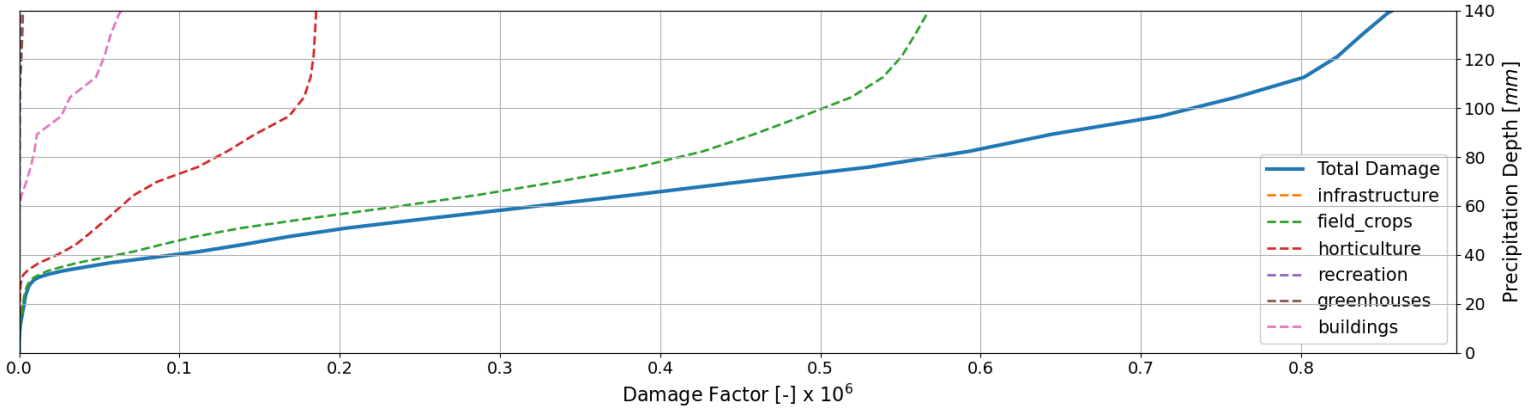
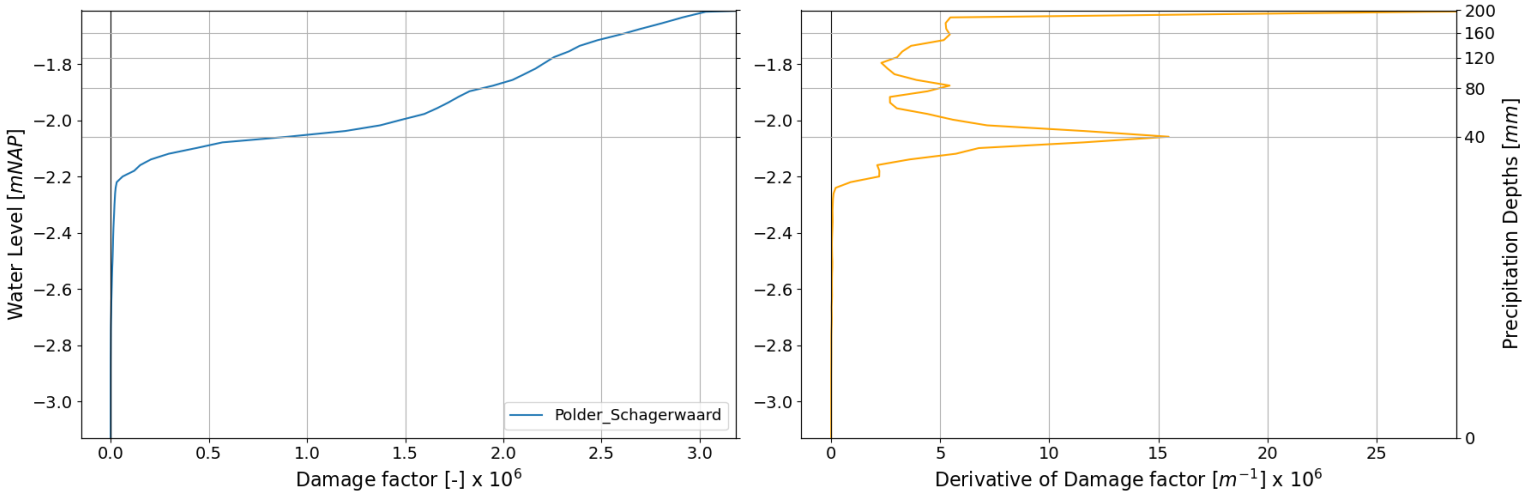
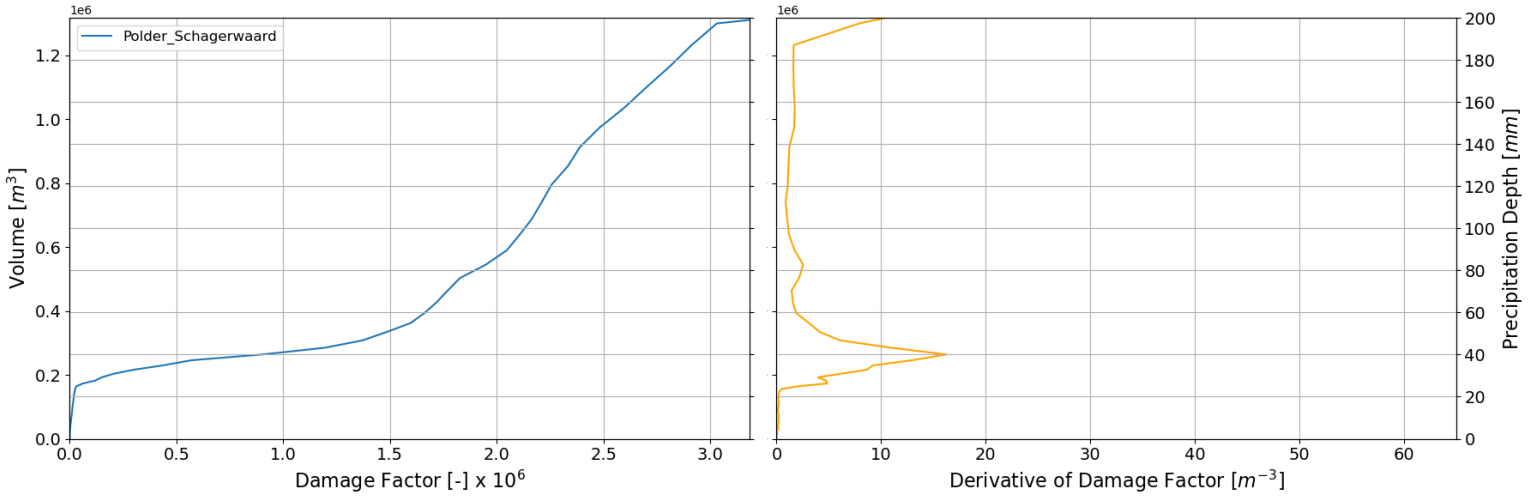


Table L.36: Polder Schagerwaard characteristics

Polder Schagerwaard			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	16.7		659	91
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	53.6	1.56	0.9	124560
Optimized	38.0	0.66		





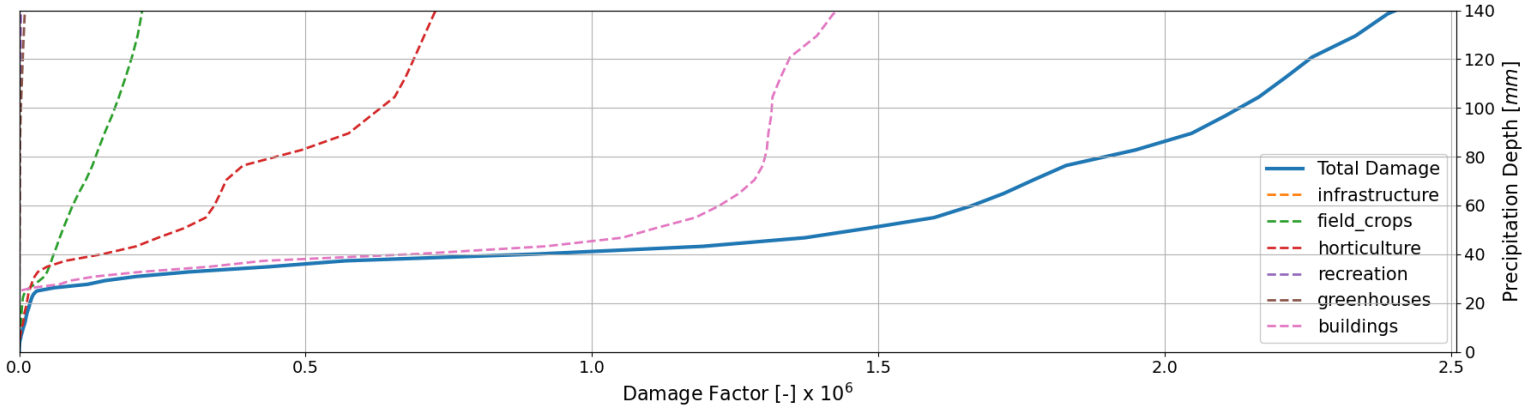
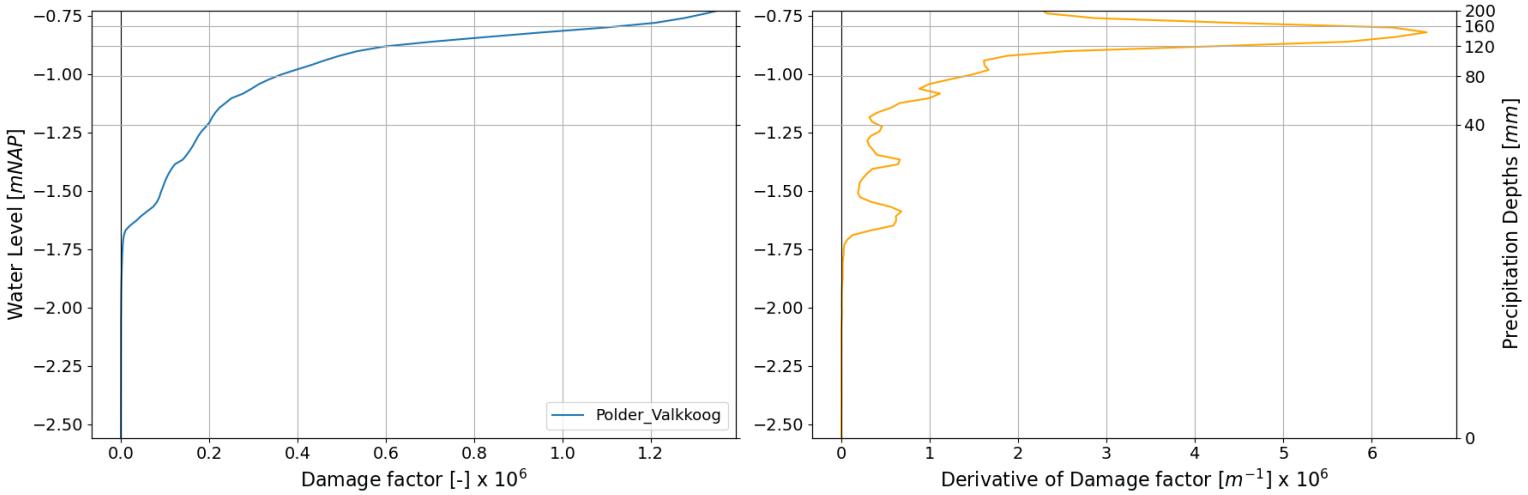
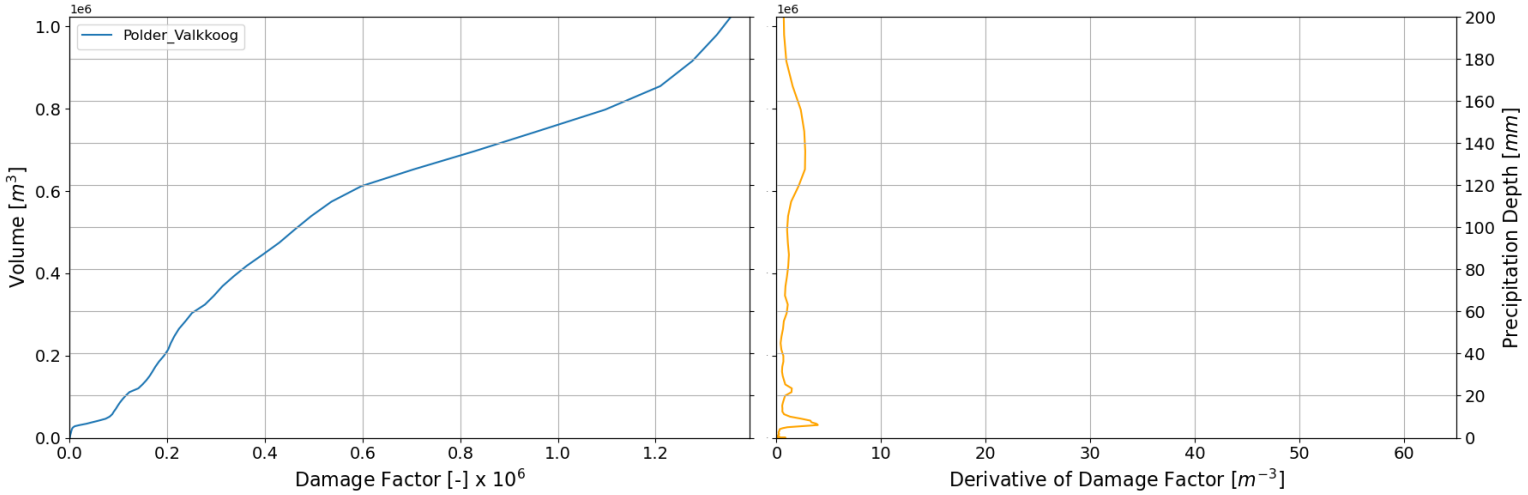


Table L.37: Polder Valkkoog characteristics

Polder Valkkoog			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	14.1		512	111
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	81	0.36	0	0
Optimized	81	0.36		





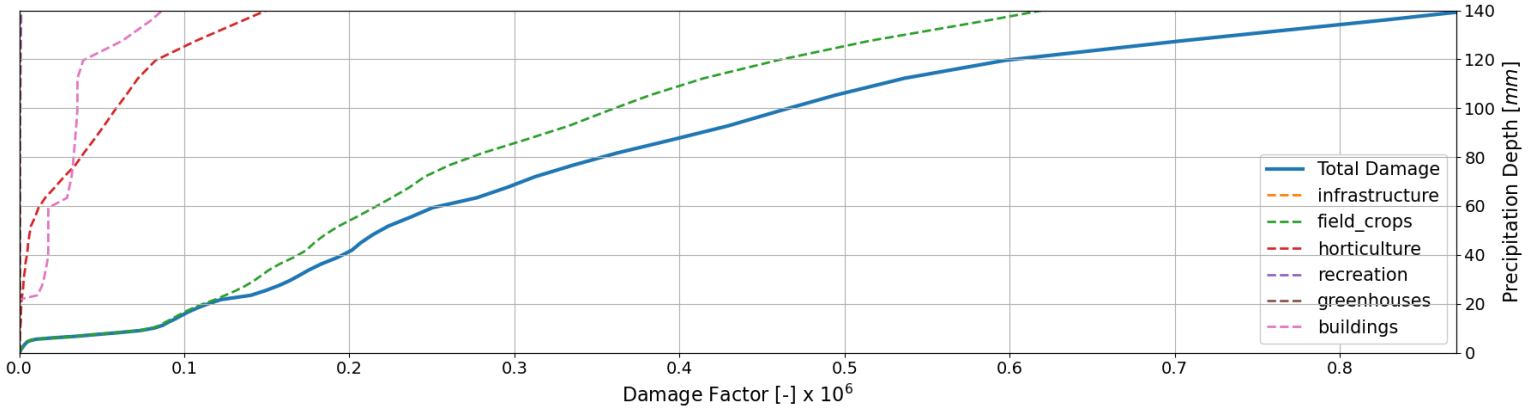
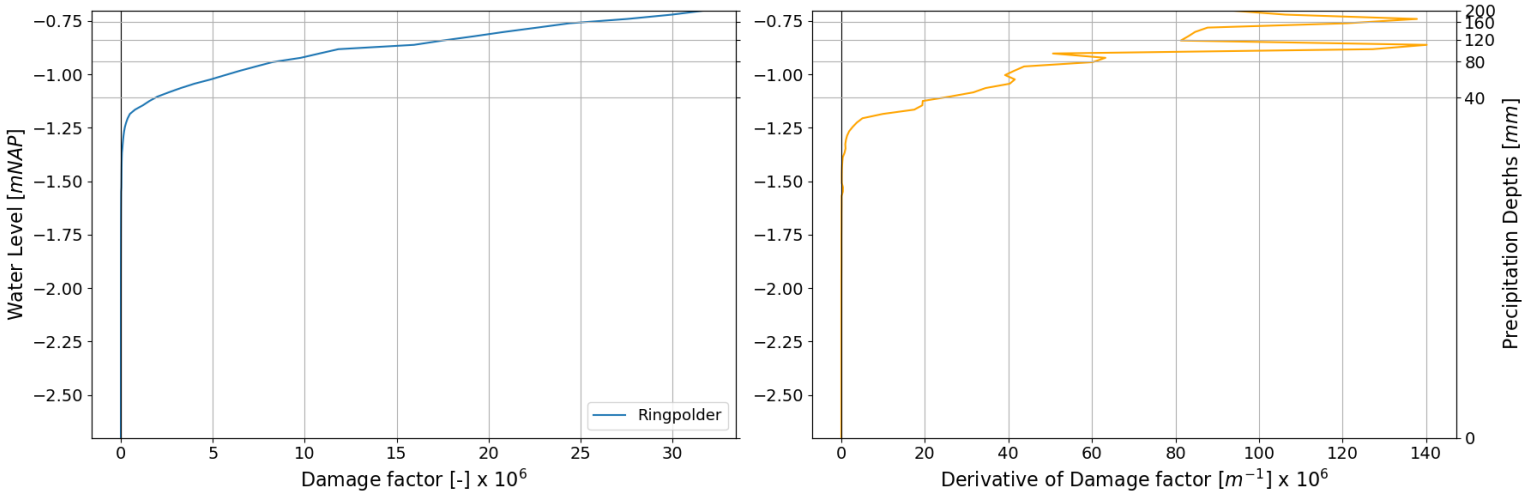
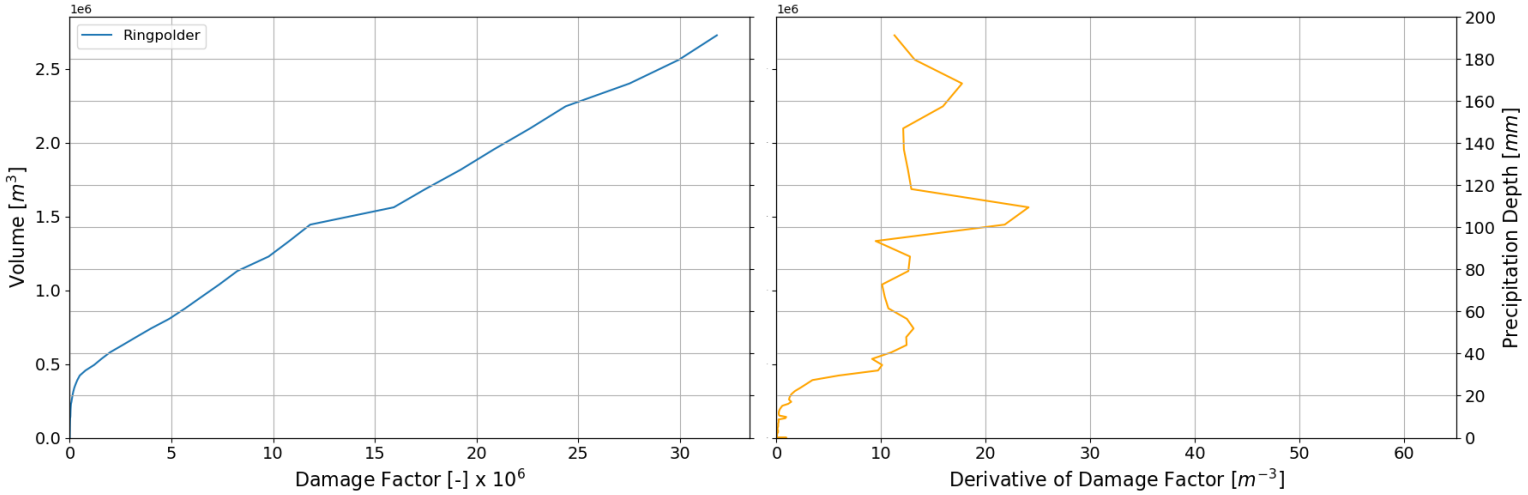


Table L.38: Ringpolder characteristics

Ringpolder			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	14.4		1425	115
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	82.8	9.04	3.282	321300
Optimized	62.0	5.76		





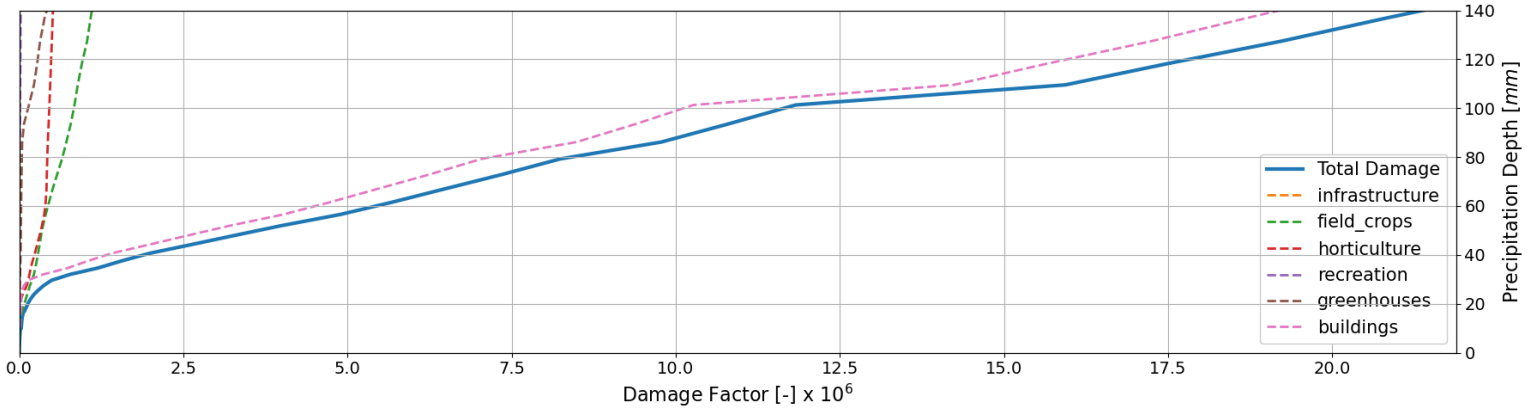
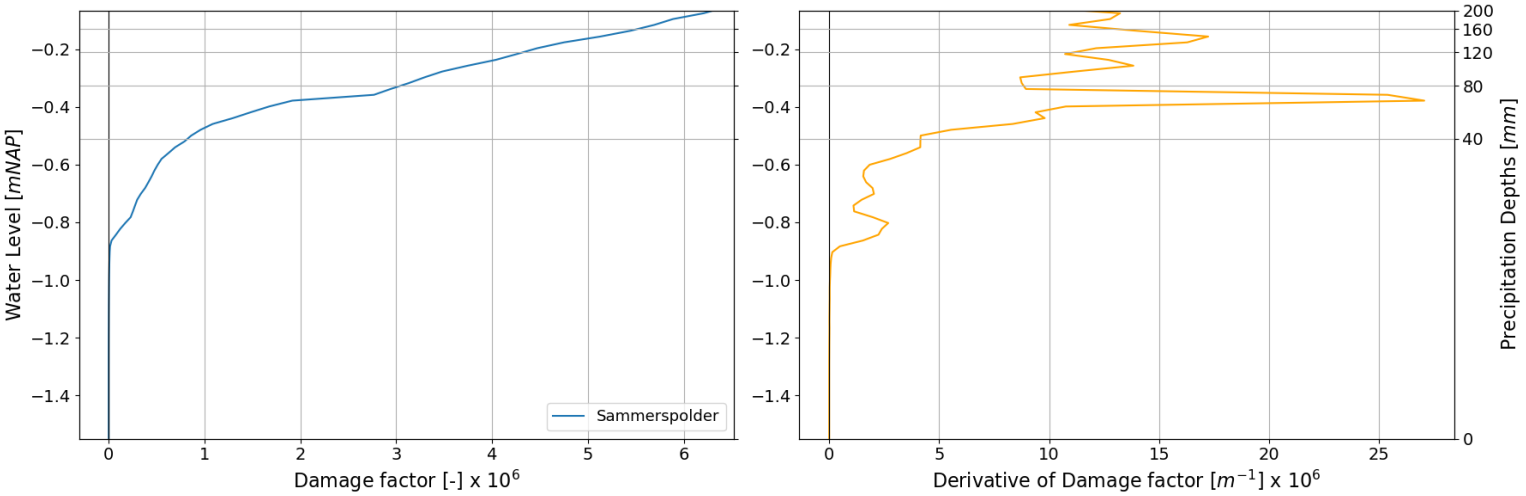
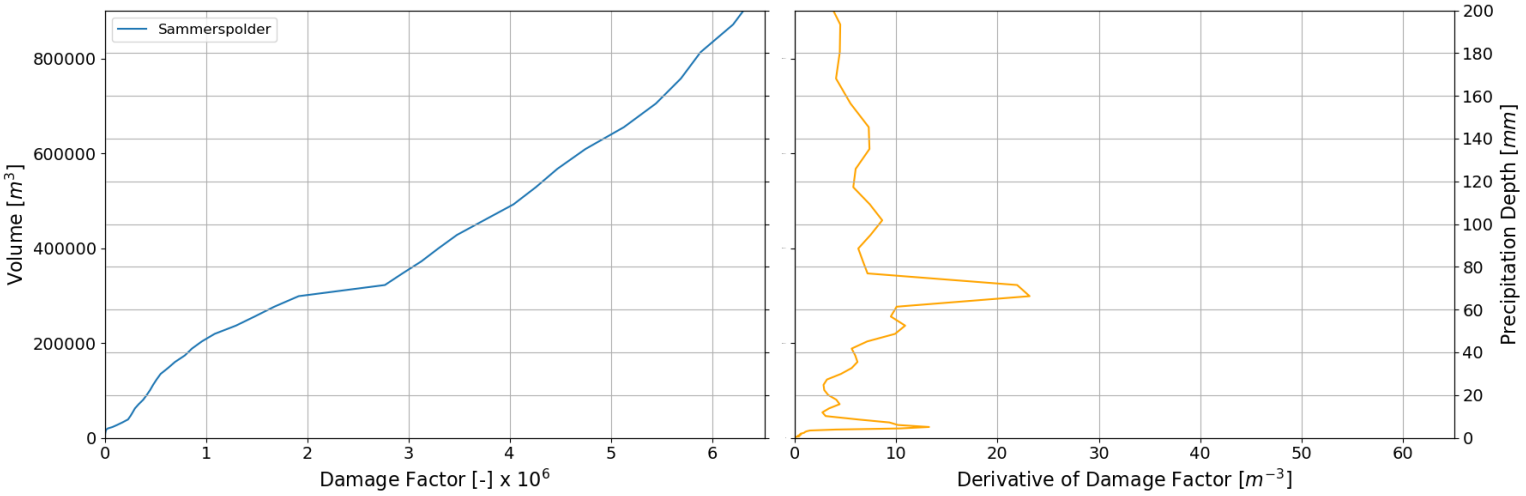


Table L.39: Sammerspolder characteristics

Sammerspolder			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	18.5		451	142
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	100.1	3.68	0.651	103680
Optimized	79.6	3.03		





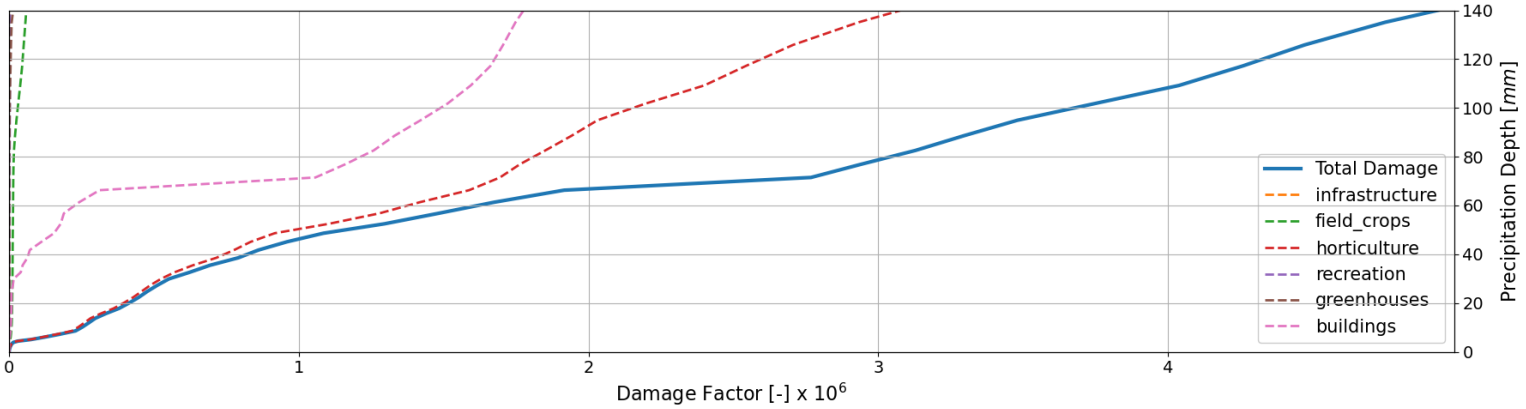
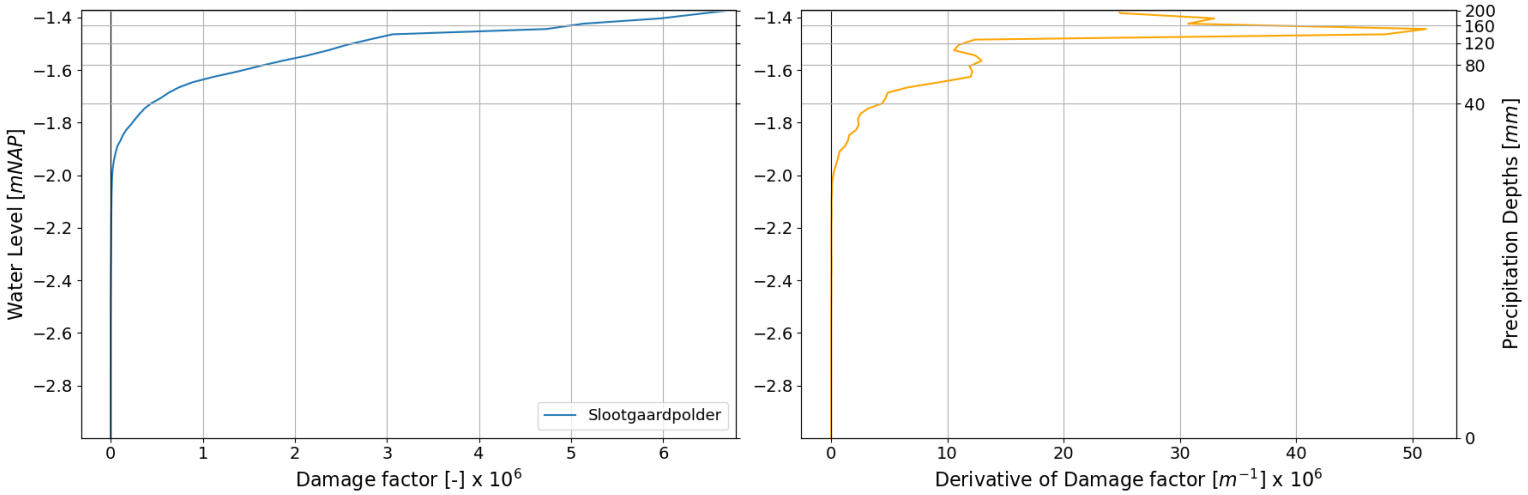
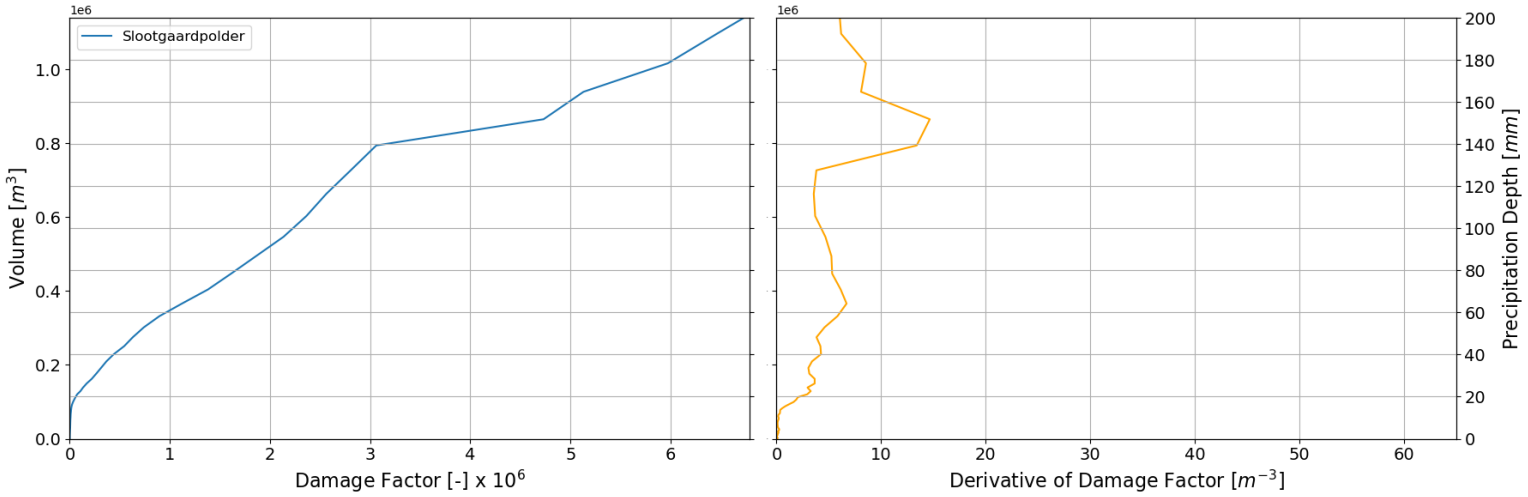


Table L.40: Slootgaardpolder characteristics

Slootgaardpolder			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	19.2		570	79
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	36	0.36	0	0
Optimized	36	0.36		





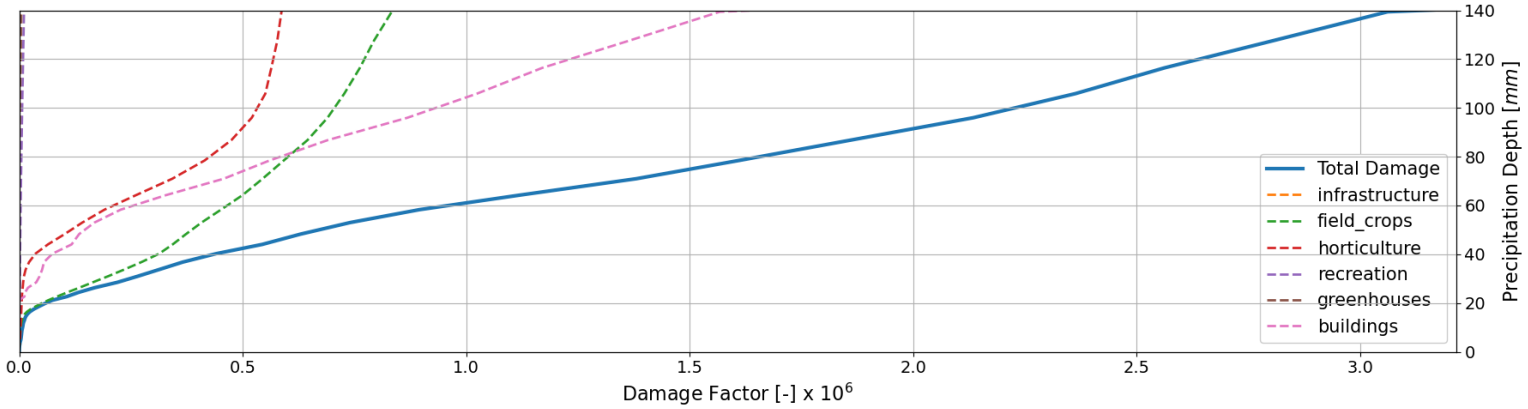
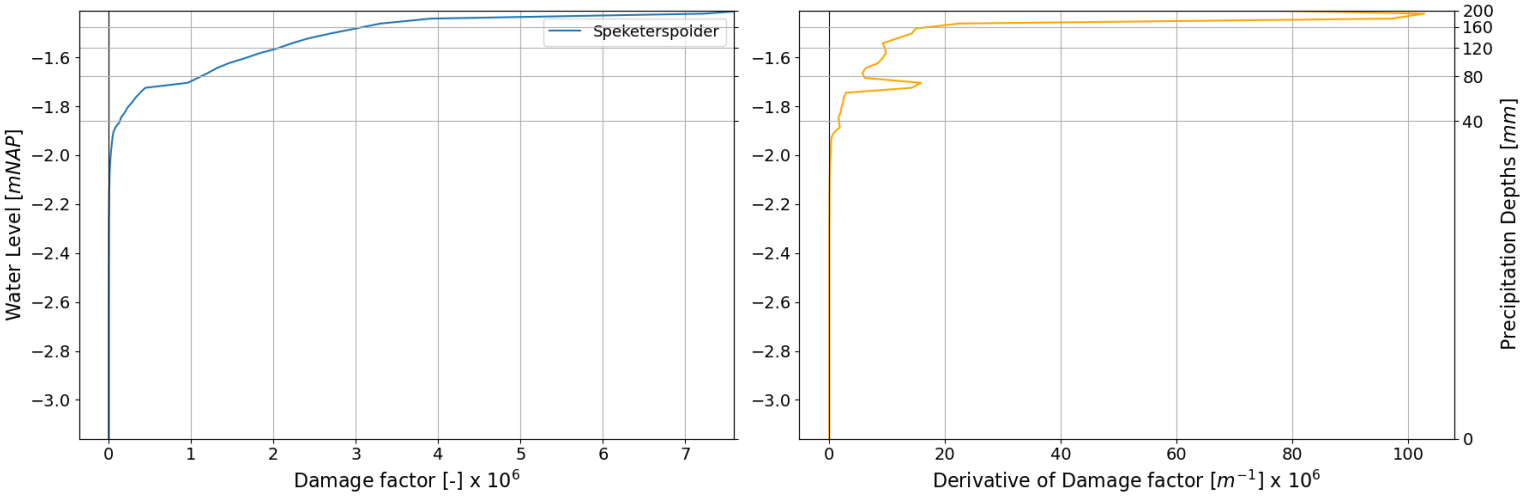
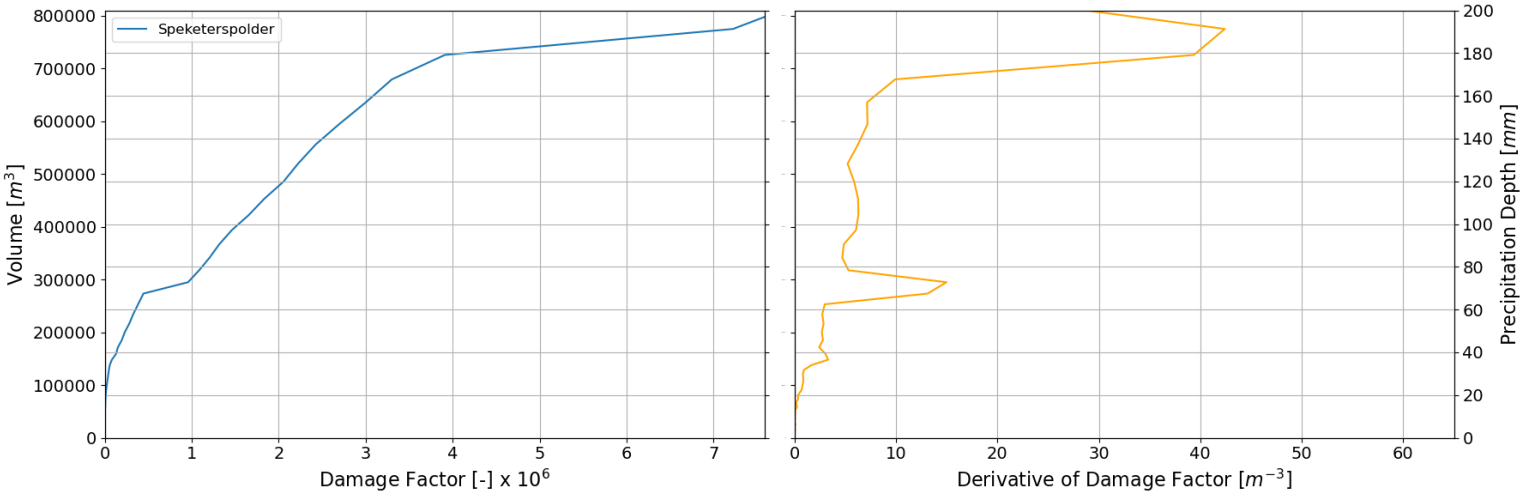


Table L.41: Speketerspolder characteristics

Speketerspolder			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	14.2		405	83
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	51	0.25	0	0
Optimized	51	0.25		





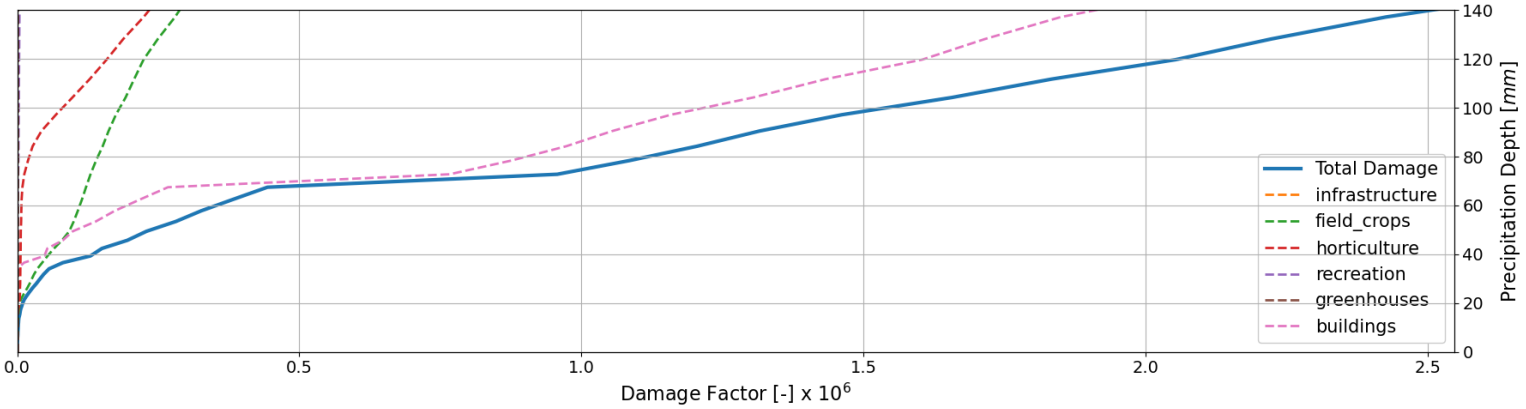
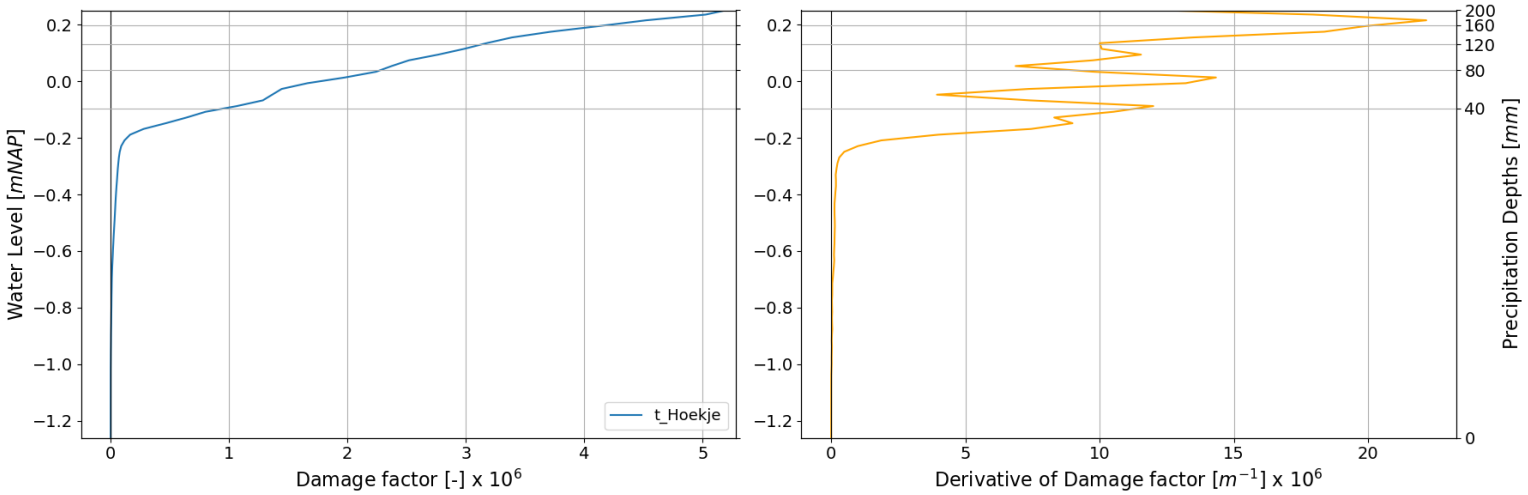
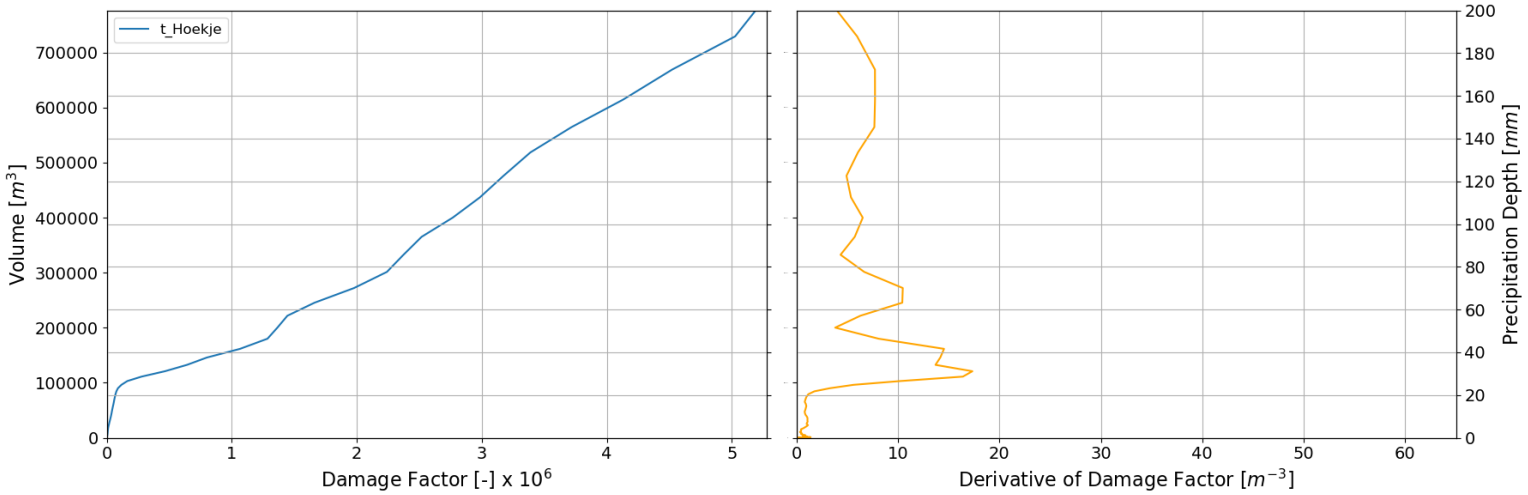


Table L.42: 't Hoekje characteristics

't Hoekje			Type 1	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	19.3		388	105
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	64.3	1.71	0.592	92520
Optimized	42.8	1.12		





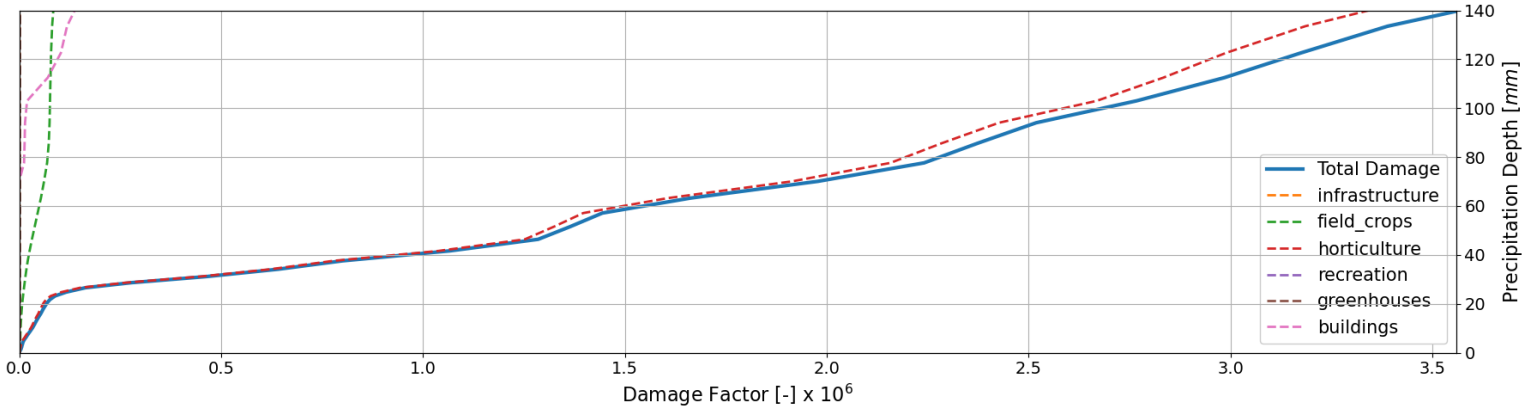
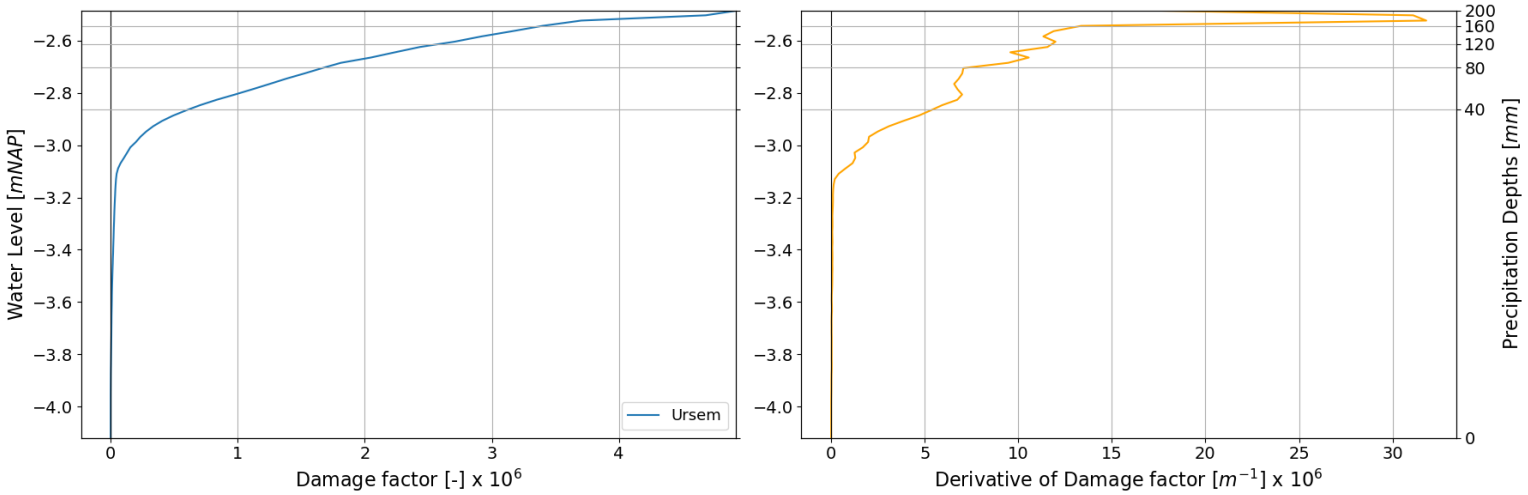
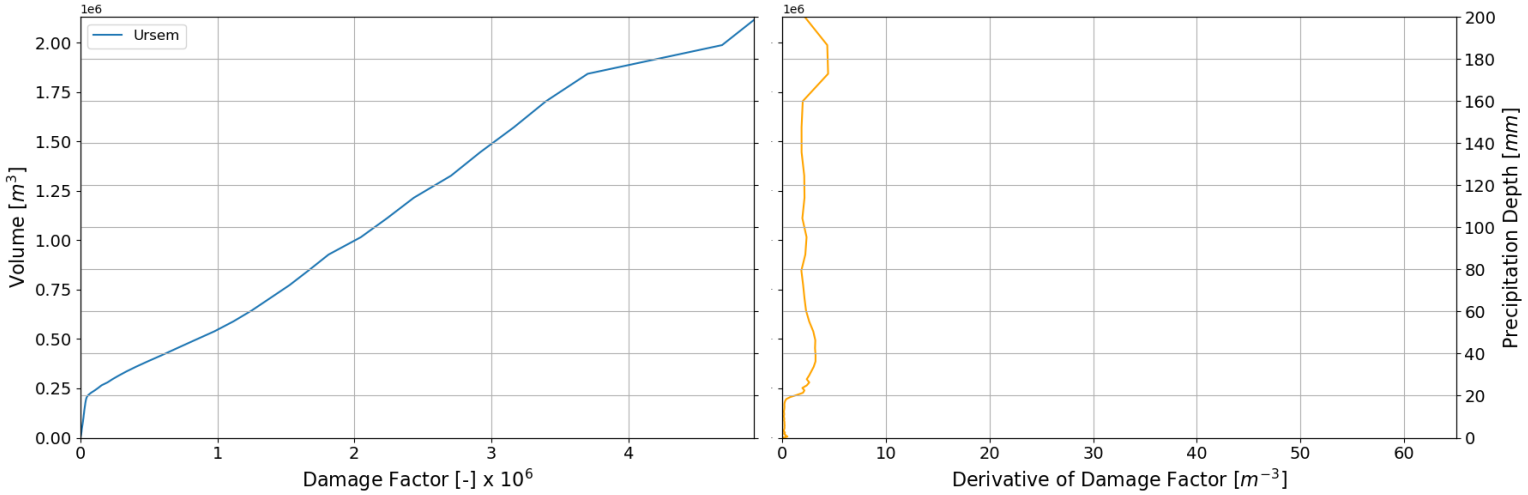


Table L.43: Ursem characteristics

Ursem			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	16.1		1065	57
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	57	0.11	0	0
Optimized	23	0.11		





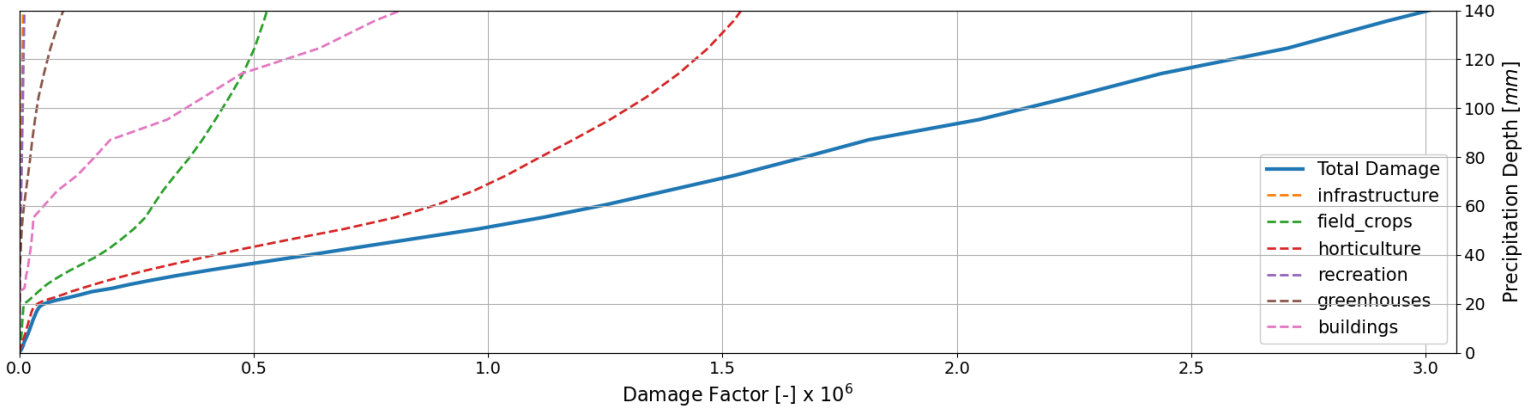
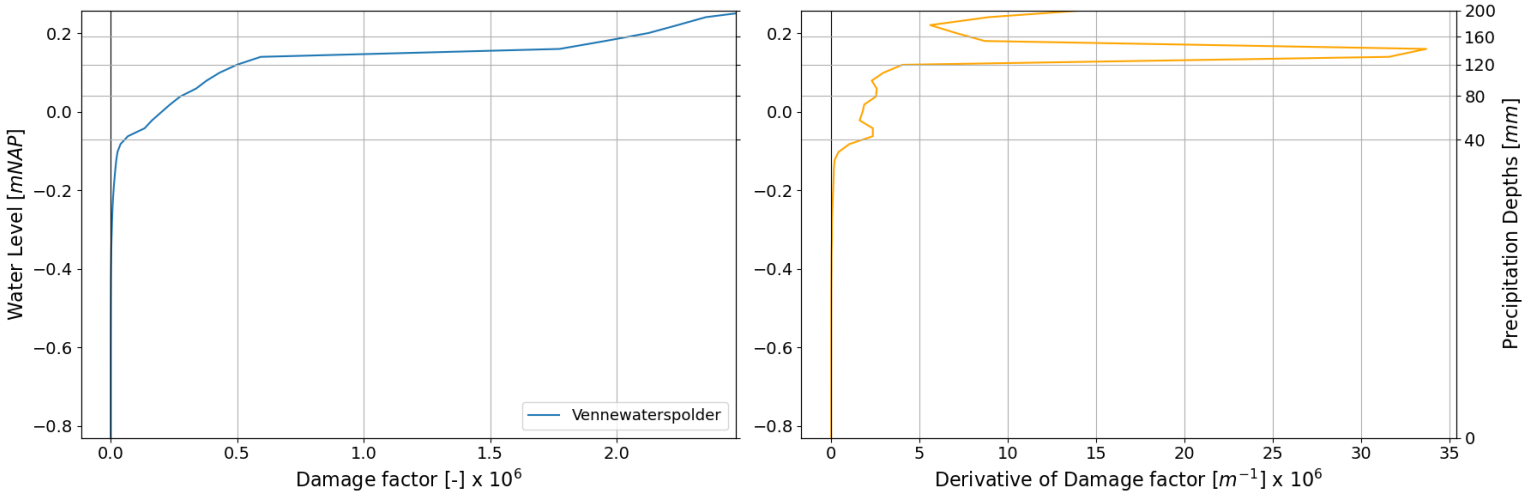
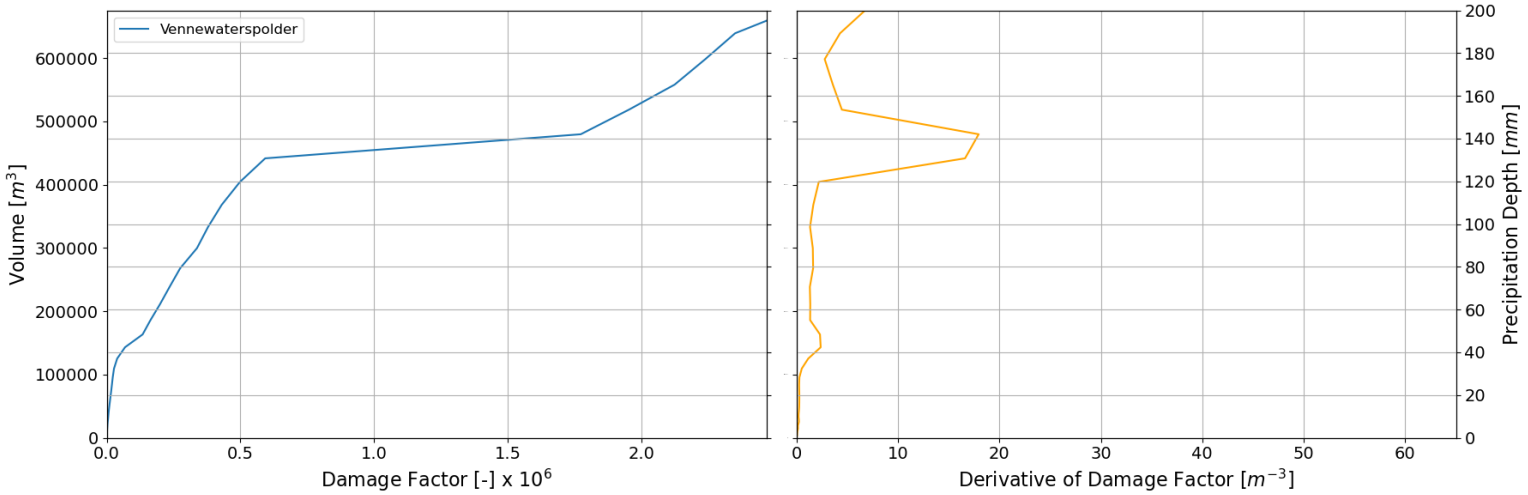


Table L.44: Vennewaterspolder characteristics

Vennewaterspolder			Type 2	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
120	13.6		338	130
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	99	0.38	0	0
Optimized	99	0.38		





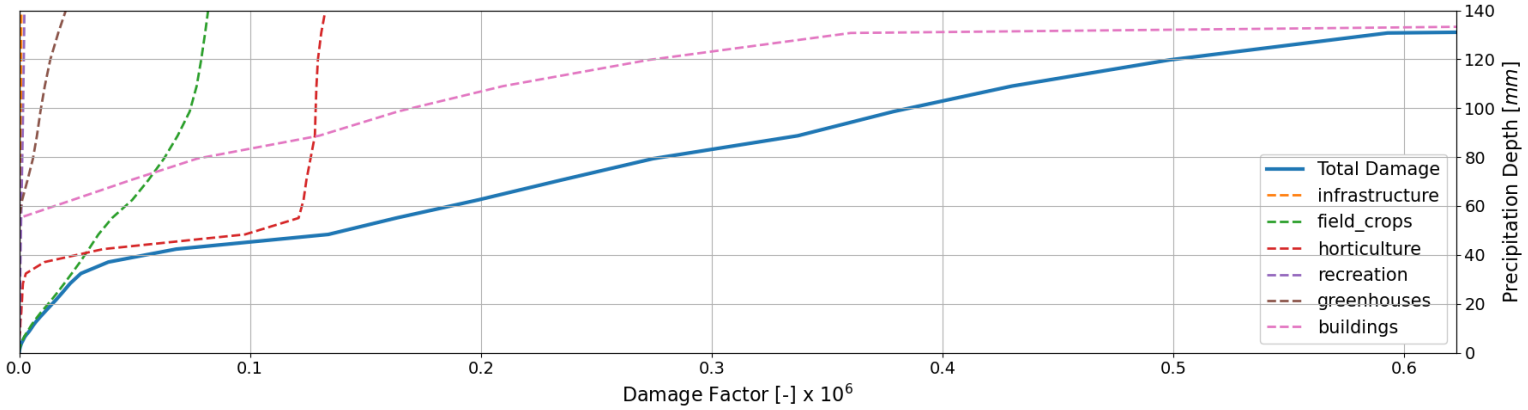
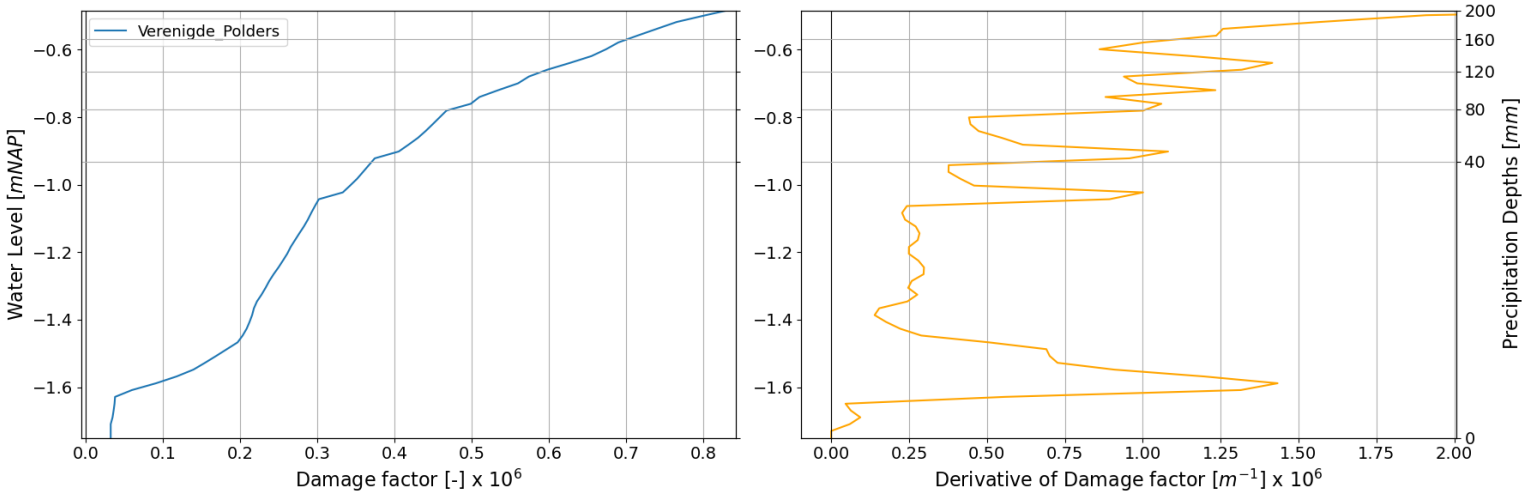
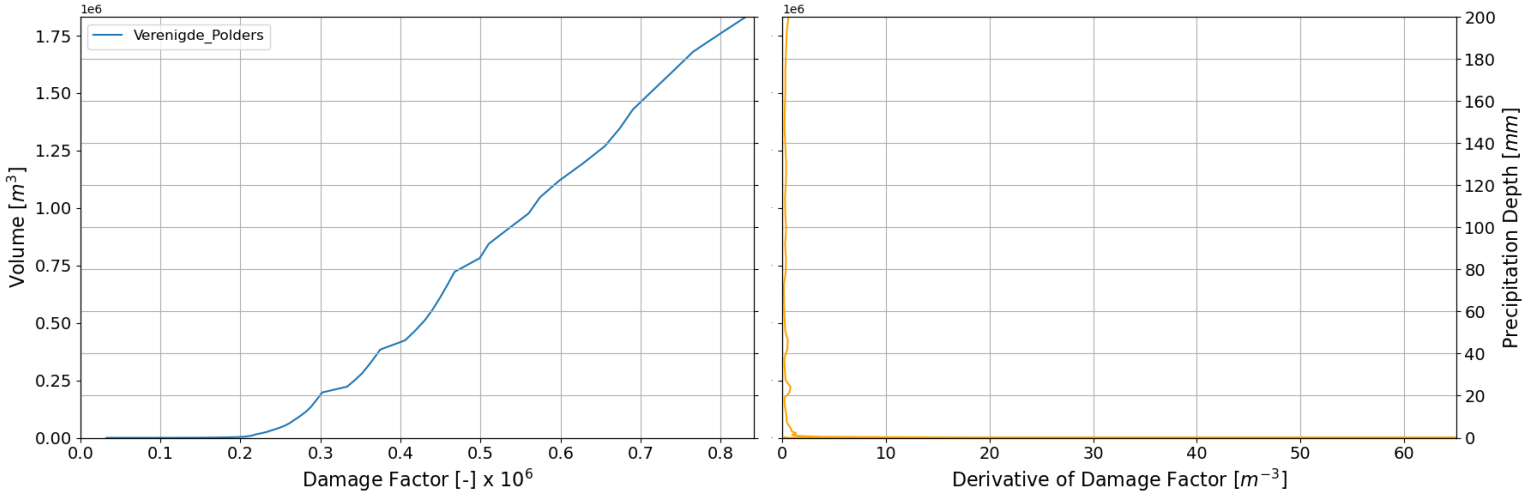


Table L.45: Verenigde Polders characteristics

Verenigde Polders			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	13.8		916	127
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	96	0.52	0	0
Optimized	96	0.52		





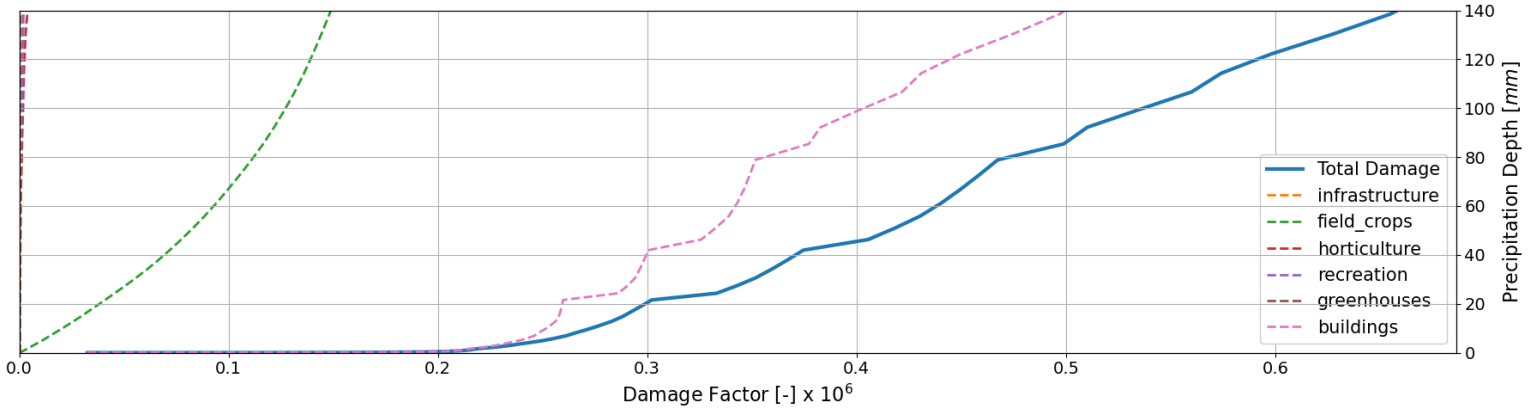
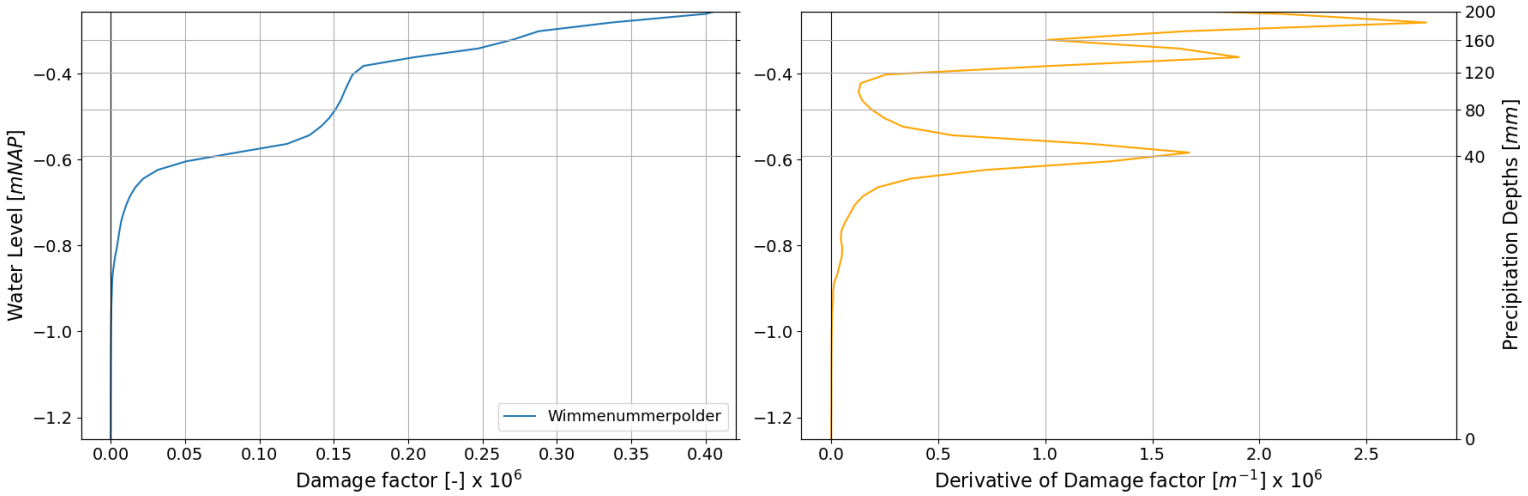
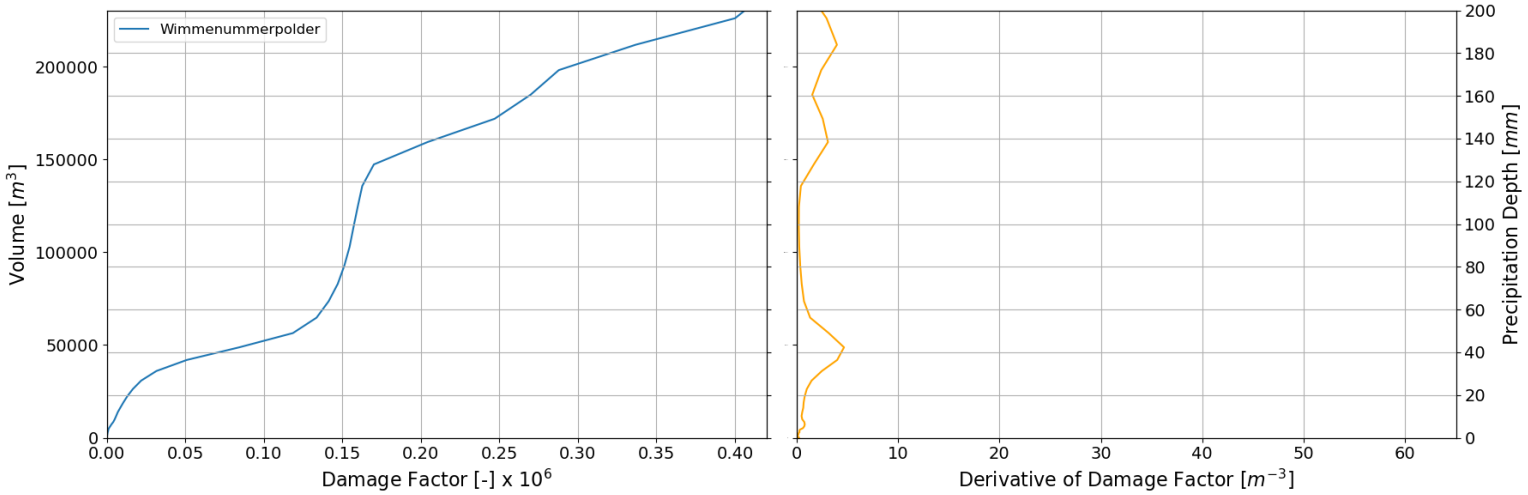


Table L.46: Wimmenummerpolder characteristics

Wimmenummerpolder			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	10.0		115	157
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	135	0.19	0	0
Optimized	135	0.19		





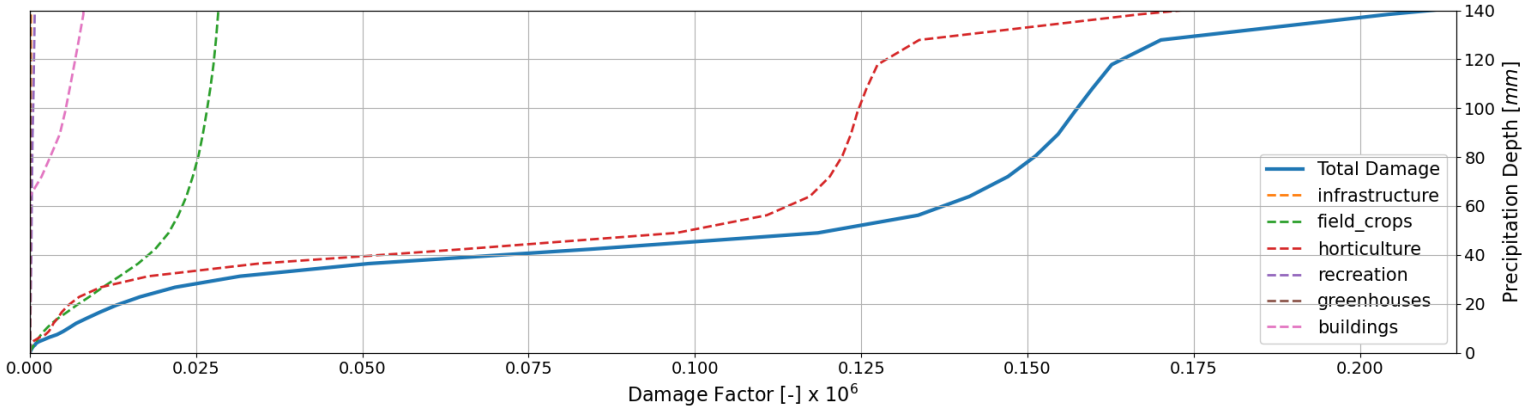


Table L.47: Wogmeer characteristics

Wogmeer			Type 3	
Storage Capacity [mm]	**Removal Capacity [mm/d]**		**Area [ha]**	**Precipitation [mm]**
0	13.5		691	56
	Maximum Volume [mm]	**Damage [m euro]**	**Damage Prevented [m euro]**	**Cumulative Pump Capacity [m3]**
Baseline	27	0.09	0	0
Optimized	27	0.09