FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Smart irrigation based on high performance computing: a new generation optimisation structure

Luís Manuel Ribeiro de Sousa



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Roman Chertovskikh

Second Supervisor: Prof. Nathalie Khalil

October 15, 2026

© Luís Sousa, 2026

Smart irrigation based on high performance computing: a new generation optimisation structure

Luís Manuel Ribeiro de Sousa

Mestrado em Engenharia Eletrotécnica e de Computadores

October 15, 2026

Resumo

Devido ao aquecimento global, a crescente frequência de períodos de seca torna a gestão eficiente dos recursos hídricos uma necessidade crítica, particularmente em setores com uso intensivo de água, como a agricultura. Em Portugal, o consumo anual de água é aproximadamente $7500 \times 10^6 \text{m}^3$, com agricultura sendo 80% deste total. Alarmantemente, aproximadamente 37.5% desta água é desperdiçada, especialmente durante a primavera e verão, quando a procura é mais elevada. Estes desperdícios agravam a escassez de água e aumentam a importância de gestão eficiente de recursos hídricos. Tendo em conta estes desafios, o desenvolvimento de métodos de rega eficazes torna-se essencial. Este trabalho contribui para responder a este problema, propondo soluções para a redução do desperdício de água na agricultura, assegurando uma utilização mais sustentável da água face às alterações climáticas. Explora a viabilidade do desenvolvimento de um sistema de rega inteligente, baseado na aplicação de um algoritmo de optimização *bilevel*, utilizando também ferramentas digitais para validação e verificação dos resultados obtidos. Construindo sobre trabalho anteriormente feito na área em que o problema foi modelado como um problema de optimização de nível único, um segundo nível foi adicionado, de maneira que o modelo consiga considerar vários campos e também uma quantidade limitada de água por intervalo de tempo, que será posteriormente distribuída da maneira mais eficiente, de modo a garantir danos mínimos nas culturas no caso de rega insuficiente. Implementado na linguagem de programação Julia, a escalabilidade e o desempenho da *framework* foram validados utilizando o *cluster* Deucalion de computação de alto desempenho (High-Performance Computing), demonstrando a sua viabilidade para aplicações agrícolas de larga escala.

Palavras-chave: optimização bi-nível • rega inteligente • computação de alto desempenho • controlo ótimo

Abstract

Driven by global warming, escalating global droughts have made efficient water management a critical necessity, particularly in water-intensive sectors like agriculture. In Portugal, the annual water consumption is approximately $7500 \times 10^6 \mathrm{m}^3$, with agriculture accounting for 80% of this total. Alarmingly, around 37.5% of this water is wasted, especially during spring and summer, when demand is highest. This wastage exacerbates water shortages, underscoring the need to focus on efficient water management. Given these challenges, developing efficient irrigation methods is essential. This work contributes to addressing this issue by proposing solutions to reduce water waste in agricultural practices, ensuring more sustainable water usage in the face of climate change. It explores the viability of developing a smart irrigation control system based on a bilevel optimisation algorithm, and of using digital tools to validate and verify the results. Building upon previous single-level optimisation models, this work introduces a second optimisation level. This extension allows the model to account for multiple fields and limited water availability per time interval, which are then distributed optimally to minimise crop damage in the event of insufficient irrigation. The framework, implemented in Julia, was validated for scalability and performance on the Deucalion High-Performance Computing cluster, demonstrating its viability for large-scale agricultural applications.

Keywords: bilevel optimisation • smart irrigation • high performance computing • optimal control

The United Nations Sustainable Development Goals

This dissertation contributes to several of the United Nations Sustainable Development Goals (SDGs), particularly through its focus on improving water-use efficiency in agriculture and developing new computational methods to support that goal. Table 1 summarises the specific targets addressed by this work, along with the corresponding contributions and performance indicators.

SDG	Target	Contribution	Performance Indicators and Metrics
2	2.4	Develop a bilevel optimisation framework that supports controlled deficit irrigation under water scarcity, allowing crop water requirements to be met as closely as possible even when total supply is constrained, rather than relying on rigid, infeasible thresholds.	Reduction in soil moisture deviation from agronomic targets under constrained water supply; feasibility of the optimisation under scenarios that would be infeasible in the single-level formulation.
6	6.4	Contribute methods for improving water-use efficiency in agricultural irrigation by formulating water allocation across fields as a strategic decision rather than a reactive correction, reducing unnecessary water consumption while maintaining crop safety.	Total irrigation water applied across test scenarios; comparison of water use efficiency between the single-level baseline and the bilevel extension.
9	9.5	Contribute new computational methods and software implementations for solving bilevel optimisation problems in an agricultural context, including a validated Julia implementation and a comparative study of solver configurations for the resulting MPEC reformulation.	Original implementation and solver methodology; reliability and performance of the developed solver configurations across tested problem sizes.

Table 1: Contribution of this dissertation to the United Nations Sustainable Development Goals.

Agradecimentos

Para começar este documento que é o fruto do trabalho dos anos que passei até agora nesta universidade, gostaria de começar por deixar umas palavras a todos os que tornaram isto possível e que me acompanharam ao longo destes anos na FEUP.

Começo por agradecer a todos os professores que ensinaram. A universidade é, acima de tudo, um lugar de aprendizagem, e foram os professores que me educaram nas várias áreas da engenharia e me deram os conhecimentos que eu cá vim para aprender. Em particular, gostaria de agradecer aos meus orientadores, o Professor Roman Chertovskikh e a Professora Nathalie Khalil, que me guiaram no processo de elaboração desta dissertação, apoiando-me com todas as minhas dúvidas e questões, e estando sempre disponíveis para me esclarecer nas várias dificuldades que foram surgindo ao longo de todo o processo.

Quero também agradecer aos meus pais, por me terem apoiado ao longo dos meus anos na universidade. Por terem visto o seu filho a ir viver sozinho para outra cidade, e por terem confiado sempre nas decisões que eu fiz. Por me incentivarem a tirar o máximo proveito de todos estes anos cá, e por conhecerem sempre alguém que arranje o que é preciso, seja isso um tronco, um armário ou 100 doses de leite creme. Foram a logística por trás da estratégia.

Estendo também estas palavras à Pepa. Aos meus amigos com que, apesar de todas as migrações para cidades e ilhas distantes, posso sempre contar em encontrar duas vezes por ano. Quando me afasto do Porto e vou para a minha terra, sei sempre que posso contar com vocês e com os berbequins, melancias e filmes búlgaros a preto e branco.

Quero terminar por mencionar todos os que tornaram estes últimos anos excepcionais. Todas as pessoas que, cada uma da sua forma, tornaram esta universidade algo que me vai deixar muita saudade. A todos os membros das atividades académico-recreativas, que de uma maneira ou outra, acabaram todos por deixar a sua marca no tempo que eu estive aqui. A Electro 021, por me ter apoiado quando eu precisei e não me ter deixado para trás. Certamente não teria chegado a este ponto sem toda a ajuda que me deram na I101 no cantinho de auto. Mas, sobretudo, estas palavras são dirigidas a Inph. Os que desde o início, estiveram sempre comigo. Que me acompanharam desde que entrei na universidade, me viram crescer, e agora me veem a sair. Ombro a ombro, subimos escadas e descemos montanhas, quer aqui, quer no outro lado do planeta. Que este agradecimento vos sirva pelos que a tradição não me deixou dar.

Para todos vós. Obrigado,

Luís Manuel

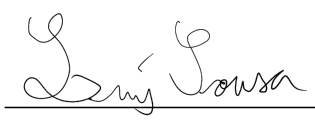
Declaration of honour

I declare that this dissertation is my own work and has not previously been submitted or assessed at this or any other institution. References to other authors (statements, ideas, thoughts), as well as the use of published works, strictly comply with the rules of attribution and are duly identified in the text and in the bibliographic references, in accordance with the applicable referencing standards. I am aware that the practice of plagiarism or self-plagiarism constitutes an academic offence, as established in the U.Porto Code of Ethics.

I further declare that I have included text(s) published in co-authorship, and that I have specified, with transparency and rigour, the contribution of each co-author, as well as stated whether there exists any other dissertation in which the same text(s) may also have been included. The authorisation from the journal in which the work is published is also included, together with the express agreement signed by all co-authors authorising the inclusion of the article.

I also declare that I have used Artificial Intelligence tools to complete part(s) of this dissertation, and that this use and its results are duly noted and have been carefully checked for errors, inaccuracies, unfounded attributions or other forms of bias in content and arguments, as well as determining authorship.

Luís Manuel Ribeiro de Sousa



"There is nothing impossible to him who will try"

Alexander the Great

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 2
- 1.3 Objectives . . . 3
- 1.4 Dissertation Structure . . . 4

2 Background and Literature Review **5**
- 2.1 Optimisation Problems . . . 5
  - 2.1.1 Definition . . . 5
  - 2.1.2 Classification . . . 6
  - 2.1.3 Solving Optimisation Problems . . . 7
  - 2.1.4 Optimal Control Problems . . . 10
- 2.2 Smart Irrigation as an Optimal Control Problem . . . 11
  - 2.2.1 Replan and Redistribution Algorithm . . . 13
  - 2.2.2 Simulation Results and Validation . . . 14
  - 2.2.3 Overall Impact and Limitations . . . 15
- 2.3 Bilevel Optimisation . . . 15
  - 2.3.1 Definition . . . 16
  - 2.3.2 Optimistic Position . . . 17
  - 2.3.3 Pessimistic Position . . . 18
  - 2.3.4 Solution Methods . . . 19
- 2.4 Bilevel Optimisation in Irrigation Water Management . . . 19
- 2.5 Summary . . . 20

3 Single Level Implementation **21**
- 3.1 Tools . . . 21
  - 3.1.1 Julia and JuMP . . . 21
  - 3.1.2 Parallelisation . . . 23
- 3.2 Implementation . . . 23
- 3.3 Validation . . . 25
- 3.4 Summary . . . 30

4 Bilevel Implementation **31**
- 4.1 Tools . . . 31
  - 4.1.1 BilevelJuMP . . . 31
  - 4.1.2 Linear Solvers . . . 32
  - 4.1.3 Deucalion Supercomputer . . . 33
- 4.2 Problem Formulation . . . 33

4.2.1 Upper Level: Strategic Resource Coordinator . . . 34
4.2.2 Lower Level: Operational Optimal Control . . . 35
4.3 Implementation . . . 36
4.4 Results and Discussion . . . 37
4.4.1 Simulation . . . 38
4.4.2 Performance . . . 41
4.5 Summary . . . 50

5 Conclusions **53**
5.1 Future Work . . . 54

References **57**

A Algorithm for the Single-Level Problem **61**

B Algorithm for the Bilevel Problem **67**

List of Figures

3.1 Result in the referenced article for Scenario 1 26
3.2 Result obtained with the same parameters for Scenario 1 26
3.3 Result in the referenced article for Scenario 1a 27
3.4 Result obtained with the same parameters for Scenario 1a 28
3.5 Result in the referenced article for Scenario 1b 29
3.6 Result obtained with the same parameters for Scenario 1b 29

4.1 Irrigation profile for two identical fields under shared capacity constraints ($C_{\text{total}} =$ 6 mm) [28]. 39
4.2 Irrigation profile for two fields with asymmetric slope topographies ($\alpha_1 = 0°$, $\alpha_2 =$ 30°) [28]. 40
4.3 Scalability performance validation for three sectors with varying slope topographies ($\alpha = [30°, 15°, 0°]$) [28]. 42
4.4 Execution time comparison between the Advanced RISC Machine (ARM) and x86 partitions of Deucalion across problem sizes. 43
4.5 Iteration count comparison between the ARM and x86 partitions of Deucalion across problem sizes. 44
4.6 Runtime comparison between Multifrontal Massively Parallel sparse direct Solver (MUMPS) and MA97 for parameters 1. 45
4.7 Iteration count comparison between MUMPS and MA97 for parameters 1. 45
4.8 Runtime comparison between MUMPS and MA97 for parameters 2. 46
4.9 Iteration count comparison between MUMPS and MA97 for parameters 2. 47
4.10 Runtime comparison between MUMPS and MA97 for parameters 3. 47
4.11 Iteration count comparison between MUMPS and MA97 for parameters 3. 48
4.12 Runtime comparison for MA97 across parameters 1, 2, and 3. 49
4.13 Iteration count comparison for MA97 across parameters 1, 2, and 3. 49
4.14 Runtime comparison for MUMPS across parameters 1, 2, and 3. 50
4.15 Iteration count comparison for MUMPS across parameters 1, 2, and 3. 51

List of Tables

1 Contribution of this dissertation to the United Nations Sustainable Development Goals. v

2.1 Description of variables and parameters (as used in [13]) 13
2.2 Summary of Central Notations (from[12]) 17

3.1 Description and implementation values of parameters used in the evapotranspiration formula. 24
3.2 Variable assignments for daily model execution. 24
3.3 Weather input data used for validation, corresponding to Base Scenario 1 in [13]. 25
3.4 Execution times with and without parallelisation, using 4 processor cores. 30

List of Acronyms

JuMP Julia for Mathematical Programming

IPOPT Interior Point Optimizer

KKT Karush Kuhn Tucker

MPEC Mathematical Problem with Equilibrium Constraints

ASCE American Society of Civil Engineering

MUMPS Multifrontal Massively Parallel sparse direct Solver

HSL Harwell Subroutine Library

ARM Advanced RISC Machine

Chapter 1

Introduction

1.1 Context

In recent years, global warming has emerged as a critical challenge. The resulting environmental shifts have caused ongoing, highly unpredictable consequences across various ecosystems [1]. Notably, the rise in global temperatures has led to an increase in agricultural drought worldwide [2]. These prolonged arid periods negatively impact crop productivity and efficiency, eventually leading to reduced agricultural yields. As such, developing methods to mitigate the impact of these droughts becomes an imperative area of research.

In Portugal, the agricultural sector accounts for about 80% of national water consumption [3]. This demand is highly seasonal, with water consumption peaking in spring and summer to sustain crop growth. Due to this substantial demand, agriculture is among the most vulnerable sectors to the impacts of prolonged seasonal droughts. Consequently, the development of regional mitigation strategies within this industry has become an urgent national priority. An estimated 37.5% of the total water allocated to agricultural irrigation is currently lost or wasted due to systemic inefficiencies [4]. This significant optimisation gap highlights a critical avenue for research, where enhancing infrastructure efficiency can directly minimise resource waste and alleviate the impacts of drought.

In modern times, traditional flooding irrigation methods have been widely replaced by more precise techniques. While modern localised systems, such as drip irrigation, a micro-irrigation method designed to deliver water directly to the plant root zone, now constitute over 65% of the country's agricultural infrastructure, systemic inefficiencies persist due to a widespread lack of volumetric metering [5]. Because these systems are usually managed using fixed-time routines

rather than real-time soil moisture dynamics, there is severe over-irrigation [6]. This use of simple management software highlights a critical technological gap that smart, sensor-driven automation must address.

1.2 Motivation

The main limitation of the most widespread agricultural practices is not the irrigation infrastructure itself, but the lack of systems to regulate water use. Operating irrigation networks with simple, time-based frameworks leads to irrigation decisions that do not account for real-time soil conditions, such as current water saturation or depletion [7], [8]. This will negatively impact the growth of crops, as both under-watering and over-watering can compromise plant development [9]. Additionally, inefficient water allocation directly increases producers' operational overhead by increasing water-pumping energy costs [10].

Transitioning toward smart irrigation systems provides a technical remedy to these compounded inefficiencies. By utilising real-time feedback from sensors, such as soil humidity sensors or meteorological data streams, smart frameworks ensure precise resource allocation, directly reducing unnecessary water waste while maximising crop productivity. However, many smart irrigation systems focus on optimising irrigation decisions at the field level and are not very flexible in defining certain variables. These models typically rely on sensor data from a single field to analyse and determine the amount of water to be delivered by the irrigation system.

Such models typically operate by setting a static soil moisture target and applying water equal to the difference between the target and the current measured value. This static approach cannot account for real-world limitations, such as limited water availability during drought, where the required amount may exceed what is available.

Furthermore, single-level optimisation models treat the minimum soil moisture threshold, $x_{\min}$, as a fixed parameter, determined in advance and constant throughout the optimisation process. This means the threshold cannot adapt to changing resource availability throughout the season, even when conditions warrant a different target.

In a bilevel formulation, by contrast, $x_{\min}$ does not need to be fixed. It can instead be treated as a decision variable at the upper level, chosen to reflect long-term water availability, while the lower level optimises irrigation scheduling in response to soil and crop conditions, targeting short-term crop stress under the threshold set above.

When such constraints are introduced across multiple fields simultaneously, these models become difficult to optimise independently under shared resource constraints [11], leaving a clear operational gap that must be addressed.

Instead, a system is required to analyse all fields simultaneously and ration the limited water as efficiently as possible across the entire agricultural system. To bridge this operational gap, this work models the scheduling challenge as a bilevel optimisation problem [12], a hierarchical structure in which two interdependent decision levels are jointly optimised. In this problem, the upper level accounts for the total available water and optimises soil moisture targets for each field, rationing water across the agricultural system as a whole. In the lower level, each field is modelled following the single-level approach of Pereira et al. [13], where the irrigation of each individual field is decided while taking into account the target soil moisture which in our work will be decided by an additional optimisation level, called the upper level and which will act as a leader, while the lower level problem acts as a follower.

By reformulating this model as a bilevel problem, our work extends it from a single-level formulation, in which the minimum soil moisture threshold is fixed, to one in which that threshold is optimised as part of the upper-level allocation decision. The lower level then schedules irrigation in response to soil and crop conditions, subject to the threshold set above, with the two levels jointly solved so that the allocation and scheduling decisions inform one another.

1.3 Objectives

Guided by the context and operational gaps established in Sections 1.1 and 1.2, the primary goal of this dissertation is to design, implement, and evaluate a scalable, high-performance computing framework for multi-field bilevel irrigation optimisation.

To achieve this overarching goal, the research is structured around the following specific technical objectives:

- Conduct a review of state-of-the-art smart irrigation control systems, emphasising single-level optimisation approaches.
- Develop a bilevel formulation for multi-field irrigation scheduling under global water constraints.
- Implement a numerical algorithm to solve the formulated bilevel optimisation problem.

- Compare the computational efficiency of diverse mathematical programming solvers for large-scale optimisation problems.
- Assess the scalability of the proposed approach using high-performance computing infrastructure.

1.4 Dissertation Structure

In addition to the introduction, this dissertation contains four more chapters. In Chapter 2, the state of the art is described, and related work is presented. Chapter 3 presents the work done in the first part of the investigation, where the previous model described in Chapter 2 was implemented using Julia for Mathematical Programming (JuMP) and the Interior Point Optimizer (IPOPT) solver, and the comparison between the previous results and the results of this new implementation is shown. In Chapter 4, the formulation of the problem as a bilevel optimisation problem is presented, along with its implementation in the Julia programming language, and the results obtained are discussed. Finally, Chapter 5 presents the conclusions and discusses possible future work for this project.

Chapter 2

Background and Literature Review

This chapter provides an introduction to important concepts that will be used throughout the rest of this work. First, Section 2.1 discusses optimisation problems, their formulation and applications. Then, Section 2.2 discusses the previous work done on the topic of smart irrigation, which this work aims to extend. Section 2.3 discusses the concept of bilevel optimisation, and how it differs from single-level optimisation problems. Finally, Section 2.4 discusses previous applications of bilevel optimisation for water management systems.

2.1 Optimisation Problems

An optimisation problem is a class of problems where the goal is to select values for a set of decision variables that maximise or minimise an objective function, subject to constraints on those variables [14]. The objective function serves as a quantitative measure of the solution's quality, such as cost, efficiency, profit, or water usage, and the constraints define the feasible region of all acceptable solutions.

2.1.1 Definition

Formally, an optimisation problem can be described by:

$$\begin{aligned} \underset{x}{\text{minimise}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m, \quad m \geq 1 \\ & h_j(x) = 0, \quad j = 1, \ldots, p, \quad p \geq 1 \end{aligned}$$

Here, $x \in \mathbb{R}^n$ represents the vector of decision variables. The decision variables are the variables whose values are chosen to optimise the objective function. Depending on the nature of the problem, these values may be continuous or discrete. A continuous variable is a variable that can take any real value within a given range. In contrast, a discrete variable is one that can take values only from a predefined set of numbers.

The objective function is represented by $f\colon \mathbb{R}^n \to \mathbb{R}$. This is the mathematical function that must be maximised or minimised to find the optimal solution to the problem by varying the decision variables.

The constraints of this problem are represented by $g_i\colon \mathbb{R}^n \to \mathbb{R}$, with $g_i(x) \leq 0$, and $h_j\colon \mathbb{R}^n \to \mathbb{R}$, with $h_j(x) = 0$. There are restrictions on the decision variables, such as physical limits, resource capacities, or logical conditions. These restrictions represent inequality constraints and equality constraints, respectively. An equality constraint represents a strict requirement: a specific combination of decision variables must match an exact value. An inequality constraint sets a limit, either a ceiling or a floor. It specifies that a combination of the decision variables must be either not greater than or not less than a given value, but it does not require the combination to match that value exactly.

A solution that satisfies all the constraints is called "feasible". The set of all feasible solutions is called the "feasible region". The objective of an optimisation problem is to find an optimal solution. An optimal solution is a solution within the feasible region that achieves the maximum or minimum value of the objective function.

2.1.2 Classification

Optimisation problems can be classified into several categories, depending on the characteristics of the objective function and the constraints.

Firstly, an optimisation problem can either be classified as convex or nonconvex. Convex problems have the property that any local optimum is also a global optimum, which largely simplifies solution methods. In contrast, nonconvex optimisation problems may contain multiple local optima, which makes them much more challenging to solve.

An optimisation problem may be additionally categorised as linear or nonlinear. A linear optimisation problem is one in which both the objective function and the constraints are linear functions of the decision variables. Since all linear functions are convex, linear optimisation problems are a subset of optimisation problems that are simpler to solve than nonlinear optimisation

problems. Nonlinear optimisation problems may have multiple local optima. Finding the global optimal value from among these possible solutions is usually difficult. However, it is worth noting that a nonlinear problem may still be convex, meaning that nonlinearity does not necessarily imply multiple local optima.

Finally, a problem may be classified as either deterministic or stochastic, depending on whether its parameters are assumed to be known with certainty or modelled as random variables to account for uncertainty. In deterministic problems, all parameters are fixed and known, meaning the same inputs always produce the same solution. Stochastic problems, in contrast, incorporate uncertainty directly into the model, optimising expected performance across possible realisations of the uncertain parameters rather than a single fixed outcome.

2.1.3 Solving Optimisation Problems

The choice of solution method for an optimisation problem depends largely on the problem's structure. Different problem classes admit different algorithmic approaches with varying computational properties. This section introduces the main problem classes and their associated solution methods, progressively building towards the nonlinear programming techniques employed in this work.

Convex Optimisation

A convex optimisation problem is one of the form [14]:

$$\begin{aligned} &\text{minimise} && f_0(x) \\ &\text{subject to} && f_i(x) \leq b_i, \quad i = 1, \ldots, m \end{aligned} \tag{2.1}$$

where the functions $f_0, \ldots, f_m : \mathbb{R}^n \to \mathbb{R}$ are convex, meaning they satisfy:

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y) \tag{2.2}$$

for all $x, y \in \mathbb{R}^n$ and all $\alpha, \beta \in \mathbb{R}$ with $\alpha + \beta = 1$, $\alpha \geq 0$, $\beta \geq 0$. Both the least-squares and linear programming problems discussed previously are special cases of the general convex optimisation problem [14].

Interior point methods are very effective in practice, capable of solving problems with hundreds of variables and thousands of constraints reliably [14]. The key advantage of convex problems is that any local optimum is guaranteed to be a global optimum, meaning that once a solution

is found, it is the best possible. If a problem can be formulated as a convex optimisation problem, it can therefore be solved efficiently and with confidence in the result [14]. However, many real-world problems, including the one formulated in this dissertation, are nonlinear and non-convex, meaning this guarantee no longer holds and more sophisticated solution methods are required.

Nonlinear Optimisation

Nonlinear optimisation (or nonlinear programming) is the term used to describe an optimisation problem in which the objective or constraint functions are not linear. Unfortunately, there are no known methods for solving the general nonlinear programming problem. Even problems with as few as hundreds of variables are very challenging to solve. As such, methods to solve these kinds of problems can take two different approaches, each involving some form of compromise [14].

The first of these approaches is known as local optimisation. In local optimisation, the compromise is that, instead of searching for the optimal value of x across all feasible points, the solution is found locally. As such, this approach will find a local minimum among nearby feasible points, though there might be better solutions farther away.

Despite this limitation, local optimisation methods are largely used in practice. Since they only require the objective and constraint functions to be differentiable, they are broadly applicable, computationally efficient, and capable of handling large-scale problems. In engineering design, for example, local optimisation is commonly used to improve upon an initial design obtained through manual or heuristic methods, where a good solution found efficiently is more valuable than a guaranteed global optimum [14].

However, these methods have many disadvantages beyond failing to find the global optimal solution. They require an initial guess for the optimisation variable, which greatly impacts the solution obtained, as there is no way to know how distant the local optimal solution is from the global optimal solution. These methods are also very sensitive to algorithm parameter values and will need to be adjusted for each particular problem.

As such, solving a local optimisation problem is not as straightforward as solving any of the previously described types of problems. It is necessary to experiment with the choice of algorithm, which may not be adequate for the specific problem under study, and to tweak the parameters and find either a good initial guess or a method for generating one. As Boyd and Vandenberghe [14] say, “local optimisation methods are more art than technology.” This is because, unlike least-squares or linear problems, which are relatively simple to solve, finding a solution to a local

optimisation problem requires much more fine-tuning of algorithm parameters and a careful choice of a starting point.

The second approach is known as global optimisation. Unlike local optimisation, global optimisation seeks to find the true optimal solution across the entire feasible region. However, this guarantee comes at a significant computational cost [14]. The complexity of global optimisation methods grows with problem size, meaning that even problems with as few as tens of variables can take hours or even days to solve. For this reason, global optimisation is typically reserved for problems with a few variables, where computational time is not a concern and finding the globally optimal solution is critical. One such example is the worst-case analysis of safety-critical systems, where a global optimisation method is needed to certify that a system is safe under all possible conditions, something a local optimisation method cannot guarantee [14].

Among many available local optimisation solvers, this dissertation makes use of the Interior Point Optimizer (IPOPT) [15]. IPOPT is a solver for large-scale nonlinear programming problems, which works by reformulating the original problem into a sequence of smaller subproblems, each of which is solved with increasing accuracy until the solution of the original problem is found.

At each subproblem, IPOPT computes the Karush Kuhn Tucker (KKT) conditions, which are the first-order necessary conditions for optimality in a constrained nonlinear program [16].

Consider a general nonlinear program of the form

$$\min_x f(x) \quad \text{subject to} \quad g_i(x) \leq 0,\ i \in \mathscr{I}, \qquad h_j(x) = 0,\ j \in \mathscr{E}, \tag{2.3}$$

where g_i and h_j are the inequality and equality constraints introduced in Section 2.1.1, with index sets $\mathscr{I}$ and $\mathscr{E}$ respectively. A point x^* satisfies the KKT conditions if there exist multipliers $\lambda_i^* \geq 0$ (for $i \in \mathscr{I}$) and μ_j^* (for $j \in \mathscr{E}$) such that

$$\nabla f(x^*) + \sum_{i \in \mathscr{I}} \lambda_i^* \nabla g_i(x^*) + \sum_{j \in \mathscr{E}} \mu_j^* \nabla h_j(x^*) = 0, \tag{2.4a}$$

$$g_i(x^*) \leq 0, \quad \forall i \in \mathscr{I}, \tag{2.4b}$$

$$h_j(x^*) = 0, \quad \forall j \in \mathscr{E}, \tag{2.4c}$$

$$\lambda_i^* \geq 0, \quad \forall i \in \mathscr{I}, \tag{2.4d}$$

$$\lambda_i^* g_i(x^*) = 0, \quad \forall i \in \mathscr{I}. \tag{2.4e}$$

Equation (2.4a) requires the gradient of the objective to be balanced by a weighted sum of the

gradients of the constraints, so that no feasible direction can further decrease f without violating a constraint. Equations (2.4b) and (2.4c) require x^* to satisfy all constraints. Equation (2.4d) requires the inequality multipliers to be non-negative, so that relaxing a constraint cannot increase the optimal objective value. Equation (2.4e) forces $\lambda_i^* = 0$ for any inequality constraint that is not active at x^*, that is, where $g_i(x^*) < 0$, so that only the active constraints

$$\mathscr{A}(x^*) = \{i \in \mathscr{I} : g_i(x^*) = 0\} \cup \mathscr{E} \tag{2.5}$$

contribute to the stationarity condition, which can then be written more compactly as

$$\nabla f(x^*) = -\sum_{i \in \mathscr{A}(x^*)} \lambda_i^* \nabla c_i(x^*), \tag{2.6}$$

where c_i denotes the i-th active constraint (either g_i or h_j), and λ_i^* its associated multiplier.

IPOPT iteratively updates both the optimisation variables and the KKT multipliers until these conditions are satisfied to a predefined numerical tolerance [15]. As with any local optimisation method, the solution found by IPOPT is sensitive to the choice of starting point, since different initial guesses may lead to different local optima. Furthermore, since the KKT conditions are necessary but not sufficient for global optimality, the solution found is not guaranteed to be the global optimum, which is consistent with the limitations of local optimisation methods discussed previously.

2.1.4 Optimal Control Problems

An extension of the general optimisation problem introduced in Section 2.1.1 arises when the decision variables are indexed over discrete time steps and split into two distinct roles: state variables and control variables [17].

A discrete-time optimal control problem is defined by three components [17]. The first is a state equation, describing how the state variable x_i evolves over discrete time steps $i = 1, \ldots, M$ under the influence of a control variable u_i, typically of the form:

$$x_{i+1} = f(x_i, u_i, i).$$

The second is a performance index, a scalar quantity J to be minimised or maximised over the time horizon, generally expressed as a sum over the time steps of the problem. The third is a set

of constraints that may restrict the admissible values of the state, the control, or both, and which typically include boundary conditions that fix the initial (and possibly final) state value.

Formally, the problem is stated as finding an admissible control sequence $\{u_i\}_{i=1}^{M}$ that drives the state $\{x_i\}_{i=1}^{M}$ along a trajectory satisfying the state equation and boundary conditions, while minimising (or maximising) the performance index [17].

Despite its different terminology, a discrete-time optimal control problem is a special case of the general optimisation problem introduced in Section 2.1.1. The state trajectory and control sequence together form the vector of decision variables x, the performance index corresponds to the objective function $f(x)$, and the state equation is enforced as a set of equality constraints $h_j(x) = 0$. As such, a discrete-time optimal control problem can be solved using the same nonlinear programming techniques discussed in Section 2.1.3 [17].

2.2 Smart Irrigation as an Optimal Control Problem

The work [13] represents a step toward adaptive smart irrigation systems, where the irrigation scheduling problem is formulated as an optimal control problem that accounts for both environmental factors and real-time sensor feedback. The central goal of their work is to minimise total water consumption while ensuring safe moisture levels for the crop throughout the irrigation period. Their formulation extends earlier optimal control models by incorporating additional realistic factors such as soil slope, runoff losses, and field capacity saturation, making the model more representative of actual field conditions.

The irrigation system is modelled in discrete time, with soil moisture changing according to a water balance law that accounts for irrigation input, evapotranspiration, rainfall, and infiltration and runoff losses. The control variable in the model represents irrigation water applied at each time step, and constraints ensure non-negative irrigation, soil moisture above a crop-specific minimum, and realistic moisture propagation through the soil. This discrete optimal control problem is solved

numerically using nonlinear programming techniques. The problem is modelled by:

$$\min_{u_i, x_i} \quad h \sum_{i=1}^{M} u_i \tag{2.7}$$

$$\text{subject to:} \quad \begin{cases} x_{i+1} = x_i + h\, f(t_i, x_i, u_i, x_{i+1}), & i = 1, \ldots, M-1 \\ x_i \geq x_{\min}, & i = 1, \ldots, M \\ u_i \geq 0, & i = 1, \ldots, M \\ x_1 = x_s \end{cases} \tag{2.8}$$

where the water balance function f governs how soil moisture evolves between time steps:

$$\begin{aligned} f(t_i, x_i, u_i, x_{i+1}) = K_I u_i [1 - R_F \sin(\alpha)] - K_C \,\text{evtp}_0(t_i) \\ + [1 - R_F \sin(\alpha)]\,\text{Rainfall}(t_i) - \text{loss}(x_i, x_{i+1}) \end{aligned} \tag{2.9}$$

and the loss term accounts for deep infiltration below field capacity and saturation-driven runoff above it:

$$\text{loss}(x_i, x_{i+1}) = \begin{cases} k x_i, & \text{if } x_i \leq x_{FC} \\ (x_{i+1} - x_{FC}) C_S, & \text{if } x_i > x_{FC} \end{cases} \tag{2.10}$$

The meaning of each variable is described in Table 2.1.

The factor $[1 - R_f \sin(\alpha)]$, which is applied to both the irrigation input u_i and rainfall values, takes into account the amount of water that is lost due to runoff losses caused by steep terrain. Steeper terrain (larger α) will increase runoff losses from both sources, thereby reducing water intake. This implies that fields on steeper grounds require more irrigation to achieve the same level of soil moisture gain.

K_C scales evapotranspiration depending on the type of crop being cultivated, allowing for the adjustment of the model to be applied for different crops without reformulation.

The loss term describes a conditional case that accounts for two distinct situations. When soil moisture is below the field capacity x_{FC}, water loss is primarily caused by deep percolation, which is proportional to the current moisture level. When water exceeds x_{FC}, additional water can not be retained by the soil and is instead lost at a rate defined by the saturation coefficient C_S. This definition, being conditional, may raise issues when implemented in a solver, as it requires the solver to handle nonconvex problems.

Table 2.1: Description of variables and parameters (as used in [13])

Symbol	Description
t	Time (days or hours)
M	Number of time intervals (days or hours) of irrigation planning
h	Time step
x_s	Initial soil moisture measured by the moisture sensor
$x_{\min}$	Minimum soil moisture to satisfy crop water needs
u_i	Irrigation amount at time step i
K_I	Irrigation coefficient related to type of irrigation
α	Angle of soil inclination
R_F	Runoff loss coefficient
K_C	Crop coefficient related to crop type
$evtp_0$	Reference evapotranspiration value
$Rainfall(t_i)$	Rainfall amount at time t_i
$loss(x_i, x_{i+1})$	Water loss due to soil characteristics and moisture level
x_{FC}	Soil moisture at available water capacity
C_S	Saturation coefficient (accounts for soil excess water infiltration)
k	Deep infiltration coefficient, related to soil permeability

The problem is solved numerically using the IPOPT algorithm, a large-scale nonlinear programming solver well-suited to problems with smooth but potentially nonconvex objective and constraint functions. The choice of a nonlinear solver over, for instance, a linear or quadratic programming approach is motivated precisely by the nonlinear water balance dynamics and the conditional loss function, both of which would be poorly approximated by linear relaxations. The discrete-time formulation also simplifies the integration of real-time sensor data, since each time step naturally corresponds to a measurement interval.

2.2.1 Replan and Redistribution Algorithm

A key contribution of [13] is the replan algorithm, which operates by periodically comparing the measured soil moisture from a soil moisture sensor with the model-predicted value at that time step. If the measured moisture falls below the minimum safe level $x_{\min}$, the algorithm triggers a response. First, an immediate irrigation correction is computed to restore soil moisture above the safety threshold as quickly as possible. Then, the full optimal control problem is re-solved from the current time step forward, using updated weather forecast data as input. This produces a revised irrigation schedule that reflects both the corrected initial condition and the latest environmental information. The algorithm thus closes the feedback loop between the physical field and the

optimisation model, transforming what would otherwise be a static open-loop plan into an adaptive controller.

The paper also proposes a water redistribution algorithm for fields with multiple irrigation points. Each irrigation point generates its own plan independently, but if a point cannot meet its planned water requirement on a given day due to capacity constraints, the shortfall is redistributed among the remaining points. If total demand exceeds the combined capacity of all points, the available water is distributed as evenly as possible, and the farmer is notified of the deficit.

2.2.2 Simulation Results and Validation

The model was validated through a set of simulation scenarios using real weather data from the Porto region, Portugal. The results showed that the model responds correctly to the problem's physical parameters: increasing terrain slope increases water consumption due to runoff, higher temperatures increase evapotranspiration and therefore irrigation demand, and the replan algorithm successfully corrects deviations caused by unexpected changes in soil moisture. Notably, the optimisation was performed at a single decision level, with each irrigation point planned independently. However, the allocation of water across irrigation points is handled reactively, redistributing water only when a point cannot meet its planned demand, rather than being optimised with the irrigation plants themselves.

The replan algorithm was designed to modify irrigation schedules in response to real-time changes in environmental conditions, such as unexpected rainfall or variations in evapotranspiration. Simulation results showed that the replan algorithm effectively corrected deviations from the planned soil moisture trajectory. When rainfall occurred, the algorithm reduced irrigation in subsequent intervals, preventing over-irrigation. Conversely, in dry periods, the replan algorithm increased irrigation to compensate for deficits, ensuring that crop water requirements were met. These results highlight the algorithm's ability to maintain soil moisture close to target values under varying weather conditions.

From a water-efficiency perspective, the results showed a meaningful reduction in total water use compared to conventional fixed-schedule irrigation, without compromising soil moisture maintenance. The mechanism ensures that water is applied precisely when and to the extent required by the safety constraint. Soil moisture was maintained within the prescribed bounds throughout the irrigation period in all scenarios tested.

2.2.3 Overall Impact and Limitations

Results of [13] demonstrate that an optimal control problem, solved by the nonlinear programming tools, provides irrigation schedules that reduce water consumption while keeping soil moisture within agronomically safe bounds, and that this remains true under varying terrain slope, runoff, and weather conditions.

Despite these results, the formulation of [13] has a structural limitation that motivates this dissertation. The target soil moisture value $x_{\min}$ is treated as a fixed static parameter, specified before optimisation. In practice, optimal moisture targets may be constrained by a limited water budget, suggesting that they should be treated as decision variables rather than constants. Because $x_{\min}$ is fixed in advance, each irrigation point can plan its own schedule independently, and the allocation of water across points is therefore handled at a single decision level, with interactions between points addressed only reactively through the redistribution algorithm when a point fails to meet its demand.

Promoting $x_{\min}$ to a decision variable, however, introduces a new difficulty. The value chosen for each irrigation point's target moisture directly determines the optimal schedule for that point, meaning the two decisions cannot be made independently. Representing this dependency requires a hierarchical structure in which an upper-level decision determines moisture targets across irrigation points, anticipating the schedules that result from them, while each lower-level irrigation point solves its own optimal control problem given its allocated target. This naturally leads to a bilevel optimisation problem, which is introduced in the following section.

2.3 Bilevel Optimisation

Bilevel optimisation problems were first studied in the context of hierarchical decision-making and are closely related to Stackelberg games, where a leader makes decisions anticipating the reaction of a follower. Bilevel optimisation has been applied in areas such as energy market design, transportation network planning, and economic modelling. Despite their wide applicability, bilevel problems are challenging due to their nonconvexity and potential discontinuities, which make them computationally intensive to solve [12], [18].

Bilevel optimisation is a specialised type of optimisation problem where two levels of decision-making processes exist. The problem can be thought of as a "nested" optimisation problem, where

the solution to the upper-level (leader) optimisation problem depends on the solution to a lower-level (follower) optimisation problem. The two levels are interdependent: the upper-level solution depends on the lower-level solution.

2.3.1 Definition

Formally, a bilevel optimisation problem consists of two optimisation problems:

- **Upper-level problem (Leader's problem)**: The decision-maker at the upper level (often referred to as the "leader") aims to optimise an objective function, subject to constraints that may include both the decision variables of the upper level and the solution to the lower-level problem.
- **Lower-level problem (Follower's problem)**: The decision-maker at the lower level (referred to as the "follower") aims to optimise its own objective function, subject to its own constraints, given the decisions made by the leader.

Mathematically, a bilevel optimisation problem can be defined as the following [12].

Definition 1. For the upper-level objective function

$$F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$$

and lower-level objective function

$$f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R},$$

the bilevel problem is given by [12]

$$\text{“min”}_{x_u \in X_U, x_l \in X_L} F(x_u, x_l)$$

subject to

$$x_l \in arg\,min_{x_l \in X_L} \left\{ f(x_u, x_l) : g_j(x_u, x_l) \leq 0, j = 1, \ldots, J \right\}$$

$$G_k(x_u, x_l) \leq 0, \quad k = 1, \ldots, K$$

where $G_k : X_U \times X_L \to \mathbb{R}, \quad k = 1, \ldots, K$ denote the upper level constraints, and $g_j : X_U \times X_L \to \mathbb{R}$ are the lower level constraints [12].

The following table summarises the central notations used in bilevel optimisation problems, as adopted from [12]:

Table 2.2: Summary of Central Notations (from[12])

Category	Notation	Description
Decision vectors	$x_u \in X_U$ $x_l \in X_L$	Leader's (upper level) decision variable and decision space. Follower's (lower level) decision variable and decision space.
Objectives	F f	Leader's (upper level) objective functions. Follower's (lower level) objective functions.
Constraints	$G_k, k = 1,\ldots,K$ $g_j, j = 1,\ldots,J$	Leader's (upper level) constraint functions. Follower's (lower level) constraint functions.
Lower level feasible region	$\Omega : X_U \to X_L$	$\Omega(x_u) = \{x_l : g_j(x_u,x_l) \leq 0 \forall j\}$.
Constraint region	$\Phi = gph\Omega$	$\Phi = \{(x_u,x_l) : G_k \leq 0, g_j \leq 0\}$.
Lower level reaction set	$\Psi : X_U \to X_L$	$\Psi(x_u) = arg\,min_{x_l \in \Omega(x_u)} f(x_l,x_u)$.
Inducible region	$I = gph\Psi$	$I = \{(x_u,x_l) \in \Phi : x_l \in \Psi(x_u)\}$.
Choice function	$\psi : X_U \to X_L$	Follower-selected solution under multiple optima.
Optimal value function	$\varphi : X_U \to \mathbb{R}$	$\varphi(x_u) = \min_{x_l \in \Omega(x_u)} f(x_u,x_l)$.

The quotation marks around "min" in Definition 1 are used deliberately, following [12], because the upper-level problem is not necessarily well-defined as written. This happens when, for a given upper-level decision vector x_u, the lower level has multiple optimal solutions rather than one. In that case, the lower level is represented by a set-valued mapping rather than an ordinary function, and the value of $F(x_u,x_l)$ can depend on the lower-level optimal solution used. The upper-level problem becomes a proper minimisation only once it is decided which of these lower-level optimal solutions the follower is assumed to choose.

There are two positions used to resolve this ambiguity: the leader takes either the optimistic or the pessimistic position [12], which are discussed further in the next sections. When the lower level has one and only one optimal solution for every x_u, the ambiguity disappears, the two positions agree with each other, and the upper-level problem reduces to a plain minimisation, which is the case originally intended by the "min" notation above.

2.3.2 Optimistic Position

In the optimistic case, when multiple solutions exist at the lower level, the leader assumes the follower will select the solution most favourable to the upper-level objective. This choice function may be defined as (see [12]):

$$\Psi^O(x_u) = arg\,min_{x_l \in \Psi(x_u)} F(x_u,x_l),$$

where Ψ^O represents the follower's optimal response that minimises the leader's objective. This formulation assumes cooperation between the leader and the follower. The corresponding bilevel

optimisation problem can be expressed as:

$$\min_{x_u \in X_U, x_l \in X_L} F(x_u, x_l), \quad \text{subject to } x_l \in \Psi^O(x_u),\ G_k(x_u, x_l) \leq 0, \quad k = 1, \ldots, K$$

When the solution is unique, we can substitute $x_l \in \Psi^O(x_u)$ with $x_l = \Psi^O(x_u)$.

The optimistic formulation is generally easier to handle than the pessimistic position. Under reasonable assumptions of regularity and compactness, the problem is guaranteed to have an optimal solution. Formally, this is stated as follows [12]:

Theorem 1: If the functions F, f, G_k, and g_i are sufficiently smooth, the constraint region Φ of the bilevel optimisation problem is nonempty and compact, and the Mangasarian-Fromowitz constraint qualification holds at all points, then the problem is guaranteed to have an optimistic bilevel optimum provided there exists a feasible solution.

The Mangasarian-Fromowitz constraint qualification requires, at a feasible point, that the gradients of the active equality constraints are linearly independent, and that there exists a feasible direction along which all active inequality constraints decrease strictly. This ensures the constraint region is well-behaved locally, ruling out cases such as constraints with zero gradient at the boundary.

2.3.3 Pessimistic Position

In contrast to the optimistic position, the pessimistic position assumes that the follower will select a solution from the lower-level optimal set that leads to the worst outcome for the upper-level objective. The pessimistic choice function can be defined as (see [12]):

$$\Psi^P(x_u) = \arg \max_{x_l \in \Psi(x_u)} F(x_u, x_l),$$

where Ψ^N denotes the follower's optimal response, which minimises the leader's objective. This formulation assumes no cooperation between the leader and the follower. Accordingly, the bilevel optimisation problem under the pessimistic position is formulated as:

$$\min_{x_u \in X_U, x_l \in X_L} F(x_u, x_l), \quad \text{subject to } x_l \in \Psi^P(x_u),\ G_k(x_u, x_l) \leq 0, \quad k = 1, \ldots, K.$$

When the solution is unique, we can substitute $x_l \in \Psi^P(x_u)$ with $x_l = \Psi^P(x_u)$.

The pessimistic position is significantly more challenging to control than the optimistic position. While certain optimistic bilevel problems can be reformulated as single-level problems, this is generally not possible under the pessimistic formulation. In particular, each lower-level solution must be evaluated to determine which outcome is worst for the upper level, effectively turning the problem into a three-level task.

Under strong assumptions, pessimistic bilevel problems are guaranteed to have optimal solutions, as formalised in the following theorem (see [12]):

Theorem 2: If the functions F, f, G_k, and g_i are sufficiently smooth, the constraint region of the bilevel problem is nonempty and compact, and the set-valued map p is lower semicontinuous for all upper-level decision vectors, then the problem is guaranteed to have a pessimistic bilevel optimum.

2.3.4 Solution Methods

Solving bilevel optimisation problems is significantly more difficult than solving single-level programs. Even in simple cases, bilevel problems are generally non-convex, and the feasible region defined by the lower-level reaction set $\Psi(x_u)$ is typically non-convex and potentially discontinuous, which rules out the direct application of standard nonlinear programming solvers.

The most common approach for problems where the lower level is convex and satisfies a suitable constraint qualification is to replace the lower-level problem with its KKT conditions. Since the KKT conditions are necessary and sufficient for optimality when the lower level is convex, this substitution is exact, and any solution to the reformulated problem is also a solution to the original bilevel problem. The result of this substitution is a single-level program, known as a Mathematical Program with Equilibrium Constraints Mathematical Problem with Equilibrium Constraints (MPEC), which can, in principle, be passed to a standard nonlinear programming solver such as IPOPT.

2.4 Bilevel Optimisation in Irrigation Water Management

While bilevel optimisation has not previously been applied to field-level irrigation planning in the manner formulated by Pereira et al. [13], it has been explored in the broader context of irrigation water management. Zhou et al. [19] observe that in large-scale irrigation systems, water allocation decisions are made by multiple independent parties: a management agency decides how water is distributed across canals, while farmers independently decide how to use their allocated share.

Traditional single-level models ignore this distinction by treating both parties as a single decision-maker, leading to solutions that are achievable only by either forcing farmers to use more water than they need or removing their right to make their own decisions.

To address this, [19] proposes a bilevel model in which the agency serves as the leader and the farmers as followers. The agency optimises the water allocation plan, anticipating how farmers will respond, rather than deciding on behalf of both. The results show that this produces water-use plans that are more consistent with what farmers actually want, reducing planned water use while remaining achievable in practice. However, the problem addressed in [19] concerns water distribution between an agency and farmers across a canal network, not field-level irrigation planning. This dissertation applies a bilevel formulation to a different problem: an upper-level decision determines the target soil moisture for each crop field, and each field then solves its own irrigation lower-level optimisation problem subject to that budget. The selection at the lower level is optimistic (as previously explained).

2.5 Summary

This chapter introduced the theoretical background necessary for the work developed in this dissertation. The fundamentals of optimisation were presented, covering the classification of optimisation problems and the solution methods applicable to each class, with particular emphasis on nonlinear programming and the IPOPT solver used throughout this work.

The smart irrigation problem formulated by Pereira et al. [13] was then reviewed, including the optimal control model and the water redistribution approach for multiple irrigation points. While the model effectively minimises water consumption while maintaining crop safety, the allocation of water across fields is handled reactively rather than optimised as a decision in its own right.

Bilevel optimisation was introduced as a framework for problems involving hierarchical decision-making, where the solution of one optimisation problem depends on the solution of another. The two main positions for handling multiple lower-level optima were discussed: the optimistic position, which assumes the follower cooperates with the leader, and the pessimistic position, which assumes the follower responds in the worst case. The KKT-based MPEC reformulation was introduced as the standard approach for solving bilevel problems with a convex lower level.

Finally, the work [19] was discussed as an existing application of bilevel optimisation to irrigation water management. Their model addresses water distribution between a managing agency and farmers across a canal network, a different problem from field-level irrigation planning.

Chapter 3

Single Level Implementation

This chapter describes the implementation of a single-level irrigation optimisation model in Julia, based on the formulation of Pereira et al. [13]. The goal of this implementation is to establish a validated computational baseline for the bilevel extension developed in Chapter 4. Section 3.1 describes the tools used. Section 3.2 describes the model implementation. Section 3.3 presents the validation results and the performance results obtained with parallelisation.

3.1 Tools

3.1.1 Julia and JuMP

This dissertation uses the Julia programming language as the primary development environment. Julia combines the syntax simplicity of high-level scripting languages with execution speeds comparable to compiled languages such as C and Fortran [20]. Unlike interpreted languages such as Python, Julia compiles code just-in-time, meaning that the same high-level mathematical expressions are executed at very high speeds. This makes it particularly attractive for optimisation problems where the objective function and constraints must be evaluated repeatedly across many iterations.

Central to the implementation is JuMP [21], an algebraic modelling language embedded in Julia. JuMP allows optimisation problems to be defined in a syntax that closely resembles standard mathematical notation, with variables, constraints, and objective functions written in forms that mirror their mathematical definitions, rather than being manually translated into solver-specific

data structures. This is particularly valuable when working with the nonlinear programming problem formulated in Section 2.2, where the constraints involve nonlinear expressions in both the state and control variables.

A key feature of JuMP relevant to this dissertation is its support for automatic differentiation. Since IPOPT requires first and second derivatives of the objective and constraint functions at each iteration, computing these manually would be both error-prone and time-consuming. JuMP handles this automatically by propagating derivatives through the model expressions at compile time, producing gradients and Hessians that are passed directly to IPOPT in the sparse matrix format it requires. This ensures accuracy and removes a significant source of potential implementation errors.

JuMP is also solver-independent, meaning the same model formulation can be passed to different solvers without modifying the problem definition. For this work, JuMP interfaces directly with IPOPT, described in Section 2.1.3, which was selected as the solver due to its suitability for large-scale nonlinear programming problems of the type addressed here, as well as for consistency with the work done previously in Pereira et al. [13], where a different implementation of the same IPOPT algorithm is used.

The optimisation problem was translated into JuMP by mapping each mathematical object in the formulation directly onto a JuMP variable, expression, or constraint. The control and state variables u_i and x_i were declared as vectors of decision variables indexed over the planning horizon $i = 1, \ldots, M$, with the non-negativity bound on u_i and the minimum soil moisture bound on x_i applied directly as variable bounds rather than as separate inequality constraints. The objective function in equation (2.7) was defined as a single linear expression summing the control variable over the horizon. The recursive state equation in equation (2.8) was implemented as $M - 1$ individual equality constraints, added within a loop over the planning horizon rather than as a single vectorised constraint, since each constraint links x_{i+1} to $x_i u_i$ and the corresponding daily weather inputs at index i. The initial condition $x_1 = x_s$ was added as a separate equality constraint rather than expressed as a variable bound.

The loss term defined in equation (2.10) was encoded in JuMP as a single algebraic expression, in which each branch of the piecewise definition is multiplied by a boolean comparison (`x[i] <= xfc` and its complement) rather than being written with explicit conditional logic, allowing the whole expression to be passed to JuMP as one differentiable formula. Although this expression is continuous at $x_i = x_{FC}$, it is not continuously differentiable there, since the boolean-

weighted formulation produces a discontinuity in the first derivative at the switching point. This is not strictly compatible with the smoothness assumptions underlying IPOPT's algorithm, which relies on first and second derivatives computed via automatic differentiation. Despite this, no convergence difficulties were observed in practice across any of the validation scenarios.

3.1.2 Parallelisation

For large problems, solving the nonlinear program with IPOPT is computationally expensive. At each iteration of the interior-point algorithm, IPOPT must solve a large sparse linear system, known as the KKT system, introduced in Section 2.1.3, to compute the search direction. The factorisation of this system is the dominant computational cost per iteration and increases with the problem size.

To address this, IPOPT was configured to use a parallelised linear solver for the factorisation step. Rather than factorising the KKT matrix sequentially on a single processor core, the work is distributed across multiple cores, reducing the time spent per iteration. This parallelisation operates at the level of the solver's internal linear algebra package, and requires no changes to the model formulation or the JuMP interface, only a change in the solver configuration passed to IPOPT at initialisation.

To avoid contention between different layers of parallelism, the BLAS thread pool used internally by Julia for general linear algebra was restricted to a single thread (BLAS.set_num_threads(1)), while the linear solver used by IPOPT was allowed four threads via the OMP_NUM_THREADS environment variable. This separation was necessary because BLAS and the solver's internal threading would otherwise compete for the same processor cores, a condition known as over-subscription that typically degrades performance rather than improving it. By confining parallel execution to the KKT factorisation step inside IPOPT, the available cores are dedicated to the single computation that dominates the per-iteration cost, rather than being shared across multiple, smaller-scale parallel regions.

3.2 Implementation

The irrigation optimisation problem described in Section 2.2 was implemented in Julia using JuMP and IPOPT, following the formulation of [13]. The objective is to minimise total water consumption subject to the soil moisture dynamics and constraints defined in equations (2.7)-(2.10). The

multiple irrigation points and water redistribution components of the original model were not implemented, as this chapter focuses on the single-field formulation, which serves as the lower-level problem in the bilevel extension developed in Chapter 4.

A key input to the model is the reference evapotranspiration E_T, which quantifies the water lost from the soil through evaporation and plant transpiration under reference conditions. This value is computed daily using the Penman-Monteith equation [22]:

$$E_T = \frac{0.408\Delta(R_n - G) + \gamma \frac{C_p}{T+273} u_2 (e_s - e_a)}{\Delta + \gamma(1 + C_d u_2)}$$

The parameters used in this formula are described in Table 3.1.

Symbol	**Description**
E_T	Evapotranspiration (mm/day)
Δ	Slope of the saturation vapour pressure curve (kPa/°C)
R_n	Net radiation at the crop surface (MJ/m^2/day)
G	Soil heat flux density (MJ/m^2/day)
γ	Psychrometric constant (kPa/°C)
C_p	Specific heat of air at constant pressure (MJ/kg°C)
T	Air temperature (°C)
u_2	Wind speed at 2 metres above the ground (m/s)
e_s	Saturation vapour pressure (kPa)
e_a	Actual vapour pressure (kPa)
C_d	Wind drag coefficient (dimensionless)

Table 3.1: Description and implementation values of parameters used in the evapotranspiration formula.

The model takes as daily inputs the rainfall and wind speed measurements, denoted Rainfall$[i]$ and WindSpeed$[i]$ respectively, where i is the day index, as well as the daily minimum and maximum temperatures, $T_{\min}[i]$ and $T_{\max}[i]$.

The specific constant values and variable assignments used in this implementation are given in Table 3.2.

Symbol	**Value**
T	$(T_{\max}[i] + T_{\min}[i])/2$
u_2	WindSpeed$[i]$
C_d	0.34
γ	0.06665
G	0.0 (negligible over 24 hours)

Table 3.2: Variable assignments for daily model execution.

The remaining quantities are computed dynamically from daily weather data using the Penman-Monteith methodology [23]. The slope of the saturation vapour pressure curve Δ is computed as:

$$\Delta = \frac{4098\left[0.6108\exp\left(\frac{17.27T}{T+237.3}\right)\right]}{(T+237.3)^2}$$

The saturation and actual vapour pressures are estimated from the daily maximum and minimum air temperatures ($T_{\max}$ and $T_{\min}$) using:

$$e_0(T) = 0.6108\exp\left(\frac{17.27\,T}{T+237.3}\right)$$

with $e_s = e_0(T_{mean})$, $e_a = e_0(T_{\min})$ [23] and $T_{mean} = (T_{\max} + T_{\min})/2$.

Finally, the net radiation R_n is estimated from the daily temperature range [23]:

$$R_s = k_R\sqrt{T_{\max} - T_{\min}}\cdot R_a \qquad R_n = 0.77\cdot R_s,$$

where the factor 0.77 accounts for the reference albedo value of grass ($\alpha = 0.23$).

3.3 Validation

To validate the implementation, the model was run with the same input data as in Base Scenario 1 of [13]. We used the data described in Table 3.3 and compared the results with those reported in the original study.

Variable	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**	**D8**	**D9**	**D10**
Rainfall (mm)	10	0	0	0	0	0	0	10	0	0
Wind speed (m/s)	2.8	3.0	2.4	2.4	3.0	2.2	2.7	2.0	2.2	2.8
Min. temp. (°C)	17.4	17.1	15.2	15.3	15.1	15.6	16.3	16.3	15.4	14.7
Max. temp. (°C)	24.4	20.8	23.1	22.2	24.9	26.9	29.3	23.6	19.4	20.4

Table 3.3: Weather input data used for validation, corresponding to Base Scenario 1 in [13].

The model was first run with a flat terrain ($\alpha = 0°$) and no runoff losses, matching the base scenario of the original study. The results are shown in Figure 3.1, with the original results above and those obtained by this implementation below.

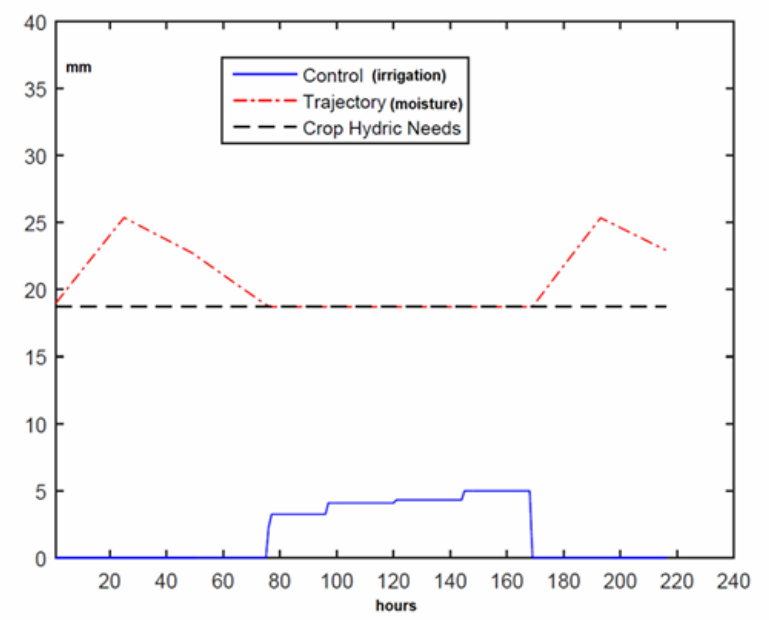


Figure 3.1: Result in the referenced article for Scenario 1

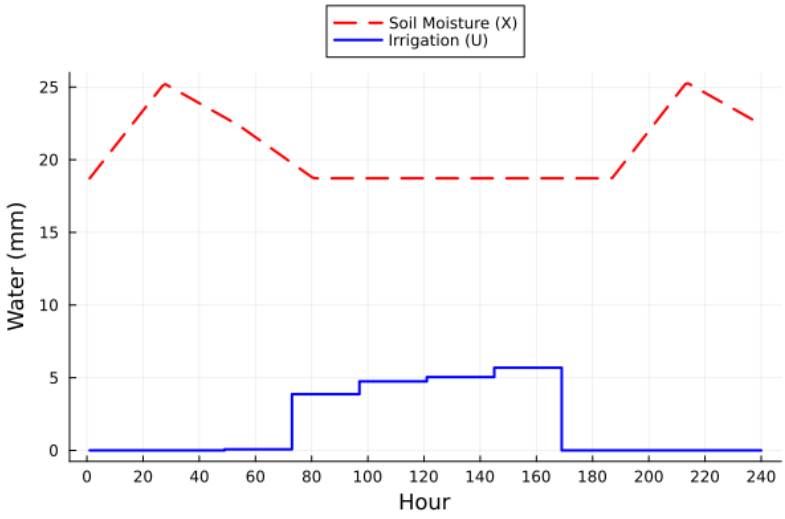


Figure 3.2: Result obtained with the same parameters for Scenario 1

The qualitative behaviour is consistent with the original: no irrigation is applied in the first days following the rainfall on day 1, irrigation begins once the soil moisture starts approaching the minimum soil moisture threshold, and stops again after the second rainfall on day 8. The shapes of both the irrigation schedule and the soil moisture trajectory closely follow those of the original study. The main visible difference lies in the magnitude of the irrigation applied once the threshold is first approached: in the original study, irrigation begins with a lower value of

approximately 3 mm on day 3, gradually increasing over subsequent days, whereas in our implementation, irrigation begins directly at a higher value of approximately 4mm. This suggests that the evapotranspiration values computed by the Penman-Monteith formula in this implementation are marginally higher than those used in the original study around the onset of the irrigation period, requiring a larger initial correction to keep soil moisture above the minimum threshold. The most likely explanation for this discrepancy is the difference between the Penman-Monteith formulation [23] used in this implementation and the American Society of Civil Engineering (ASCE) standardised reference evapotranspiration equation [24] cited by Pereira et al., which uses similar but not identical approaches to computing reference evapotranspiration.

The model was then tested under modified conditions corresponding to Scenario 1a of the original study, with the terrain inclination set to $\alpha = 20°$ and the runoff factor set to $R_f = 0.5$, meaning that half of the water from both irrigation and rainfall is lost due to runoff before reaching the soil.

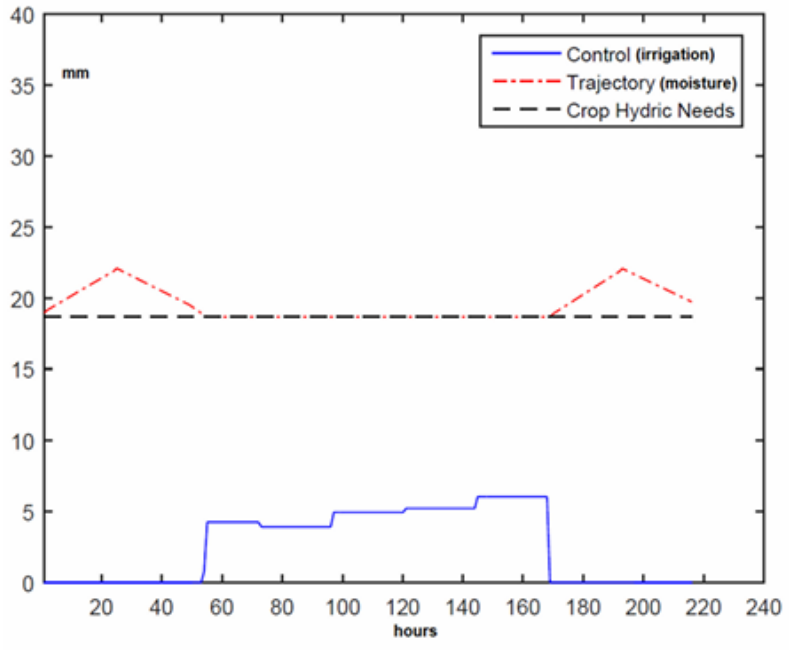


Figure 3.3: Result in the referenced article for Scenario 1a

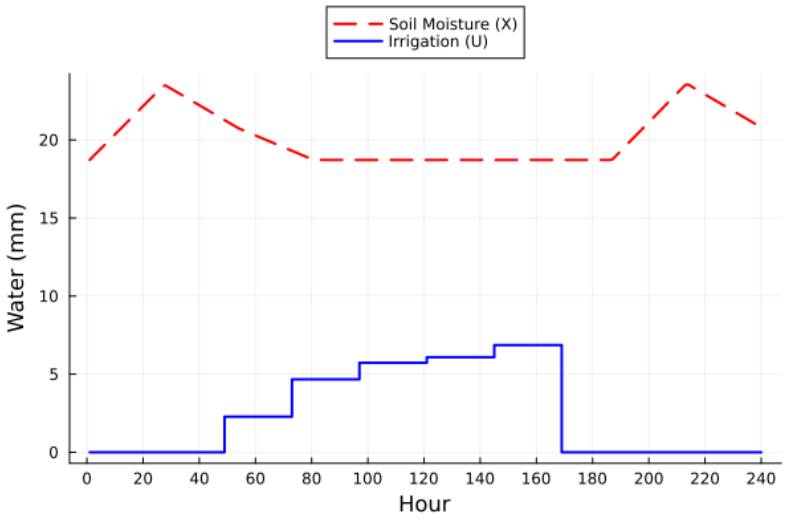


Figure 3.4: Result obtained with the same parameters for Scenario 1a

As expected, the introduction of slope and runoff losses reduces the amount of water reaching the soil, causing the controller to apply significantly more irrigation than in the base scenario. This behaviour is correctly reproduced by the implementation. Compared to the previous iteration, this version shows a clear improvement in agreement with the reference. However, the deviations in this scenario remain large and clearly visible. In particular, the soil moisture trajectory is situated noticeably higher in the implementation results, and the irrigation pulses are concentrated later in the planning period, rather than being spread more evenly across a longer stretch as in the original. These differences are consistent with the evapotranspiration discrepancy observed in the base scenario, when runoff losses are also present, the two sources of water loss compound, amplifying the effect of any difference in the evapotranspiration calculation on the resulting irrigation schedule. The reduced magnitude of these deviations relative to the previous iteration confirms that the new implementation improves the simulation's accuracy, even though a substantial gap with respect to the reference persists in this more demanding scenario.

The model was additionally tested with the same parameters as the base scenario, but with the maximum temperature on day 5 increased to 38.9°C instead of 24.9°C, corresponding to Scenario 1b of the original study. This modification was introduced to assess whether the implementation correctly captures the effect of a temperature spike on evapotranspiration and the resulting irrigation response.

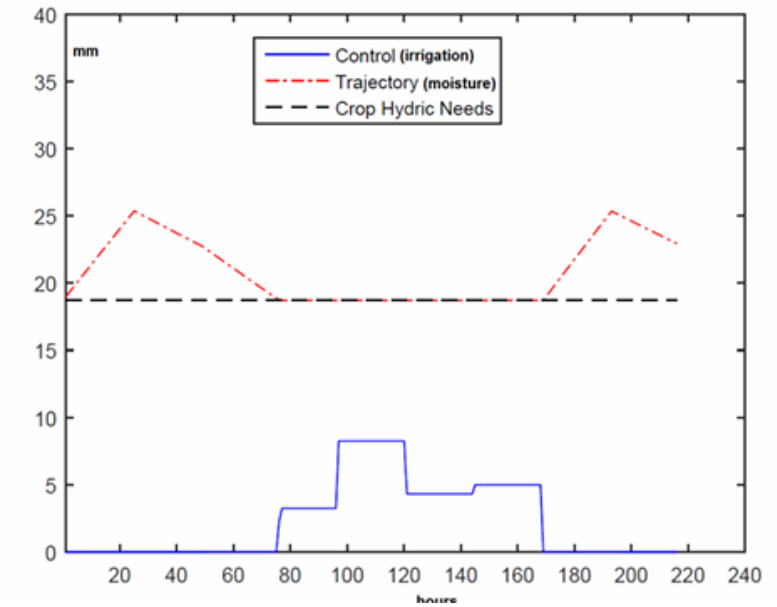


Figure 3.5: Result in the referenced article for Scenario 1b

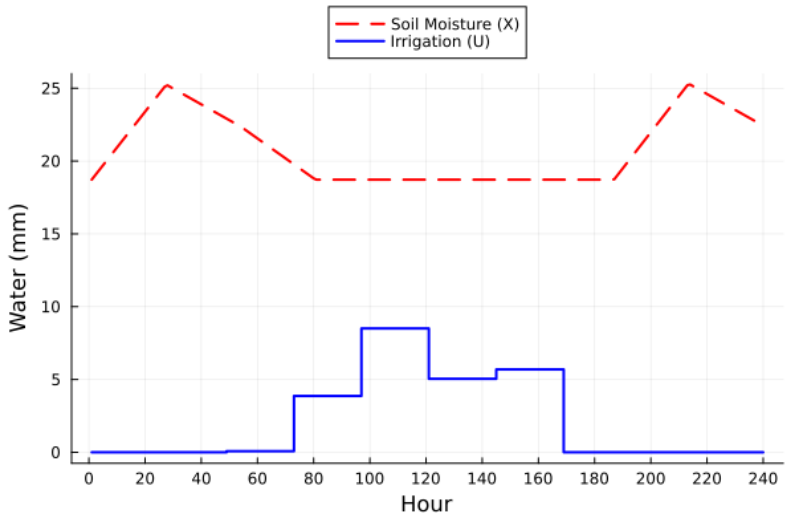


Figure 3.6: Result obtained with the same parameters for Scenario 1b

In this scenario, the results are in very close agreement with those of the original study. The higher temperature on day 5 causes a spike in evapotranspiration, prompting the controller to apply more irrigation around that day to prevent soil moisture from dropping below the minimum threshold. The irrigation schedule and soil moisture trajectory closely match those of the original study, with slight deviations, as in previous scenarios.

Despite these quantitative differences, the controller's overall behaviour is correct in both scenarios: soil moisture remains above the minimum threshold throughout the planning period, ir-

rigation is applied only when necessary, and the model responds appropriately to the problem's physical parameters.

To evaluate the scalability of the implementation, the model was executed with planning horizons of 1000 and 10000 days, using the original 10-day weather data repeatedly. The results are summarised in Table 3.4.

Horizon (days)	**Without parallelisation (s)**	**With parallelisation (s)**
1000	8.3	2.1
10000	83.2	21.5

Table 3.4: Execution times with and without parallelisation, using 4 processor cores.

The execution time scales approximately linearly with the number of time steps in both cases. With 4 processor cores, the parallelised solver achieves a speed-up of approximately 3.9, consistent across both problem sizes. This indicates that the parallelisation overhead is negligible and that the implementation scales efficiently with the number of available cores.

These results confirm that the implementation is computationally feasible for large problems.

3.4 Summary

This chapter described the implementation of the single-level irrigation optimisation model of Pereira et al. in Julia, using JuMP as the modelling interface and IPOPT as the solver. The evapotranspiration value was computed daily using the Penman-Monteith equation [22].

The implementation was validated against three scenarios from the original study: a flat-terrain base case, a sloped terrain with runoff losses, and a temperature spike scenario. In all cases, the controller reproduces the correct qualitative behaviour. Soil moisture remains above the minimum threshold, irrigation is applied only when necessary, and the model responds appropriately to changes in terrain slope, runoff, and temperature. Minor quantitative deviations were observed, attributed to differences between the Penman-Monteith and ASCE evapotranspiration formulations.

Performance tests with planning horizons of 1000 and 10000 days confirmed that execution time scales linearly with problem size, and that parallelising the linear algebra operations within IPOPT achieves a near-linear speed-up of approximately 3.9 times on 4 processor cores. This validated implementation serves as the foundation for the bilevel extension developed in the following chapter.

Chapter 4

Bilevel Implementation

This chapter describes the extension of the single-level irrigation optimisation model of Chapter 3 into a bilevel framework.

Section 4.1 introduces the additional tools used to implement the bilevel model, namely BilevelJuMP, the linear solvers tested alongside IPOPT, and the Deucalion supercomputer used to run the experiments. Section 4.2 presents the formal bilevel problem formulation, detailing both the upper-level and lower-level optimisation problems. Section 4.3 describes the implementation details and the solver configurations tested. Finally, Section 4.4 presents the results, including both simulation outcomes and a comparison of the solvers' performance across the tested configurations.

4.1 Tools

Similarly to Chapter 3, the implementation makes use of Julia and JuMP. Although it is possible to solve a bilevel optimisation problem using JuMP alone, doing so requires manually reformulating the problem as a single-level MPEC by deriving the KKT conditions of the lower level by hand and introducing the corresponding dual variables and complementary slackness constraints into the model. This section describes the additional tools used in this chapter to automatically handle reformulation and address the computational challenges arising in the bilevel setting.

4.1.1 BilevelJuMP

BilevelJuMP [25] is an extension of the JuMP modelling language designed specifically for bilevel optimisation problems. It allows the upper- and lower-level problems to be declared separately

within a single model object, using syntax that closely mirrors the way a single-level JuMP model is built, with shared variables and parameters referenced naturally across the two levels.

At the point of declaration, the model remains conceptually bilevel: the upper and lower levels are still two distinct, nested optimisation problems. However, IPOPT cannot solve a nested optimisation problem. It expects a single, flat set of variables, a single objective function, and a single set of constraints. BilevelJuMP bridges this gap by automatically constructing the KKT conditions of the lower-level problem and substituting them for the lower-level problem itself.

Once this substitution is complete, what remains is the upper-level objective and constraints, together with the KKT conditions of the lower level acting as additional constraints, forming a single, ordinary nonlinear program, the MPEC introduced in Section 2.3.4. It is this MPEC, rather than the original bilevel formulation, that is passed to IPOPT. This process removes the need to derive or implement the KKT conditions manually, which is particularly valuable when the lower-level problem involves many constraints, as is the case here.

4.1.2 Linear Solvers

At each interior-point iteration, IPOPT must solve a large sparse linear system, the KKT system introduced in Section 2.1.3, to compute the search direction for that iteration. This is done by an external linear solver, a software library responsible specifically for factorising and solving systems of linear equations, separate from IPOPT's own logic for managing the nonlinear iterations. The choice of linear solver does not change the mathematical problem being solved, but it can substantially affect computation time and numerical robustness, since different solvers use different pivoting strategies and sparsity-exploiting techniques.

By default, IPOPT uses the Multifrontal Massively Parallel sparse direct Solver (MUMPS) linear solver. IPOPT also supports the Harwell Subroutine Library (HSL) collection of sparse linear solvers [26], which are commonly used alongside IPOPT in the literature on numerical optimisation as an alternative to MUMPS. Among the solvers available in HSL, MA97 is of particular interest, since it is designed to exploit sparsity efficiently on large systems. Given that the bilevel formulation developed in this chapter increases the size and density of the KKT system relative to the single-level model of Chapter 3, both MUMPS and MA97 were tested, with the results of this comparison presented in Section 4.4.

4.1.3 Deucalion Supercomputer

All computations described in this chapter were executed on the Deucalion supercomputer [27], hosted by the Minho Advanced Computing Center in Guimarães, Portugal. Deucalion is a heterogeneous system, the first in Europe to combine Advanced RISC Machine (ARM) and x86 microprocessors within the same cluster, and consists of two general-purpose compute partitions based on these architectures, alongside a separate GPU partition. The ARM partition consists of 1,632 Fujitsu PRIMEHPC FX700 nodes, each equipped with a single Fujitsu A64FX processor (48 cores, 2.0 GHz) and 32 GB of in-processor HBM2 memory, with a combined peak performance of approximately 5 PFlops. The x86 partition consists of 500 Atos Bull Sequana nodes, each equipped with two AMD EPYC 7742 processors (64 cores each, 2.25 GHz) and 256 GB of DDR4 memory, with a combined peak performance of approximately 2.3 PFlops. All nodes are connected via an InfiniBand fat-tree network for high-speed communication and to a Lustre parallel file system for shared storage. The system runs Rocky Linux 8 and uses Slurm for resource management and job scheduling.

Deucalion was used in this work for two main reasons. First, the parameter study described in Section 4.3 required running the model across multiple solver configurations and problem sizes, and the resulting number of runs would have been impractical to execute sequentially on a personal machine within a reasonable time frame. Access to a shared HPC system allowed these runs to be distributed and completed in parallel. Second, Deucalion's combination of ARM and x86 partitions within a single cluster enabled testing the same model on two distinct processor architectures without requiring access to two separate systems, allowing any architecture-dependent differences in solver behaviour or performance to be assessed directly.

4.2 Problem Formulation

The main objective of this dissertation is to formulate a bilevel problem, as described in Section 2.3, that is able to distribute the water available efficiently by use of a bilevel model that is able to process this input data with the same weather data as the model described by Pereira et al. [13] and a daily limited amount of water. The bilevel model separates the irrigation control problem into two hierarchical levels. The upper level acts as a central coordinator that sets a dynamic soil moisture safety threshold $X_{\min,j,i}$ for each field sector $j \in \{1,\ldots,N\}$ at each time step $i \in \{1,\ldots,M\}$. The lower level, given those thresholds, solves a joint optimal control problem

across all fields to minimise total water consumption while respecting the soil moisture dynamics and the thresholds defined by the upper level.

The most important difference from the single-level formulation is that $X_{\min,j,i}$ is now a decision variable set by the upper-level daily, rather than a fixed value for every algorithm iteration. This allows the upper level to lower soil moisture target values during periods of water scarcity, enabling controlled deficit irrigation rather than excessive water use, as in previous models. However, lowering the soil moisture target value below the ideal value is penalised in the upper-level objective, therefore, the thresholds will be reduced to the minimum amount possible.

4.2.1 Upper Level: Strategic Resource Coordinator

Let need_j denote the ideal soil moisture target for field j, representing the agronomic requirement for full plant health. The upper-level objective is to minimise the aggregate squared deviation of the dynamically adjusted threshold $X_{\min,j,i}$ from need_j across all fields and time steps:

$$\min_{X_{\min},\,\text{deviation}} J = \sum_{j=1}^{N}\sum_{i=1}^{M} (\text{deviation}_{i,j})^2 \tag{4.1}$$

subject to the following constraints:

The shared daily water delivery capacity C_{total} limits the aggregate irrigation applied across all fields at each time step:

$$\sum_{j=1}^{N} u_{i,j} \leq C_{\text{total}}, \quad \forall i = 1,\ldots,M. \tag{4.2}$$

The threshold is bounded below by zero and above by the agronomic requirement:

$$0 \leq X_{\min,j,i} \leq \text{need}_j, \quad \forall j,i. \tag{4.3}$$

The slack variable $\text{deviation}_{i,j}$ tracks the difference between the ideal requirement and the threshold actually set by the upper level:

$$\text{deviation}_{i,j} = \text{need}_j - X_{\min,j,i}, \quad \text{deviation}_{i,j} \geq 0, \quad \forall j,i. \tag{4.4}$$

Finally, the irrigation schedules $(u_{i,j})$ and soil moisture trajectories $(x_{i,j})$ that appear in constraint (4.2) must be optimal responses of the lower level to the threshold matrix $X_{\min}$:

$$(u,x) \in \Psi(X_{\min}), \tag{4.5}$$

where $\Psi(X_{\min})$ denotes the set of solutions of the problem of the lower-level (4.6)–(4.12), parameterised by $X_{\min}$. This is the formal expression of the hierarchy between the two levels: the upper level chooses $X_{\min}$, and u and x are then fixed by the lower level's response to that choice, rather than being free variables of the upper-level problem itself.

The quadratic penalty structure in (4.1) means that deviations are penalised increasingly as they grow, which causes the upper level to distribute water stress as evenly as possible across fields rather than concentrating it on a single sector. This way, the model ensures that the deficit in each field is as balanced as possible, thereby mitigating total yield losses. When C_{total} is large enough that all fields can be maintained at their full soil moisture requirements, the optimal solution sets $X_{\min,j,i} = \text{need}_j$ for all j and i, with zero total deviation. When supply is restricted, the upper level reduces soil moisture target values across sectors to minimise the sum of squared deviations.

4.2.2 Lower Level: Operational Optimal Control

The lower level acts as the agent that executes localised irrigation. For a fixed soil moisture threshold matrix $X_{\min}$ determined by the upper level, the lower-level problem minimises total water use across all N fields over the planning horizon of M time steps:

$$\min_{u,x,\text{loss},\text{overflow}} \sum_{j=1}^{N} \sum_{i=1}^{M} u_{i,j} \tag{4.6}$$

subject to the soil moisture dynamics and physical constraints defined in the following.

The soil moisture at each field must remain at or above the threshold set by the upper level:

$$x_{i,j} \geq X_{\min,j,i}, \quad \forall i, j. \tag{4.7}$$

The initial soil moisture is fixed at a measured value $x_{S,j}$:

$$x_{1,j} = x_{S,j}, \quad \forall j. \tag{4.8}$$

The soil moisture dynamics follow the mass balance equation of Pereira et al. [13], adapted to the multi-field setting:

$$x_{i+1,j} = x_{i,j} + h \cdot f(t_i, x_{i,j}, u_{i,j}, x_{i+1,j}), \quad \forall i = 1, \ldots, M-1, \forall j, \tag{4.9}$$

where the function f incorporates the effects of irrigation, evapotranspiration, rainfall, slope-induced run-off, deep percolation, and overflow:

$$\begin{aligned} f(t_i, x_{i,j}, u_{i,j}, x_{i+1,j}) = K_I u_{i,j}\left[1 - R_F \sin(\alpha_j)\right] - K_C \cdot \mathrm{ET}_{0,i} \\ + \left[1 - R_F \sin(\alpha_j)\right]\mathrm{Rainfall}(t_i) - \mathrm{loss}_{i,j} - \mathrm{overflow}_{i,j}. \end{aligned} \tag{4.10}$$

In the single-level model of Chapter 3, water loss in each field was governed by a conditional function, switching between deep soil percolation when $x_{i,j} \leq x_{FC}$ and saturation overflow when $x_{i,j} > x_{FC}$. This conditional form was used directly in Chapter 3 without difficulty, since IPOPT was able to solve the resulting model despite the function not being differentiable at $x_{i,j} = x_{FC}$. In the bilevel setting, however, the same condition caused the model to fail, as BilevelJuMP reformulates the lower level by substituting its KKT conditions for the lower-level problem, and deriving these conditions requires differentiating the lower-level constraints. Since the conditional is not differentiable at $x_{i,j} = x_{FC}$, the KKT reformulation could not be constructed correctly at that point, and the resulting program failed to solve reliably.

To avoid this, the conditional was replaced with two separate non-negative variables, $\mathrm{loss}_{i,j}$ and $\mathrm{overflow}_{i,j}$, each governed by an inequality rather than a conditional equality:

$$\mathrm{loss}_{i,j} \geq k x_{i,j}, \quad \forall j,\ i = 1, \ldots, M-1, \tag{4.11}$$

$$\mathrm{overflow}_{i,j} \geq (x_{i+1,j} - x_{FC}) C_S, \quad \forall i, j. \tag{4.12}$$

Both inequalities are differentiable everywhere, since neither depends on which branch of the original condition would have been active. Unlike the original formulation, where loss and overflow are mutually exclusive, this decomposition allows both terms to be active simultaneously once $x_{i,j} > x_{FC}$. The loss variable remains pinned to $k x_{i,j}$, while the overflow variable additionally becomes active once moisture exceeds field capacity. This follows directly from the requirement of smooth constraints in the KKT reformulation. Above field capacity, the model therefore accounts for a small additional water loss relative to the original formulation, since the loss term is not switched off in this regime.

4.3 Implementation

The bilevel model was implemented in Julia using BilevelJuMP, with IPOPT as the underlying solver, following the tools introduced in Section 4.1. All runs were executed on the Deucalion

supercomputer, on both the ARM and x86 partitions.

A number of IPOPT parameters were fixed across all runs. The maximum number of iterations was set to $3 \cdot 10^5$, well beyond what any run was expected to need, so that no run would be cut off prematurely by the iteration limit rather than by genuine convergence or failure. The `acceptable_iter` parameter, which allows IPOPT to terminate early if an acceptable, rather than fully optimal, solution persists for a given number of consecutive iterations, was set to 15. The `nlp_scaling_method` was set to `gradient-based`, IPOPT's default automatic scaling scheme, which rescales the problem internally so that the magnitudes of the objective and constraints are comparable. When MA97 was used as the linear solver, MC64 scaling was applied, a permutation-and-scaling strategy intended to improve the numerical conditioning of the KKT matrix before factorisation.

Since the KKT-based MPEC reformulation introduces complementarity conditions that may cause difficulties for interior-point solvers, more parameters were varied in an attempt to find a configuration that converged reliably across a range of problem sizes. Two linear solvers were tested for the KKT factorisation step at each interior-point iteration, MUMPS and MA97. Two settings for the `bound_push` and `bound_frac` parameters were also tested, 10^{-6} and 0.01. All combinations of these varied settings were tested across the range of problem sizes described in Section 4.4, with each configuration run to successful termination.

Two starting guess schemes were tested in addition to the different solver parameters. The first left all variables at IPOPT's default initialisation. The second set $X_{\min,j,i}$ to the agronomic requirement need$_j$ for every field and time step, and set the irrigation variables $u_{i,j}$ to zero, leaving all remaining variables, including the dual variables introduced by the KKT reformulation, at IPOPT's default.

All runs were submitted as Slurm batch jobs on the relevant Deucalion partition, ARM or x86, depending on the configuration being tested, rather than executed interactively. Each run produced a log file recording the solver's termination status, iteration count, and total execution time, which were collected and compared in Section 4.4.

4.4 Results and Discussion

This section presents the problem outcome after configuration with the different parameters described in Section 4.3, comparing the linear solvers, barrier update strategies, and bound-related parameters across the range of problem sizes considered in this chapter.

4.4.1 Simulation

The findings detailed below have been accepted for presentation at an upcoming conference and will soon be published in its proceedings [28].

The following graphs show tests over 10 days, where the x-axis represents the current day and the y-axis represents the amount of water (in mm) used or required by the corresponding system. The dark blue bar represents actual soil moisture at the time of measurement. The dashed yellow line represents the target soil moisture $x_{\min}$ set by the upper level, which may be reduced to accommodate the constraints. The green line represents the ideal amount of water for the plant to be healthy, $need_j$. The grey bars represent the amount of rainfall on a given day.

For the first test, the simulation was run over two fields using the same parameters as the first scenario of Pereira et al., as shown previously in Figure 3.1, and with a daily water limitation of $C_{\text{total}} = 6$ mm. In this case, the field inclination in both fields is $\alpha = 0°$. The optimisation trajectory obtained is shown in Figure 4.1.

Since all the initial parameters and inputs for both fields are the same, the solutions generated are also the same, as expected.

After the rainfall on day 1, the natural increase in soil moisture makes irrigation unnecessary for the following two days. However, on day 4, soil moisture falls below the minimum value, and the irrigation system will activate to make up the difference. Since this difference is slightly smaller than the defined C_{total}, the system can fully irrigate both fields to the desired soil moisture level on this day. However, from day 5 to day 8, the system is unable to provide sufficient water for the plants to maintain healthy hydration levels. As such, the upper levels of the system define reduced $x_{\min}$ values, and the lower levels define the u values required to achieve these reduced targets. The soil moisture target of both models is the same, as expected, since the upper level's main objective is to reduce the sum of the square deviation from the ideal soil moisture amount. In this specific case, irrigation for both fields also matches because the field parameters are identical. On day 8, irrigation is not necessary due to natural rainfall during the day. Lastly, on day 9, irrigation is necessary to prevent soil moisture from falling below the threshold the next day.

The second test uses the same parameters from the previous test, however, the slope of the second field was set to $\alpha_2 = 30°$. This allows us to observe how the model behaves when it must compare two fields with different parameter values. The optimisation trajectory obtained with these conditions is shown in Figure 4.2.

In this case, we can see that the graph of the second field has more difficulty retaining the

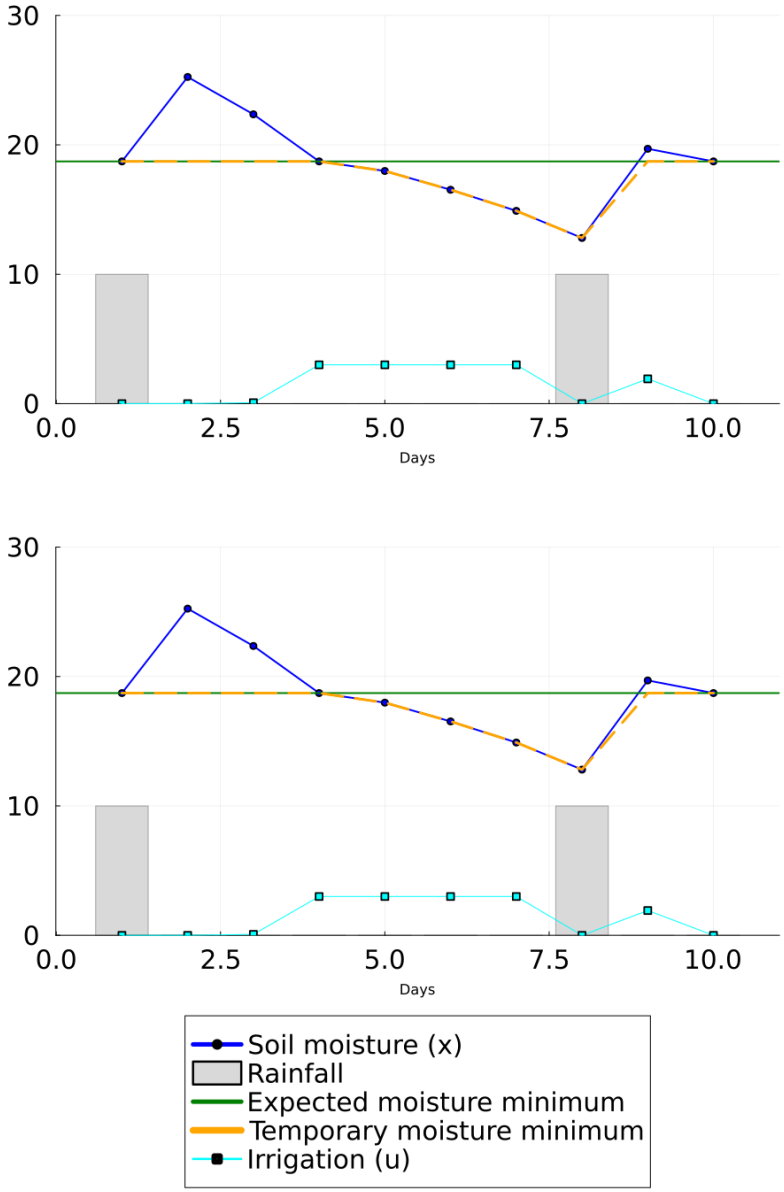


Figure 4.1: Irrigation profile for two identical fields under shared capacity constraints ($C_{\text{total}} = 6$ mm) [28].

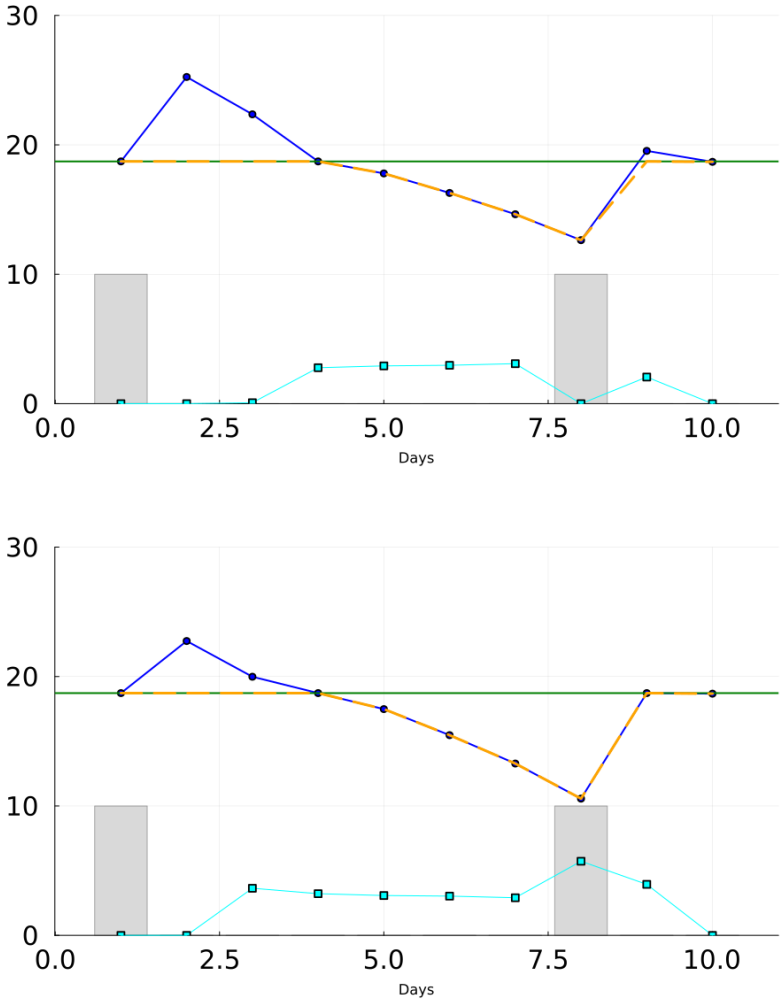


Figure 4.2: Irrigation profile for two fields with asymmetric slope topographies ($\alpha_1 = 0°$, $\alpha_2 = 30°$) [28].

water used to irrigate it. This is because of the added slope in that field. Water applied to it, either by irrigation or rainfall, raises soil moisture by less than the first field, where the slope is $\alpha_1 = 0°$. Because of these differences, irrigation is necessary one day earlier for field 2 than for field 1. The increased soil moisture losses due to the slope in the second field are particularly notable on day 8, when, despite rainfall, the model still irrigates the field, as the rainfall-induced increase in soil moisture is insufficient to reach the level required for plant health. Unlike the previous example, where the irrigation patterns for both fields were the same, the irrigation patterns for these fields differ, driven by these compounded differences.

To evaluate scalability, a third test was run, in which the model accounts for three different fields with slopes of $\alpha_1 = 0°$, $\alpha_2 = 15°$, and $\alpha_3 = 30°$. Additionally, for this test, a capacity limit of $C_{\text{total}} = 9$ mm was considered. The results are shown in Figure 4.3

In this example, we can see how the model correctly accounts for the new C_{total} value and the increased number of fields, generating an appropriate solution for each case despite their differing parameters. Although this test involves three fields, the model is not limited to this number and can be extended to handle more fields. The results are as expected: the model requires more irrigation for steeper fields, while flatter fields require much less. These run-off losses are directly related to each field's slope, as reflected in the timing of irrigation events. The steepest field, with its reduced capacity to retain applied water, depletes its soil moisture more rapidly than the others and, as a result, requires irrigation as early as day 2. The flattest field, by contrast, retains moisture longer, with soil moisture levels dropping below the minimum threshold only by day 3. This gap illustrates how sensitive the model is to terrain differences, even when all other parameters, such as rainfall and the ideal soil moisture target, remain identical across all three fields. These results confirm that the bilevel structure can balance competing water demands across an arbitrary number of fields under a shared, constrained water supply, regardless of their topographical differences.

4.4.2 Performance

This section presents the performance results obtained when testing the model under the solver configurations described in Section 4.3, over an increasing number of repetitions of the original 10-day data set. The objective of these tests is to evaluate how solver choice, barrier strategy, starting guess, and hardware architecture affect both runtime and solution quality as problem size grows.

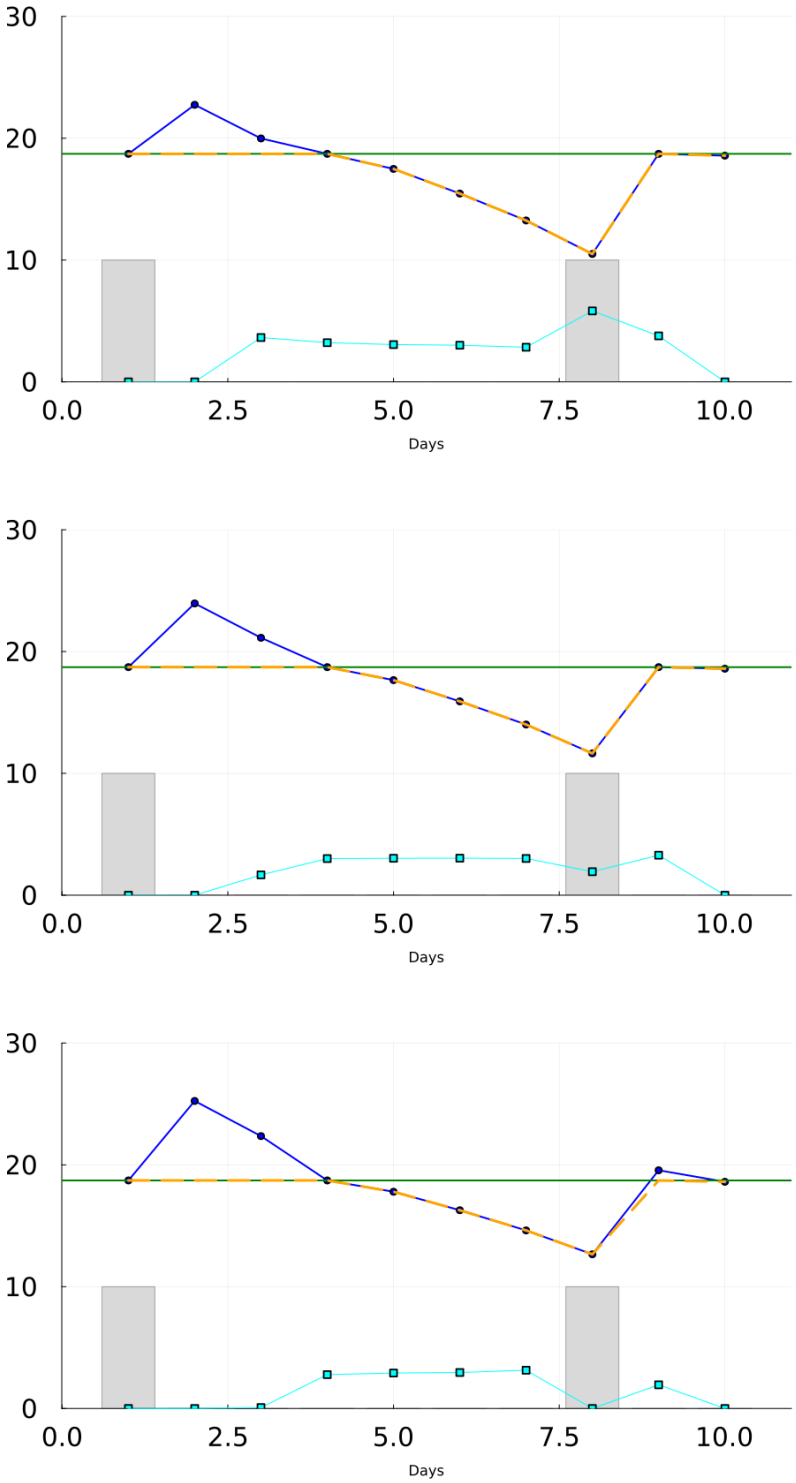


Figure 4.3: Scalability performance validation for three sectors with varying slope topographies ($\alpha = [30°, 15°, 0°]$) [28].

4.4.2.1 Comparison between ARM and x86 hardware

The same code, with the same solver parameters, was executed on both the ARM and x86 partitions of Deucalion to assess whether architecture choice affects solver performance. This comparison used the MUMPS linear solver, with `nlp_scaling_method` set to gradient-based and `bound_push`/`bound_frac` set to 0.01. The physical and environmental parameters were those of the first scenario described in Section 4.4.1 (slopes $\alpha = 0°$, daily capacity $C_{\text{total}} = 6$), restricted to two fields. Figure 4.4 compares the execution time between the problem sizes, and Figure 4.5 compares the corresponding iteration counts.

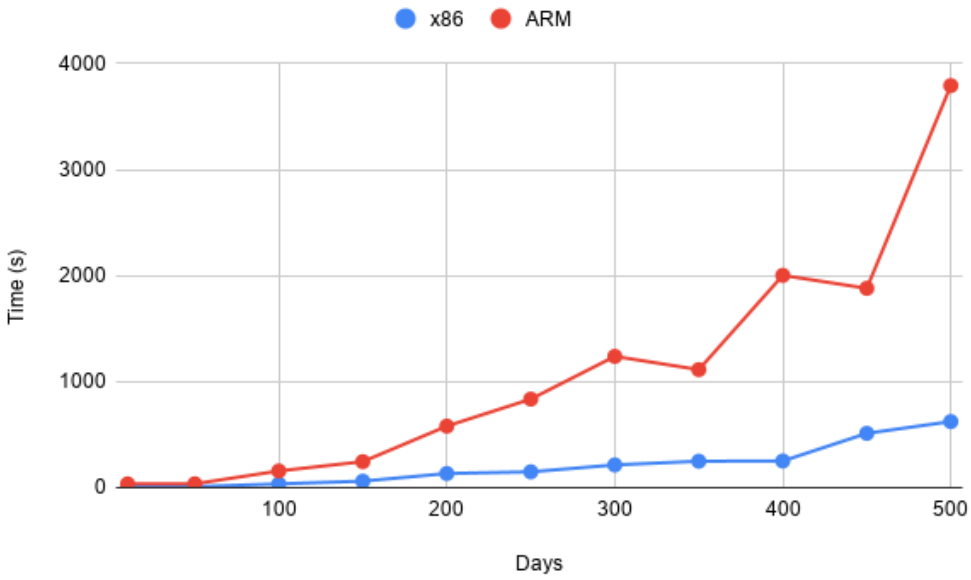


Figure 4.4: Execution time comparison between the ARM and x86 partitions of Deucalion across problem sizes.

The results show that the ARM partition is consistently slower than the x86 partition, even though the number of iterations required for convergence is similar between the two. To confirm that this difference was purely computational, the decision variables returned by both partitions, x^*_{min}, u^*, were compared directly and found to be identical in every test case. This confirms that the two architectures converge to the same solution, and that the observed gap is limited to runtime and iteration count rather than reflecting any difference in solution quality or the solver's convergence path. Given the consistency of this result across problem sizes, no further testing was conducted on the ARM partition, and all further solver comparisons in this section were run on the x86 partition.

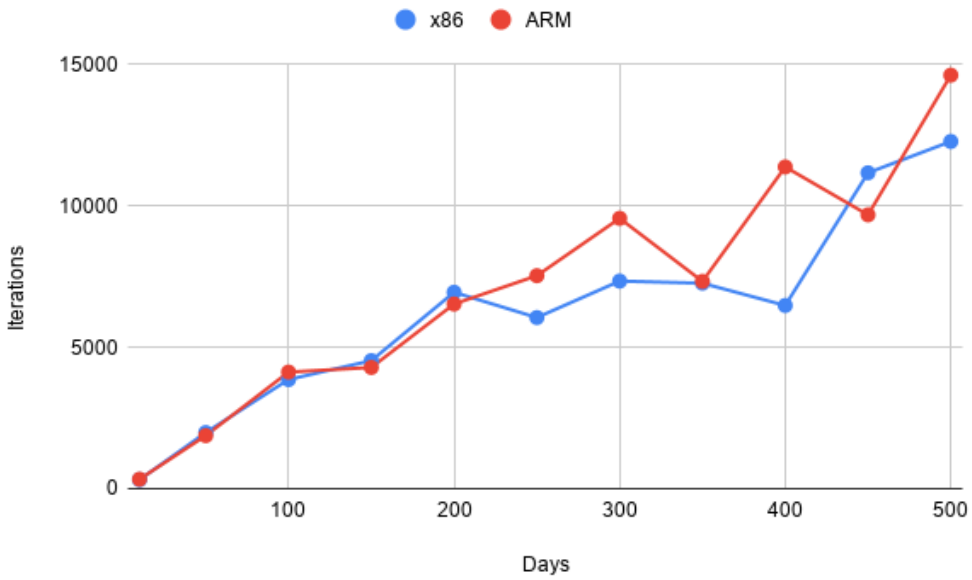


Figure 4.5: Iteration count comparison between the ARM and x86 partitions of Deucalion across problem sizes.

4.4.2.2 Comparison between Linear Solvers

This comparison isolates the effect of the linear solver, MUMPS versus MA97, on execution time and iteration count, across three parameter sets, all using the adaptive barrier update strategy:

- Parameters 1: `bound_push`/`bound_frac` $= 10^{-6}$, default starting guess.
- Parameters 2: `bound_push`/`bound_frac` $= 10^{-6}$, informed starting guess ($X_{\min,j,i} = \text{need}_j$ and $u_{i,j} = 0$ for all i, j, as described in Section 4.3).
- Parameters 3: `bound_push`/`bound_frac` $= 0.01$, informed starting guess.

All runs in this comparison were executed on the x86 partition of Deucalion, using the first scenario described in Section 4.4.1, with problem size varied by repeating the underlying 10-day dataset up to 50 times, producing horizons of up to 500 days. Figures 4.6 and 4.7 show the execution time and iteration count for the parameter set 1.

For parameters 1, the two solvers perform similarly up to approximately day 250, after which MUMPS begins to scale worse than MA97. By day 500, MUMPS requires around 1050 seconds to converge, compared to approximately 560 seconds for MA97, despite MUMPS needing only around 1.6 times as many iterations, approximately 20500 as opposed to 12500, rather than a proportionally larger execution time increase. This suggests that the cost of MUMPS per iteration,

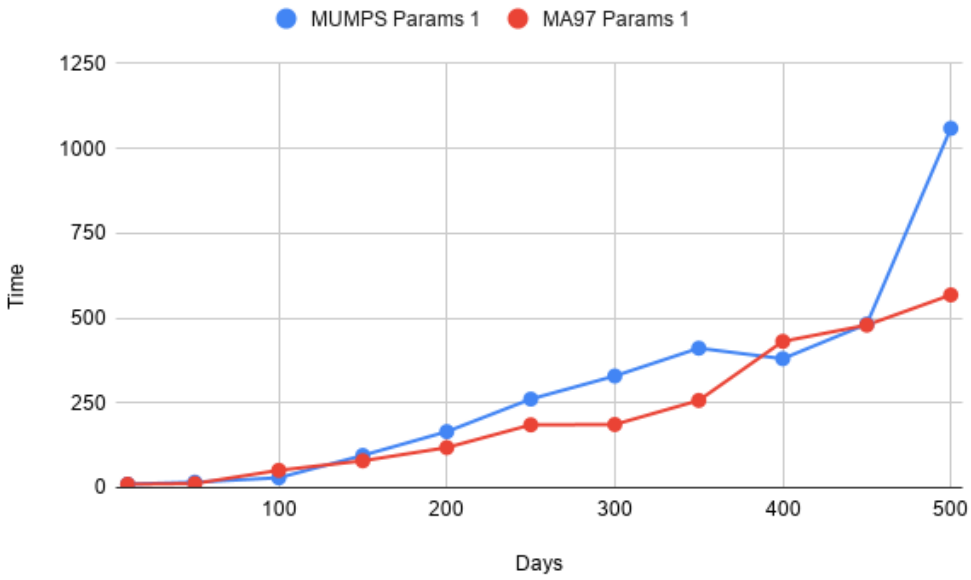


Figure 4.6: Runtime comparison between MUMPS and MA97 for parameters 1.

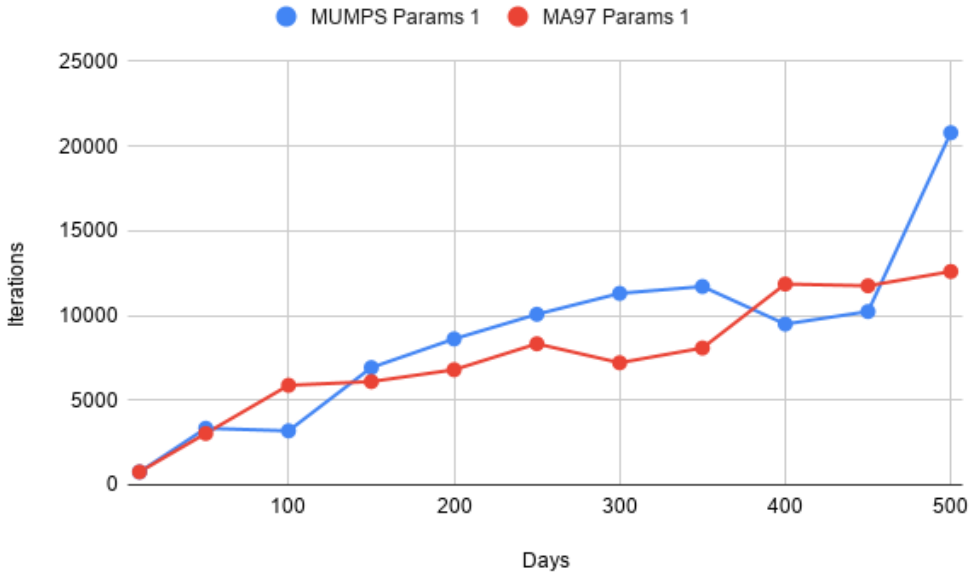


Figure 4.7: Iteration count comparison between MUMPS and MA97 for parameters 1.

rather than the number of iterations alone, is responsible for the increase in runtime gap at larger problem sizes. Figures 4.8 and 4.9 show the same comparison for parameters 2.

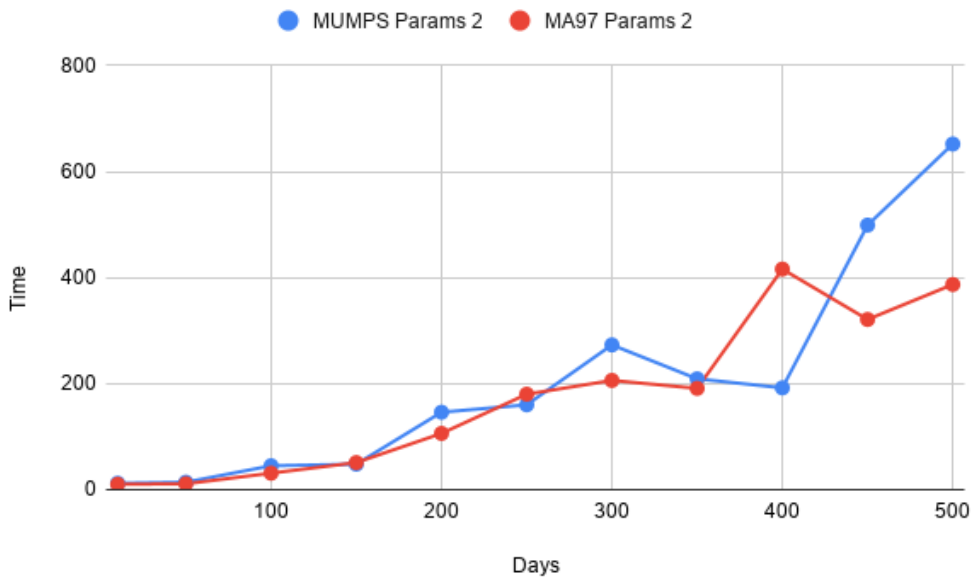


Figure 4.8: Runtime comparison between MUMPS and MA97 for parameters 2.

For parameters 2, the two solvers track each other closely up to day 400, with MA97 briefly exceeding MUMPS in both runtime and iterations around day 400. Beyond this point, MUMPS overtakes MA97, reaching approximately 650 seconds at day 500 against approximately 385 seconds for MA97. The corresponding iteration counts follow the same pattern, with MUMPS ending at around 13000 iterations, compared with approximately 8500 for MA97.

Figures 4.10 and 4.11 show the same comparison for parameters 3. Parameters 3 show a pattern similar to that of parameters 2. Runtime and iteration counts for the two solvers remain close up to around day 400, after which MUMPS diverges upward, reaching approximately 620 seconds at day 500 compared to approximately 380 seconds for MA97.

Across all three parameter settings, MA97 scales better than MUMPS as problem size increases, even though the two solvers perform comparably at smaller problem sizes. In each case, the gap that emerges at larger problem sizes is not matched by a proportional gap in iteration count, indicating that MA97's advantage lies in the cost of each individual iteration rather than in requiring fewer iterations overall to converge. On this basis, MA97 is identified as the preferred linear solver for the largest problem sizes considered in this chapter.

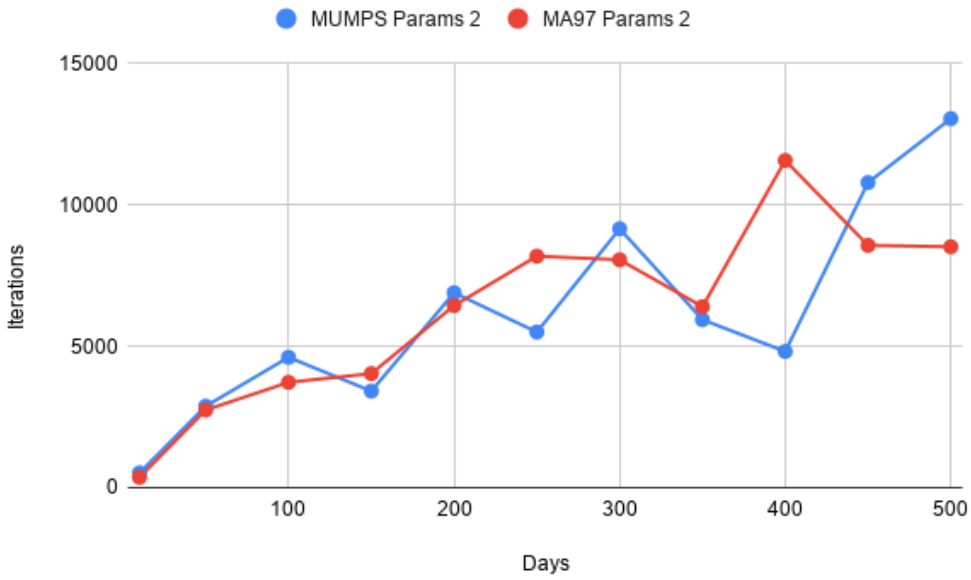


Figure 4.9: Iteration count comparison between MUMPS and MA97 for parameters 2.

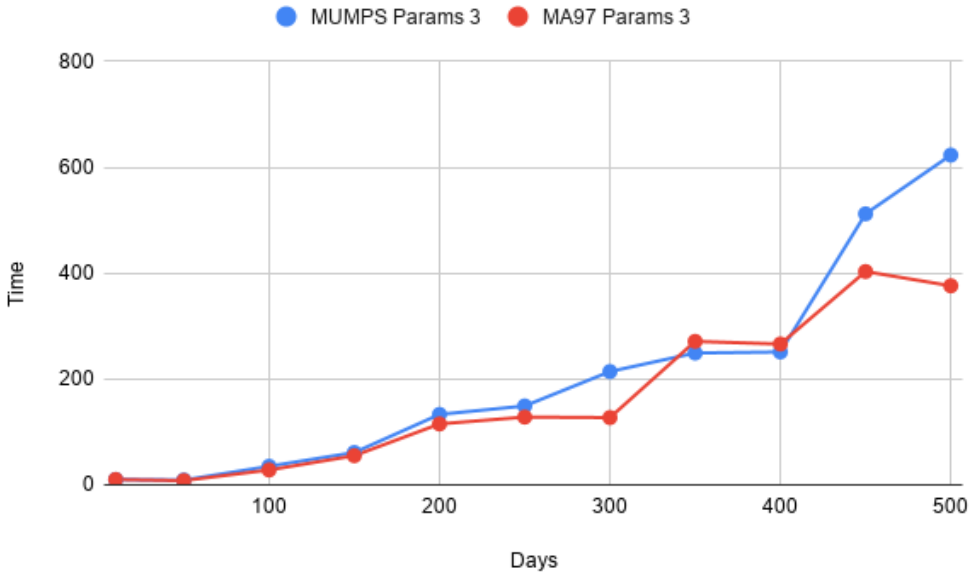


Figure 4.10: Runtime comparison between MUMPS and MA97 for parameters 3.

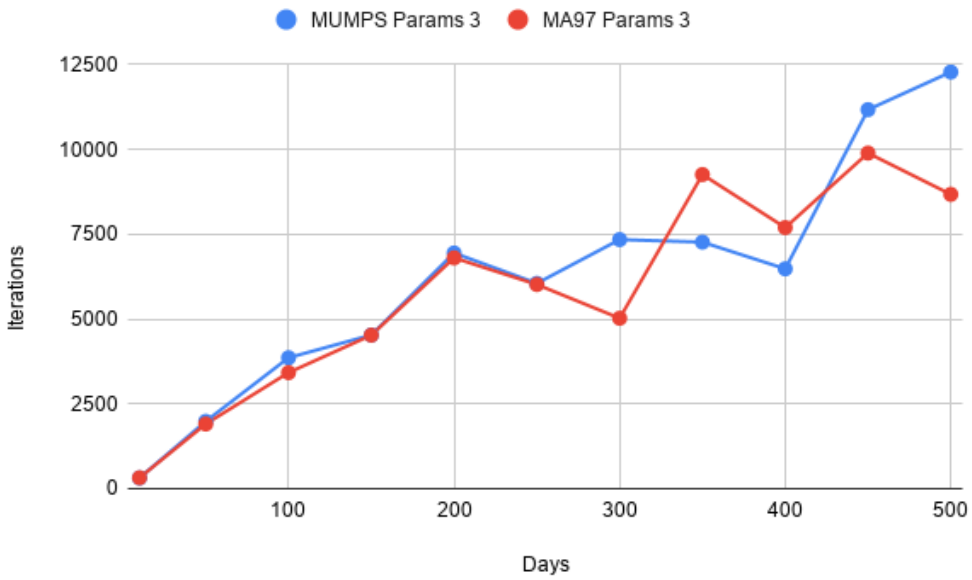


Figure 4.11: Iteration count comparison between MUMPS and MA97 for parameters 3.

4.4.2.3 Comparison of Parameter Settings

This subsection compares the effect of `bound_push`/`bound_frac` and the starting guess on solver performance, by comparing parameters 1, 2, and 3, as defined in Section 4.4.2.2, separately for MA97 and MUMPS.

Figures 4.12 and 4.13 show the execution time and iteration count for MA97 across all three parameter settings.

The MA97's execution time for parameters 1 reaches approximately 560 seconds by day 500, compared to approximately 385 seconds under parameters 2 and approximately 380 seconds under parameters 3. The corresponding iteration counts follow the same ordering: parameters 1 require around 12500 iterations at day 500, approximately 8500 for parameters 2, and approximately 8.700 for parameters 3. This indicates that, for MA97, switching from the default starting guess (parameters 1) to the informed starting guess (parameters 2) produces a meaningful reduction in both runtime and iteration count at larger problem sizes. By contrast, further change from `bound_push`/`bound_frac` $= 10^{-6}$ (parameters 2) to 0.01 (parameters 3), with the informed starting guess held fixed, produces very little additional change, with parameters 2 and parameters 3 remaining close to one another across the full range of problem sizes tested. This suggests that, when paired with MA97, the informed starting guess is the more influential of the two settings, while the choice of `bound_push`/`bound_frac` value has comparatively little effect on performance.

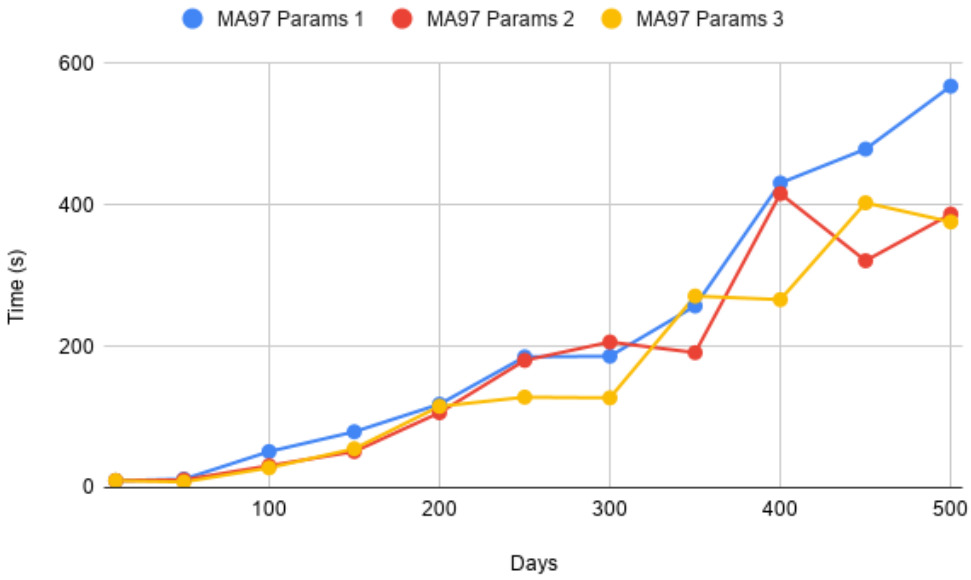


Figure 4.12: Runtime comparison for MA97 across parameters 1, 2, and 3.

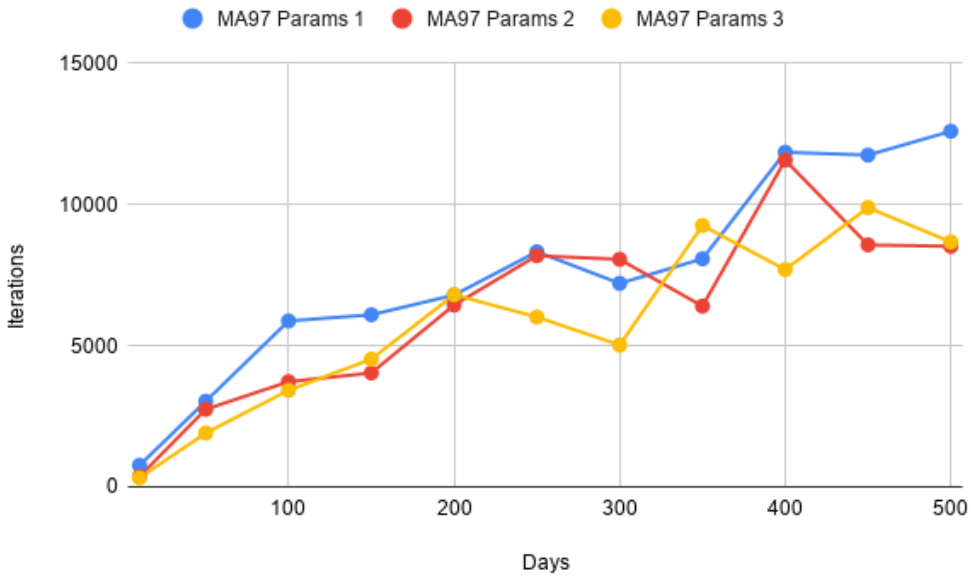


Figure 4.13: Iteration count comparison for MA97 across parameters 1, 2, and 3.

Figures 4.14 and 4.15 show the same comparison for MUMPS.

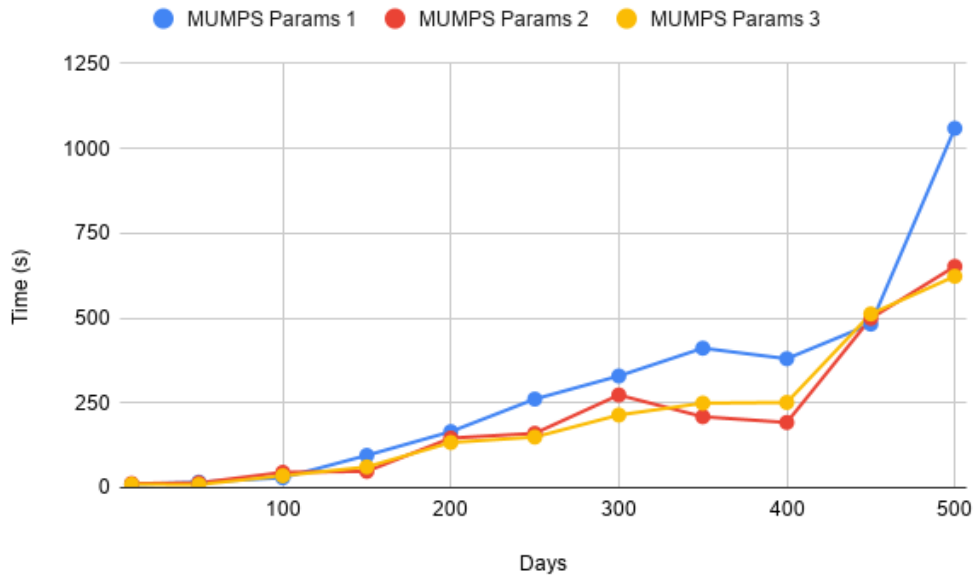


Figure 4.14: Runtime comparison for MUMPS across parameters 1, 2, and 3.

Under parameters 1, MUMPS reaches approximately 1050 seconds and around 20500 iterations by day 500. This is substantially higher than under parameters 2, where MUMPS reaches approximately 650 seconds and around 13000 iterations, and under parameters 3, where it reaches approximately 620 seconds and around 12300 iterations. As with MA97, the largest improvement comes from switching to the informed starting guess (parameters 1 to 2), while the subsequent change in `bound_push`/`bound_frac` (parameters 2 to 3) produces a much smaller reduction in both execution time and iteration count. Unlike MA97, however, MUMPS shows a more pronounced late-stage divergence under all three parameter settings, with runtime increasing sharply between days 400 and 500 in each case, even as iteration counts grow more gradually over the same interval. This reinforces the earlier observation in Section 4.4.2.2 that MUMPS's per-iteration cost, rather than the number of iterations required, is the main driver of its weaker scaling behaviour at larger problem sizes, regardless of which starting guess or `bound_push`/`bound_frac` value is used.

4.5 Summary

This chapter extended the single-level irrigation model of Chapter 3 into a bilevel framework, in which a strategic upper level dynamically sets soil moisture thresholds for each field, while an

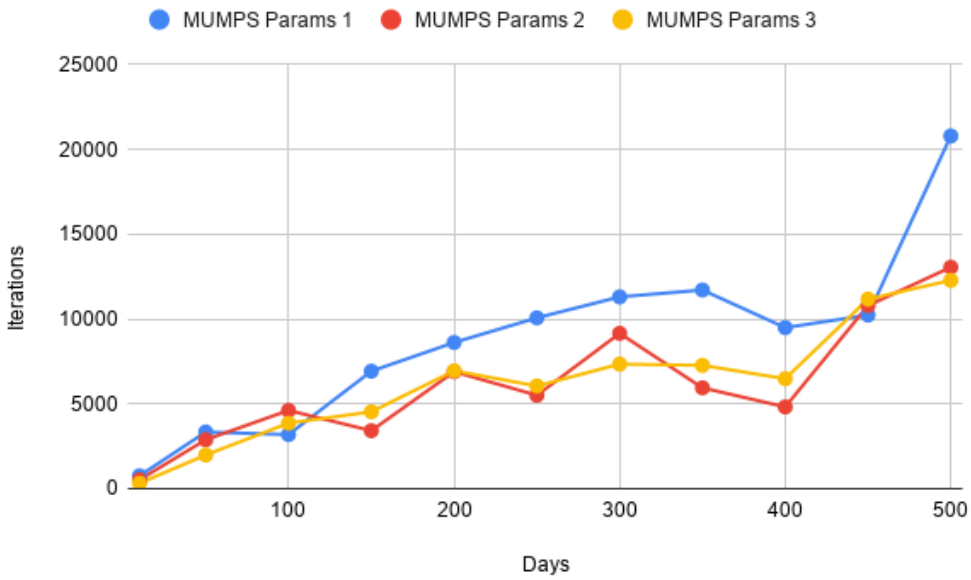


Figure 4.15: Iteration count comparison for MUMPS across parameters 1, 2, and 3.

operational lower level determines the irrigation schedule needed to meet those thresholds at minimum water cost. This formulation allows the model to perform controlled deficit irrigation during periods of water scarcity, distributing the resulting stress across fields rather than disproportionately depleting any single sector.

Implementing this bilevel structure required reformulating the original conditional loss and overflow terms as smooth inequalities, since the KKT-based reformulation performed by BilevelJuMP requires the lower-level constraints to be differentiable everywhere. It also required testing a range of solver configurations, including different linear solvers, barrier update strategies, and starting guess schemes, in order to identify a combination that converged reliably as the problem size increased.

The simulation results presented in Section 4.4.1 suggest that the bilevel model behaves as intended, reducing soil moisture targets only when necessary and distributing water stress according to each field's individual characteristics, such as slope-induced runoff. Fields with steeper slopes tended to require irrigation earlier and more frequently than flatter fields under the same rainfall and capacity conditions, and the upper level's threshold reductions during periods of water scarcity were consistent with minimising the aggregate squared deviation from each field's ideal soil moisture target.

The performance comparison in Section 4.4.2 suggests that hardware architecture affects execution time but not the solution obtained, with the ARM and x86 partitions of Deucalion converg-

ing to identical decision variables despite the x86 partition consistently completing runs faster. Comparing the linear solvers, MA97 appeared to scale better than MUMPS as the problem size increased, even though the two solvers performed comparably at smaller horizons, with the gap seemingly driven more by the cost of each individual iteration than by the number of iterations required to converge. Comparing starting guess schemes and `bound_push`/`bound_frac` settings similarly suggests that the informed starting guess, in which $X_{\min,j,i}$ is initialised at the agronomic requirement and $u_{i,j}$ at zero, is the more influential setting for both solvers, while the choice of `bound_push`/`bound_frac` value appears to have a comparatively minor effect once the informed starting guess is used.

These results suggest that the bilevel approach is computationally tractable and scalable to a large number of fields, providing a foundation for comparisons with alternative models presented in the next chapter.

Chapter 5

Conclusions

This dissertation set out to design, implement, and evaluate a high-performance computing framework for multi-field bilevel irrigation optimisation, motivated by the amount of agricultural water currently wasted in Portugal and by a limitation in existing smart irrigation models, which treat the minimum soil moisture threshold as a fixed value rather than a variable to be optimised.

To address this, we first reviewed optimisation theory and smart irrigation control, with particular attention to the single-level formulation of Pereira et al. [13], which minimises irrigation water use subject to soil moisture dynamics, but handles water allocation across multiple fields only reactively. This model was reimplemented in Julia using JuMP and IPOPT, and it was checked against three published scenarios: flat terrain, sloped terrain with runoff losses, and a temperature spike scenario. In all three cases, our implementation reproduced the behaviour reported in the original paper, with small differences attributable to using the Penman-Monteith equation rather than the ASCE reference evapotranspiration equation.

Building on this model, a bilevel extension was formulated in which an upper-level problem allocates a shared, limited daily water supply across fields by adjusting each field's soil moisture threshold, while a lower-level problem schedules irrigation for each field subject to the threshold assigned to it. This structure allows the model to reduce irrigation targets in situations of water scarcity in a controlled way, spreading the resulting shortfall across fields via a quadratic penalty on deviations from ideal agronomic targets, rather than leaving any single field without water or rendering the problem infeasible when supply cannot meet demand. The bilevel problem was implemented using BilevelJuMP, which reformulates the model as a single-level MPEC using the KKT conditions of the lower-level problem. This required replacing the original conditional loss and overflow terms with smooth inequality constraints, since the KKT reformulation requires the

lower-level constraints to be differentiable everywhere.

Simulation results show that the model behaves as intended. Soil moisture targets are reduced only when the shared capacity constraint is binding, and the size of the reduction reflects each field's characteristics rather than being applied evenly across fields. A comparison of solver configurations run on the Deucalion supercomputer across both its ARM and x86 partitions showed that the choice of hardware does not affect the solution obtained, but does affect runtime. The x86 partition was consistently faster than ARM despite similar iteration counts. Among the linear solvers tested, MA97 scaled better than the default MUMPS solver as the problem size increased, and this difference was driven by the cost per iteration rather than by differences in the number of iterations needed to converge. Of the solver settings tested, using an informed starting guess yielded the largest reductions in both runtime and iteration count, while changing `bound_push`/`bound_frac` led to a small but noticeable improvement.

Overall, these results show that the bilevel formulation extends existing single-level irrigation models and can be solved reliably at scale, using high-performance computing infrastructure to handle an arbitrary number of fields. Thus, this work contributes a validated Julia implementation, a bilevel model for allocating water across fields under a shared constraint, and a comparison of solver configurations that may be useful for similar optimisation problems in agriculture.

5.1 Future Work

Despite achieving all research goals, there remain many avenues for further research on this topic, including:

- The model is functionally limited by the non-linear and linear problem solvers used. Adopting different solvers could result in significantly higher solution speeds, though many of the best-performing options are proprietary and license-restricted, which may limit their accessibility for future work building on this dissertation.

- Manually deriving the single-level reformulation of the bilevel problem, rather than relying on BilevelJuMP's automated KKT-based reformulation, could also be a way of improving execution speed. A hand-crafted reformulation could be tailored to the problem structure, potentially avoiding unnecessary auxiliary variables, constraints, or big-M terms introduced by a general-purpose tool, and could also allow selection of solvers or formulations not

directly supported by BilevelJuMP. However, this approach would come at the cost of generality and would require considerably more implementation effort and domain-specific expertise.

- Further testing, experimentation, and fine-tuning of parameters could also be conducted as a means of improving performance or solution quality. This could include systematically varying solver tolerances, time limits, and presolve settings, as well as experimenting with different reformulation modes offered by BilevelJuMP, such as the big-M and SOS1-based approaches, to determine which combination yields the best trade-off between solution speed and accuracy for this particular class of problem.

- The model currently operates on a daily time step, consistent with the formulation of [13]. Extending the model to operate at finer time resolutions, such as half-hourly steps, could enable more responsive irrigation scheduling and a more accurate representation of rapidly changing conditions, such as short rainfall events. However, this would not be a straightforward change, since several parameters in the evapotranspiration calculation, such as the soil heat flux density G, were set to fixed values under the assumption that they are negligible over a full day, as discussed in Section 3.2. At finer time resolutions, these assumptions may no longer hold, and the corresponding formulas would need to be revisited and, likely, re-derived to remain accurate at sub-daily scales.

References

[1] H. Lee and J. Romero, "Climate change 2023: Synthesis report," in *Contribution of Working Groups I, II and III to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*, Geneva, Switzerland: IPCC, 2023. DOI: 10.59327/IPCC/AR6-9789291691647.001.

[2] S. I. Seneviratne et al., "Weather and climate extreme events in a changing climate," in *Climate Change 2021: The Physical Science Basis. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*, Cambridge, UK: Cambridge University Press, 2021.

[3] O. for Economic Co-operation and Development, "Water and agriculture: Portugal country note," OECD Publishing, Paris, Country Note, 2023, Accessed: 2026-01-13. [Online]. Available: https://www.oecd.org/content/dam/oecd/en/topics/policy-sub-issues/water-and-agriculture/oecd-water-policies-country-note-portugal.pdf.

[4] Agência Portuguesa do Ambiente, "Programa nacional para o uso eficiente da água (pnuea): Implementação 2012-2020," Agência Portuguesa do Ambiente, I.P., Lisbon, Portugal, Tech. Rep., 2012, Portuguese National Plan for the Efficient Use of Water. [Online]. Available: https://apambiente.pt/sites/default/files/_SNIAMB_Agua/DRH/PlaneamentoOrdenamento/PlanosGestaoSecaEscassez/PNUEA/PNUEA_2020_Jun2012.pdf.

[5] Fundação Calouste Gulbenkian and C-Lab, "O uso da água em portugal: Olhar, compreender e atuar com os protagonistas-chave," Fundação Calouste Gulbenkian, Lisbon, Portugal, Research Report, 2020, Comprehensive assessment detailing that 65% of Portuguese farmers utilize localized drip systems, while 71% operate without volumetric water meters. [Online]. Available: https://content.gulbenkian.pt/wp-content/uploads/2020/06/23155719/Uso-da-%C3%A1gua-em-Portugal_Estudo-Gulbenkian.pdf.

[6] G. Nikolaou, D. Neocleous, N. Katsoulas, and C. Kittas, "Irrigation of greenhouse crops," *Horticulturae*, vol. 5, no. 1, p. 7, 2019. DOI: 10.3390/horticulturae5010007.

[7] H. G. Jones, "Irrigation scheduling: Advantages and pitfalls of plant-based methods," *Journal of Experimental Botany*, vol. 55, no. 407, pp. 2427–2436, 2004. DOI: 10.1093/jxb/erh213.

[8] A. Younes, Z. E. Abou Elassad, O. El Meslouhi, D. E. Abou Elassad, and E.-d. A. Majid, "The application of machine learning techniques for smart irrigation systems: A systematic literature review," *Smart Agricultural Technology*, vol. 7, p. 100 425, 2024.

[9] S. Mukherjee, P. K. Dash, D. Das, and S. Das, "Growth, yield and water productivity of tomato as influenced by deficit irrigation water management," *Environmental Processes*, vol. 10, p. 10, 2023. DOI: 10.1007/s40710-023-00624-z.

[10] Y. Wang, K. Too Yok, W. Wu, A. R. Simpson, E. Weyer, and C. Manzie, "Minimizing pumping energy cost in real-time operations of water distribution systems using economic model predictive control," *Journal of Water Resources Planning and Management*, vol. 147, no. 7, p. 04 021 042, 2021.

[11] A. Goap, D. Sharma, A. K. Shukla, and C. R. Krishna, "An iot based smart irrigation management system using machine learning and open source technologies," *Computers and Electronics in Agriculture*, vol. 155, pp. 41–49, 2018. DOI: 10.1016/j.compag.2018.09.012.

[12] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," *IEEE transactions on evolutionary computation*, vol. 22, no. 2, pp. 276–295, 2017.

[13] R. Pereira, S. Lopes, M. F. Costa, N. Haie, and F. Fontes, "Modelling of smart irrigation with replan and redistribution algorithms," *Journal of Sustainable Development of Energy, Water and Environment Systems*, vol. 10, no. 3, pp. 1–17, 2022.

[14] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004, ISBN: 9780521833783. [Online]. Available: https://web.stanford.edu/~boyd/cvxbook/.

[15] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[16] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 2006.

[17] D. E. Kirk, *Optimal Control Theory: An Introduction* (Dover Books on Electrical Engineering). Dover Publications, 2004, ISBN: 9780486434841. [Online]. Available: https://store.doverpublications.com/products/9780486434841.

[18] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of Operations Research*, vol. 153, no. 1, pp. 235–256, 2007.

[19] S. Zhou, T. Hu, R. Zhu, F. Wu, and X. Wang, "A bilevel modeling approach for optimizing irrigation canal scheduling under a hierarchical institutional arrangement," *Agricultural Water Management*, vol. 284, p. 108 322, 2023. DOI: 10.1016/j.agwat.2023.108322.

[20] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017. DOI: 10.1137/141000671.

[21] M. Lubin, O. Dowson, J. Dias Garcia, J. Huchette, B. Legat, and J. P. Vielma, "JuMP 1.0: Recent improvements to a modeling language for mathematical optimization," *Mathematical Programming Computation*, 2023. DOI: 10.1007/s12532-023-00239-3.

[22] L. S. Pereira, "Necessidades de água e de rega das culturas: Uso de modelos," *Lisboa, Sf*, p. 18, 2010.

[23] R. G. Allen, L. S. Pereira, D. Raes, and M. Smith, *Crop Evapotranspiration: Guidelines for Computing Crop Water Requirements* (FAO Irrigation and Drainage Paper 56). Rome: Food and Agriculture Organization of the United Nations, 1998, ISBN: 92-5-104219-5.

[24] R. Allen et al., "The asce standardized reference evapotranspiration equation. 2005," *Technical Committee report to the Environmental and Water Resources Institute of the American Society of Civil Engineers from the Task Committee on Standardization of Reference Evapotranspiration, ASCE*, 2005.

[25] J. Dias Garcia, G. Bodin, and A. Street, "BilevelJuMP.jl: Modeling and Solving Bilevel Optimization Problems in Julia," *INFORMS Journal on Computing*, pp. 1–9, 2023. DOI: https://doi.org/10.1287/ijoc.2022.0135.

[26] HSL, *A collection of Fortran codes for large-scale scientific computation*, http://www.hsl.rl.ac.uk, 2026.

[27] MACC/FCCN, *Deucalion user guide: Architecture*, https://docs.macc.fccn.pt/deucalion/, 2024.

[28] R. Chertovskih, N. Khalil, and L. Sousa, "Bilevel optimization for smart irrigation control: A framework for enhancing water-use efficiency in agriculture," in *Proceedings of the 17th APCA International Conference on Automatic Control and Soft Computing (CONTROLO 2026), to be held in Coimbra, Portugal, September 9–11, 2026*. [Online]. Available: https://arxiv.org/abs/2605.31133.

Appendix A

Algorithm for the Single-Level Problem

This appendix lists the Julia implementation of the single-level irrigation optimisation model, built using JuMP with IPOPT as the underlying solver, presented in Chapter 3.

The code includes a loop that repeats the base ten-day weather and rainfall pattern a variable number of times, so that the model can be tested over increasingly long planning horizons within a single execution. This allows the solver's performance to be assessed as problem size increases.

Algorithm 1 Single-Level Scheduling for Smart Field Irrigation

Input:

Time horizon M

Fixed threshold: $x_{\min} \in \mathbb{R}$

Initial moisture: $x_S \in \mathbb{R}$

Profiles: rain, wind, temp $\in \mathbb{R}^M$

Physics: $K_I, K_C, \alpha, R_F, x_{FC}, k, c_S$

Output:

Irrigation schedule: $u \in \mathbb{R}^M$

State trajectory: $x \in \mathbb{R}^M$

{Phase 1: Environmental Pre-processing}

for $i = 1$ **to** M **do**

  Calculate reference evapotranspiration $\mathrm{ET}_{0,i}$ via FAO-56.

end for

{Phase 2: Single-Level Optimization Setup}

Initialize model via `JuMP` with `Ipopt` optimizer.

Define decision variables: u_i, x_i

Set Objective: Minimize total water used (2.7)

Enforce Constraints:

- Initialize baseline moisture state: $x_1 = x_S$
- Enforce fixed threshold: $x_i \geq x_{\min}$, $\forall i \in 1 \ldots M$
- Non-negativity of irrigation: $u_i \geq 0$

Evaluate combined loss/overflow expression (2.10) (piecewise, via boolean indicator):

- If $x_i \leq x_{FC}$: percolation loss $k \cdot x_i$
- If $x_i > x_{FC}$: overflow loss $c_S \cdot (x_{i+1} - x_{FC})$

Evaluate mass-balance soil state transition ($\forall i \in 1 \ldots M-1$) (2.8)

{Phase 3: Telemetry & Execution}

Invoke optimization via IPOPT.

if Solver Status is Optimal **then**

  Extract decision vectors u^*, x^*

  Interpolate daily trajectories to hourly resolution.

  Compute total water use and plot soil-moisture/irrigation trajectories.

  return u^*, x^*

else

  return `Optimization Failure Flag`

end if

Listing A.1: Single-level irrigation scheduling script

```
# --- Constants ---
xs = 18.72 # initial soil moisture (mm)
Kl = 0.85 # irrigation coefficient (0 to 1) (default value for sprinkler irrigation)
Kc = 0.85 # crop coefficient (0 to 1) (grass reference crop value)
Rf = 0.5 # runoff kept (0 to 1)
α= 20 * π/ 180 # soil inclination in deg (converted to rad)
cs = 1 # excess water lost (0 to 1)
xfc = 46.8 #soil maximum saturation (mm)
xmin = 18.72 # minimum humidity for plant (mm)
h = 1.0 # step between measurements (in days)
k = 0.05 # constant related to permeability of the soil used for loss calculation

tempos = Float64[]

# --- Data ---
reps = [1,10,100,1000,10000]

using HSL_jll
using JuMP
using Ipopt
using LinearAlgebra

hsl_lib = get(ENV, "COINHSL_LIB", "")

BLAS.set_num_threads(1) # avoid oversubscription
ENV["OMP_NUM_THREADS"] = "1" # threads for OpenMP linear solvers (MA86/MA97)

for repTimes in reps

   rainfall = repeat([10, 0, 0, 0, 0, 0, 0, 10, 0, 0],repTimes)
   windspeed = repeat([2.8, 3, 2.4, 2.4, 3, 2.2, 2.7, 2, 2.2, 2.8],repTimes)
   mintemp = repeat([17.4, 17.1, 15.2, 15.3, 15.1, 15.6, 16.3, 16.3, 15.4, 14.7],repTimes)
   maxtemp = repeat([24.4, 20.8, 23.1, 22.2, 24.9, 26.9, 29.3, 23.6, 19.4, 20.4],repTimes)

   M = length(rainfall)
   t = 1:M

   function evpt0(i)
      Tmin = mintemp[i]
      Tmax = maxtemp[i]
      u2 = windspeed[i]
      Tmean = (Tmax + Tmin) / 2

      # Grass constants
      α2 = 0.23 # grass albedo
```

```
    γ= 0.06665 # psicometric constant (kPa/°C)
    G = 0.0 # soil heat flow (MJ/m²/dia)
    kR = 0.16 # Hargreaves coefficient (0.19 if coastal)

    e0(T) = 0.6108 * exp((17.27 * T) / (T + 237.3))

    es = e0(Tmean)
    ea = e0(Tmin)
    ediff = es - ea

    Δ= (4098 * (0.6108 * exp((17.27*Tmean) / (Tmean + 237.3)))) / ((Tmean + 237.3)^2)

    Ra = 28
    Rs = kR * sqrt(Tmax - Tmin) * Ra
    Rn = (1 - α2) * Rs

    ET0 = (0.408 * Δ* (Rn - G) + γ* (900 / (Tmean + 273)) * u2 * ediff) /
        (Δ + γ* (1 + 0.34 * u2))

    return ET0
end

base_attrs = [
    "linear_solver" => "mumps",
    "ma97_scaling" => "mc64"
]
if !isempty(hsl_lib) && isfile(hsl_lib)
    push!(base_attrs, "hsllib" => hsl_lib)
end

optimizer = optimizer_with_attributes(Ipopt.Optimizer, base_attrs...)
model = Model(optimizer)

@variable(model, u[1:M] >= 0) # Control variable (irrigation)
@variable(model, x[1:M]) # State variable (soil moisture)

@expression(model, loss_expr[i=1:M-1],
    k * x[i] * (x[i] <= xfc) + (x[i+1] - xfc) * cs * (x[i] > xfc))

@objective(model, Min, h * sum(u[i] for i in 1:M))

@constraint(model, x[1] == xs)
@constraint(model, x .>= xmin)
@constraint(model, u .>= 0)

for i in 1:M-1
```

```
92          @constraint(model, x[i+1] == x[i] + h * ( (Kl * u[i] * (1 - Rf * sin(α))) - (Kc * evpt0
                (t[i])) + ((1 - Rf * sin(α)) * rainfall[i]) - loss_expr[i] ))
93      end
94
95      t0 = time()
96      optimize!(model)
97      tf = time()
98
99      X = round.(value.(x), digits=3)
100     U = round.(value.(u), digits=3)
101
102     println("Total water used: ", round(sum(U),digits=3))
103     println("Optimal control u: ", U)
104     println("Optimal state x: ", X)
105
106     using Plots, Interpolations
107
108     n_days = length(X)
109     hours_per_day = 24
110     total_hours = n_days * hours_per_day
111     hours = 1:total_hours
112
113     day_fraction = range(1, n_days, length=total_hours)
114     itp = LinearInterpolation(1:n_days, X)
115     X_hourly = itp.(day_fraction)
116     U_hourly = repeat(U, inner = hours_per_day)
117
118     plot(hours, X_hourly,
119        label = "Soil Moisture (X)",
120        xlabel = "Hour",
121        ylabel = "Soil Moisture (mm)",
122        legend = :outertop,
123        linewidth = 2,
124        linestyle = :dash,
125        color = :red,
126        xticks = 0:20:total_hours)
127
128     plot!(hours, U_hourly,
129        label = "Irrigation (U)",
130        line = (:step, 2, :blue),
131        ylabel = "Water (mm)")
132     savefig("moisture_irrigation.png")
133
134     println("\nTempo total: ", tf - t0, " s")
135     push!(tempos, tf-t0)
136 end
```

```
137 for (r,t) in zip(reps, tempos)
138     println("repTimes = $(r) -> $(round(t,digits=3)) s")
139 end
```

Appendix B

Algorithm for the Bilevel Problem

This appendix lists the Julia implementation of the bilevel irrigation optimisation model presented in Chapter 4, built using JuMP and BilevelJuMP, with IPOPT as the underlying solver. The upper level determines the minimum soil moisture threshold for each field, while the lower level determines the irrigation schedule required to satisfy it, subject to soil moisture dynamics, evapotranspiration, rainfall, and water availability constraints.

The code includes a loop that repeats the base ten-day weather and rainfall pattern a variable number of times, so that the model can be tested over increasingly long planning horizons within a single execution. This allows the solver's performance (runtime, iteration count, and convergence status) to be assessed as problem size increases, as discussed in Section 4.4.2.

Algorithm 2 Bilevel Hierarchical Allocation for Smart Field Irrigation

Input:

Time horizon M, Spatial fields N

Requirements: $\text{need}_{\mathbf{j}} \in \mathbb{R}^N$

Capacity: $C_{\text{total}} \in \mathbb{R}$

Initial moisture: $\mathbf{x_{S,j}} \in \mathbb{R}^N$

Profiles: $\text{rain}, \text{wind}, \text{temp} \in \mathbb{R}^M$

Physics: $\mathbf{K_I}, \mathbf{K_C}, \alpha_{\mathbf{j}}, R_F, x_{FC}, k$

Output:

Strategic matrix: $X_{\min} \in \mathbb{R}^{N \times M}$

Centralized scheduling matrix: $u \in \mathbb{R}^{M \times N}$

State trajectory matrix: $x \in \mathbb{R}^{M \times N}$

{Phase 1: Environmental Pre-processing}

for $i = 1$ **to** M **do**

  Calculate reference evapotranspiration $\text{ET}_{0,i}$ via FAO-56.

end for

{Phase 2: Monolithic Bilevel Optimization Setup}

Initialize Bilevel Container via `BilevelJuMP`.

Begin Upper-Level Problem (Central Coordinator):

Define strategic variables: $X_{\min,j,i}, \text{deviation}_{i,j}$

Set Upper Objective: Minimize aggregate baseline deviations (4.1)

Enforce Global Constraints:

- Shared delivery network capacity limits (4.2)
- Strategic domain bounds and slacks (4.3)–(4.4)

Begin Linked Lower-Level Problem (Central Scheduler):

Define: $u_{i,j}, x_{i,j}, \text{loss}_{i,j}, \text{overflow}_{i,j}$

Set Lower Objective: Minimize total collective water use

Enforce Joint Multi-Field Dynamics ($\forall j \in 1 \ldots N, \forall i \in 1 \ldots M$):

- Bind states to strategic matrix: $x_{i,j} \geq X_{\min,j,i}$
- Initialize baseline moisture states: $x_{1,j} = x_{S,j}$
- Evaluate mass-balance soil state transitions (4.9)
- Constrain deep percolation and saturation slacks

End Lower-Level Problem

End Upper-Level Problem

{Phase 3: Telemetry & Execution}

Set underlying solver to IPOPT and invoke optimization.

if Solver Status is Optimal **then**

  Extract unified decision matrices $X^*_{\min}$, u^*, x^*

  Compute macro water-saving and yield-stress metrics.

  return $X^*_{\min}, u^*, x^*$

else

  return `Optimization Failure Flag`

end if

Listing B.1: Main processing script

```
verbose = false

n_fields = 2 #number of fields
available = 3*n_fields #available water daily
need = repeat([18.72],n_fields) #daily water needed by each field

# --- Constants ---
xs = repeat([18.72],n_fields) # initial soil moisture (mm)
Kl = repeat([0.85],n_fields) # irrigation coefficient (0 to 1) (default value for sprinkler
    irrigation)
Kc = repeat([0.85],n_fields) # crop coefficient (0 to 1) (grass reference crop value)
Rf = repeat([0.5],n_fields) # runoff kept (0 to 1)
α= [30 * π/ 180, 0 * π/ 180, 0 * π/ 180] # soil inclination in deg (converted to rad)
cs = repeat([1],n_fields) # excess water lost (0 to 1)
xfc = 46.8 #soil maximum saturation (mm)
h = 1.0 # step between measurements (in days)
k = 0.05 # constant related to permeability of the soil used for loss calculation

time = Float64[]
iterations_list = Int[]
status_list = []
n_cores = 1
blas_cores = 1

reps = [1,5,10,15,20,25,30,35,40,45,50]

ENV["OMP_NUM_THREADS"] = n_cores
ENV["OPENBLAS_NUM_THREADS"] = blas_cores

using HSL_jll

using JuMP,BilevelJuMP,HiGHS,Ipopt,LinearAlgebra,Plots

function writeFile(i,status,iters,tf,t0,formatted_values,formatted_u,deviations,total_deficit
    ,n_fields)
   open("logs/results_$(i).txt", "w") do f
      println(f, "=== Repetition $(i) ===")
      println(f, "Status: $(status)")
      println(f, "Iterations: $(iters)")
      println(f, "Time: $(round(tf - t0, digits=3)) s")
      println(f, "\nX_min values")
      for fi in 1:n_fields
         print(f, "Field $fi: ")
         println(f, formatted_values)
      end
```

```
        println(f, "\nU values")
        for fi in 1:n_fields
            print(f, "Field $fi Irrigation: ")
            println(f, formatted_u)
        end
        println(f, "\nSlack values")

        for fi in 1:n_fields
            println(f, "Field $fi Deviations: $deviations")
            println(f, "Field $fi Total Deficit: $total_deficit mm")
            println(f, "-"^30)
        end

    end
end

for repTimes in reps

    # --- Data ---
    rainfall = repeat([10, 0, 0, 0, 0, 0, 0, 10, 0, 0],repTimes)
    windspeed = repeat([2.8, 3, 2.4, 2.4, 3, 2.2, 2.7, 2, 2.2, 2.8],repTimes)
    mintemp = repeat([17.4, 17.1, 15.2, 15.3, 15.1, 15.6, 16.3, 16.3, 15.4, 14.7],repTimes)
    maxtemp = repeat([24.4, 20.8, 23.1, 22.2, 24.9, 26.9, 29.3, 23.6, 19.4, 20.4],repTimes)

    function evpt0(i)

        Tmin = mintemp[i]
        Tmax = maxtemp[i]
        u2 = windspeed[i]
        Tmean = (Tmax + Tmin) / 2

        # Grass constants
        α2 = 0.23 # grass albedo
        λ= 2.45 # evaporation latent heat (MJ/kg) ##unused
        γ= 0.06665 # psicometric constant (kPa/°C)
        G = 0.0 # soil heat flow (MJ/m²/dia)
        kR = 0.16 # Hargreaves coefficient (0.19 if coastal)

        # Vapour pressure relative to temperature
        e0(T) = 0.6108 * exp((17.27 * T) / (T + 237.3))

        # Vapour pressure
        es = e0(Tmean)
        ea = e0(Tmin) # Assume Tdew ~= Tmin
        ediff = es - ea # Vapour deficit

```

```
    # Saturation Δ
    Δ= (4098 * (0.6108 * exp((17.27*Tmean) / (Tmean + 237.3)))) / ((Tmean + 237.3)^2)

    # Estimated solar radiation
    Ra = 28 # extraterrestrial radiation
    Rs = kR * sqrt(Tmax - Tmin) * Ra
    Rn = (1 - α2) * Rs # Liquid radiation

    # Simplified FAO-56 Penman-Monteith equation
    ET0 = (0.408 * Δ* (Rn - G) + γ* (900 / (Tmean + 273)) * u2 * ediff) /
        (Δ + γ* (1 + 0.34 * u2))

    if(verbose) print("Evapotranspiration is ",ET0,"\n") end

    return ET0
end

M = length(rainfall)

hsl_lib = get(ENV, "COINHSL_LIB", "")

base_attrs = [
    "linear_solver" => "mumps", #mumps, ma97
    "ma97_scaling" => "mc64",
    "max_iter" => 300000,
    "print_level" => 3,
    "acceptable_iter" => 15,
    "nlp_scaling_method" => "gradient-based",
    "mu_strategy" => "adaptive", #adaptive, monotone
    "bound_push" => 0.01, #1e-6, 0.01
    "bound_frac" => 0.01 #1e-6, 0.01
]

if !isempty(hsl_lib) && isfile(hsl_lib)
    push!(base_attrs, "hsllib" => hsl_lib)
end

optimizer = optimizer_with_attributes(Ipopt.Optimizer, base_attrs...)
model = BilevelModel(optimizer, mode = BilevelJuMP.ProductMode())

# UPPER LEVEL
# Variable: control variable (irrigation)
@variable(Upper(model), X_min[i=1:n_fields, t=1:M] >= 0)
for i in 1:n_fields, t in 1:M
    @constraint(Upper(model), X_min[i, t] <= need[i])
end
```

```
136    # Lower level decision variables
137    @variable(Lower(model), x[i=1:n_fields, t=1:M] >= 0) # Field humidity
138    @variable(Lower(model), u[i=1:n_fields, t=1:M] >= 0) # Irrigation
139
140    # xmin - x only if positive, otherwise = 0
141    @variable(Upper(model), deviation[i=1:n_fields, t=1:M] >= 0)
142    @constraint(Upper(model), [i=1:n_fields, t=1:M], deviation[i,t] >= need[i] - X_min[i,t])
143
144    # Objective: minimize deviation
145    # No UPPER LEVEL
146    ## NLobjective?
147    @objective(Upper(model), Min,
148       sum((deviation[i,t])^2 for i=1:n_fields, t=1:M)
149    )
150
151    # Don't use more water than available
152    @constraints(Upper(model), begin
153       [t = 1:M], sum(u[i,t] for i in 1:n_fields) <= available
154    end)
155
156    # LOWER LEVEL
157    for i in 1:n_fields, t in 1:M
158       @constraint(Lower(model), x[i,t] >= X_min[i,t]) #Constraint for water in field relative
              to upper level
159    end
160
161    #Loss expression defined as a variable
162    @variable(Lower(model), loss[i=1:n_fields, t=1:M-1] >= 0)
163
164    #Overflow defined as a variable
165    @variable(Lower(model), overflow[i=1:n_fields, t=1:M] >= 0)
166
167    for i in 1:n_fields
168       @constraint(Lower(model), x[i, 1] == xs[i]) #Defines starting soil moisture
169
170       for t in 1:M-1
171          @constraint(Lower(model),
172             x[i, t+1] == x[i, t] + h * (
173                (Kl[i] * u[i, t] * (1 - Rf[i] * sin(α[i]))) -
174                (Kc[i] * evpt0(t)) +
175                ((1 - Rf[i] * sin(α[i])) * rainfall[t]) -
176                loss[i, t] -
177                (cs[i] * overflow[i,t])
178             )
179          ) #Daily change in soil moisture
180
```

```
        @constraint(Lower(model), overflow[i, t] >= x[i+1, t] - xfc) #Overflow expression
        @constraint(Lower(model), loss[i, t] >= k * x[i, t]) #Loss expression
    end
    @constraint(Lower(model), overflow[i, M] >= x[i, M] - xfc)
end


@objective(Lower(model), Min, sum(u[i, t] for i in 1:n_fields, t in 1:M)) #Lower level
    problem objective

# Starting guesses
# Upper level: X_min initial guess
for i in 1:n_fields, t in 1:M
    set_start_value(X_min[i, t], need[i])
end
# Lower level: u initial guess starting guess
for i in 1:n_fields, t in 1:M
    set_start_value(u[i, t], 0)
end

println("Starting to iterate for $repTimes")
t0 = time()
pipe = Pipe()
Base.link_pipe!(pipe; reader_supports_async=true, writer_supports_async=true)
redirect_stdout(pipe.in) do
    optimize!(model)
end
close(pipe.in)
tf = time()
println("Finished iterating for $repTimes")

time_elapsed = tf-t0
println("Time taken: $time_elapsed")
push!(time,time_elapsed)

status = termination_status(model)
push!(status_list, status)
println("Status: ", status)

logtxt = String(read(pipe.out))
m = match(r"Number of Iterations\.\.\.\.:\s+(\d+)", logtxt)
iters = m === nothing ? -1 : parse(Int, m.captures[1])
close(pipe.out)

push!(iterations_list, iters)

```

```
226 if !(status in [MOI.OPTIMAL, MOI.LOCALLY_SOLVED])
227     open("logs/ipopt_log_$(repTimes).txt", "w") do f
228         write(f, logtxt)
229     end
230 end
231
232 if status == MOI.OPTIMAL || status == MOI.LOCALLY_SOLVED
233     # Extracts variable values
234     val_u = value.(u) # Irrigation matrix
235     val_x = value.(x) # Soil humidity matrix
236
237     for i in 1:n_fields
238         field_water = sum(val_u[i, :])
239         mean_wet = sum(val_x[i, :]) / M
240         if(verbose) println("Field $i: Water used = $(round(field_water, digits=2)) | Mean
                Soil Humidity = $(round(mean_wet, digits=2))") end
241     end
242 else
243     println("No optimal solution found.")
244 end
245
246 if termination_status(model) in [MOI.OPTIMAL, MOI.LOCALLY_SOLVED]
247
248     val_x = value.(x)
249     xmin_decided = value.(X_min)
250     val_u = value.(u)
251     days = 1:M
252
253     plots_fields = []
254
255     for i in 1:n_fields
256
257         # Soil humidity graph
258         p = plot(days, val_x[i, :],
259             lw=2, color=:blue, marker=:circle,
260             ylims=(0, 30), tickfontsize=20,
261             ylabel="Field $i", legend=false, label="", xlabel=("Days"))
262
263         # Rainfall as bars
264         bar!(p, days, rainfall,
265             label = "",
266             color=:gray, alpha=0.3)
267         # Best case xmin constant
268         hline!(p, [need[i]],
269             label = "",
270             color=:green, lw=2)
```

```
271         # Upper level X_min
272         plot!(p, days, xmin_decided[i, :],
273             label = "",
274             color=:orange, linestyle=:dash, lw=3)
275
276         # Irrigation control line
277         plot!(p, days, val_u[i, :],
278             label = "",
279             color=:cyan, marker=:square)
280
281         push!(plots_fields, p)
282     end
283
284     #Legend plot
285     p_legend = plot([0], [0], label="Soil moisture (x)", color=:blue, lw=2, marker=:circle,
286             grid=false, showaxis=false, ticks=false, frame=:none,
287             legendfontsize=14, legend=:outerbottom)
288     plot!(p_legend, [0], [0], label="Rainfall", color=:gray, seriestype=:shape, fillalpha
            =0.3)
289     plot!(p_legend, [0], [0], label="Expected moisture minimum", color=:green, lw=2)
290     plot!(p_legend, [0], [0], label="Temporary moisture minimum", color=:orange, linestyle
            =:dash, lw=3)
291     plot!(p_legend, [0], [0], label="Irrigation (u)", color=:cyan, marker=:square)
292
293     legend_layout = Plots.@layout [
294         grid(n_fields, 1)
295         l{0.2h}
296     ]
297
298     final_plot = plot(plots_fields..., p_legend, layout=legend_layout, size=(900, 600*
            n_fields+150), legend=:outerbottom, legendfontsize=20)
299     display(final_plot)
300
301     ##SAVE GRAPH
302     savefig("graphs/fields_$(repTimes).png")
303
304     formatted_values = []
305
306     ##PRINTS
307
308     if (verbose) println("\nX_min values") end
309     for i in 1:n_fields
310         if(verbose) print("Field $i: ") end
311         formatted_values = [round(xmin_decided[i, t], digits=2) for t in 1:M]
312         if(verbose) println(formatted_values) end
313     end
```

```
314       deviations = []
315       total_deficit = []
316
317       if(verbose) println("\nslack values") end
318       val_xmin = value.(X_min)
319       for i in 1:n_fields
320          # Calculate the difference for each time step
321          deviations = [round(need[i] - val_xmin[i, t], digits=2) for t in 1:M]
322
323          # Calculate total deficit for the field
324          total_deficit = sum(deviations)
325
326          if(verbose) println("Field $i Deviations: $deviations") end
327          if(verbose) println("Field $i Total Deficit: $total_deficit mm") end
328          if(verbose) println("-"^30) end
329       end
330
331       formatted_u = []
332
333       if(verbose) println("\nU values") end
334       for i in 1:n_fields
335          if(verbose) print("Field $i Irrigation: ") end
336          # Rounding for readability
337          formatted_u = [round(val_u[i, t], digits=2) for t in 1:M]
338          if(verbose) println(formatted_u) end
339       end
340
341       println("\ntime total: ", tf - t0, " s")
342
343       writeFile(repTimes,status,iters,tf,t0, formatted_values, formatted_u, deviations,
              total_deficit, n_fields)
344    else
345       println("Solver couldn't finish. ", termination_status(model))
346    end
347 end
348
349 for (idx, r) in enumerate(reps)
350    println("Repetition: $(r) | Status: $(status_list[idx]) | Iterations: $(iterations_list[
          idx]) | Time: $(round(time[idx], digits=3))s")
351 end
352 open("logs/resultados_final.txt", "w") do f
353    for (idx, r) in enumerate(reps)
354       println(f, "Repetition: $(r) | Status: $(status_list[idx]) | Iterations: $(
             iterations_list[idx]) | Time: $(round(time[idx], digits=3))s")
355    end
356 end
```