Applying Stochastic Optimisation to the New Zealand Dairy Industry

Oscar Dowson

Supervised by Prof. Andy Philpott,
Assoc. Prof Andrew Mason, and Dr. Anthony Downward.

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy in Engineering Science
at the University of Auckland, 2018.

Abstract

Pastoral dairy farmers continually make sequential decisions in the face of long-term environmental uncertainty and price volatility. Decisions made early in the season, such as the number of cows to farm per hectare, can have significant effects later in the season if, for example, the farmer is forced to import additional feed to meet the cows' energy demands during a drought.

This thesis analyses the problems faced by pastoral dairy farmers through the lens of multistage stochastic optimisation. The thesis is structured in two distinct parts that are approximately equal in content.

The first part of this thesis, *Multistage Stochastic Optimisation*, addresses the theory and computation of multistage stochastic optimisation. We focus on the stochastic dual dynamic programming algorithm as a solution technique for multistage stochastic optimisation models. There are three main chapters in Part I.

First, we conduct a literature review of the current state-of-the-art stochastic dual dynamic programming implementation. We also discuss our experience of a range of numerical issues related to stochastic dual dynamic programming. Second, we describe `SDDP.jl`, a software package we have developed for solving multistage stochastic optimisation problems using stochastic dual dynamic programming. Finally, we describe an extension to the stochastic dual dynamic programming algorithm to enable the solution of problems with stagewise-dependent objective uncertainty.

The second part of the thesis, *Applications in the Dairy Industry*, applies the theory and techniques developed in the first part of the thesis to the problems faced by pastoral dairy farmers in New Zealand.

We develop two models: MOO – the Milk Output Optimiser, and POWDER – the milk Production Optimiser incorporating Weather Dynamics and Economic Risk. We use these models to analyse the decision-making of a farmer in the Bay of Plenty region of New Zealand in a variety of circumstances. Finally, we introduce forward contracting for milk and show how a risk-averse farmer can use forward contracts to effectively manage their downside risk.

I never saw a Purple Cow,
I never hope to see one;
but I can tell you, anyhow,
I'd rather see than be one!

– Gelett Burgess

Acknowledgements

First, thank you to my supervisors: Andy Philpott, Andrew Mason, and Anthony Downward. To reprise a comment from my honours thesis: "their suggestions and feedback were invaluable, despite the fact that at times neither they, nor this author, understood what was being discussed."

Andy, thank you for the sushi, the week skiing in Italy, and the Wednesday night sailing. Andrew, thank you for the clear and concise sailing instructions, for returning my writing dripping in red ink, and for developing my affinity for Excel. Tony, thank you for the far too subtle sarcasm, for hosting our somewhat lengthy round-table discussions each Thursday afternoon, and for unravelling the mystery of dynamic price interpolation.

The Julia package outlined in Chapter 3, `SDDP.jl`, is the culmination of a decade of work at the Electric Power Optimisation Centre (EPOC) at the University of Auckland. Many people at EPOC have contributed to the development of SDDP and related codes over the years. These include Andy Philpott, Ziming Guan, Geoffrey Pritchard, Vitor de Matos, and Faisal Wahid. Credit also goes to Vincent Leclère, François Pacaud, Tristan Rigaut, Henri Gerard, Benoît Legat, and Joaquim Dias Garcia for their work in creating other implementations of the SDDP algorithm in Julia.

Special credit is due to Lea Kapelevich, who was the first adopter of `SDDP.jl`. In addition, she coded the model of the New Zealand Hydro-Thermal Scheduling Problem used in Chapter 3.

I also thank Miles Lubin, Iain Dunning, and Joey Huchette (the creators of JuMP), and the rest of the JuliaOpt contributors, for providing the foundation upon which to base my work. In particular, Miles, thank you for organising the first (annual) JuMP developers workshop and for leading the charge towards JuMP1.0.

To the many friends who, over the last four years, put up with my incessant discussion of cows, thank you. Emily, I'm sorry to inform you that if this is published, I will have won our competition to see who could add Dr. to their name first. B-Meister, McDane, C-Dawg, and Frodo, thank you for the support, the memorable nights out, and for a tramping trip that introduced me to Små Grå. Henri, my doppelganger, thank you for adding to my DUPLO farm and for being a wonderful host in Paris. Philip, thank you for providing me with some Austrian folk music; I almost know the words. Sibylle, thank you for the constant (declined) invitations to dancing, and for sharing your parents' wine. Mats, thank you for never tiring of a constant rotation of Korean pancakes, vegan lunch, and sushi. Bob, thank you for the football match in

Kaiserslautern and for making me try a sport that involves full-contact wrestling on the bottom of a 5m deep pool. Finally, Georgina, thank you for teaching me the difference between through and thorough.

To my parents, Mary and Nick, thank you for pretending to understand what I've been doing all these years. I hope that part of this thesis is at least somewhat comprehensible. Nick, in addition to teaching me how to milk a cow (Figure 1), provided the data used for the case studies in Part II.

Figure 1: The author, aged 7.

Lastly, Rebecca Astwood deserves a very special thank you for inadvertently setting me off on this journey a little over 10 years ago.

Oscar Dowson
Auckland, May 2018

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

School of Graduate Studies
AskAuckland Central
Alfred Nathan House
The University of Auckland
Tel: +64 9 373 7599 ext 81321
Email: postgradinfo@auckland.ac.nz

Co-Authorship Form

This form is to accompany the submission of any PhD that contains published or unpublished co-authored work. **Please include one copy of this form for each co-authored work**. Completed forms should be included in all copies of your thesis submitted for examination and library deposit (including digital deposit), following your thesis Acknowledgements. Co-authored works may be included in a thesis if the candidate has written all or the majority of the text and had their contribution confirmed by all co-authors as not less than 65%.

Please indicate the chapter/section/pages of this thesis that are extracted from a co-authored work and give the title and publication details or details of submission of the co-authored work. Chapter 3 is based on O Dowson and L. Kapelevich. SDDP.jl: a Julia package for Stochastic Dual Dynamic Programming. Optimization Online, 2017. URL: http://www.optimization-online.org/DB_HTML/2017/12/6388.html	
Nature of contribution by PhD candidate	All of the chapter except the development of the NZ HTSP Model.
Extent of contribution by PhD candidate (%)	90

CO-AUTHORS

Name	Nature of Contribution
Lea Kapelevich	Wrote the SDDP.jl implementation of the NZ HTSP model in Section 3.4, performed the experiments.

Certification by Co-Authors

The undersigned hereby certify that:

- the above statement correctly reflects the nature and extent of the PhD candidate's contribution to this work, and the nature of the contribution of each of the co-authors; and
- that the candidate wrote all or the majority of the text.

Name	Signature	Date
Lea Kapelevich	Eap	24 May 2018

Last updated: 28 November 2017

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

School of Graduate Studies
AskAuckland Central
Alfred Nathan House
The University of Auckland
Tel: +64 9 373 7599 ext 81321
Email: postgradinfo@auckland.ac.nz

Co-Authorship Form

This form is to accompany the submission of any PhD that contains published or unpublished co-authored work. **Please include one copy of this form for each co-authored work**. Completed forms should be included in all copies of your thesis submitted for examination and library deposit (including digital deposit), following your thesis Acknowledgements. Co-authored works may be included in a thesis if the candidate has written all or the majority of the text and had their contribution confirmed by all co-authors as not less than 65%.

Please indicate the chapter/section/pages of this thesis that are extracted from a co-authored work and give the title and publication details or details of submission of the co-authored work. Chapter 8 is based on O. Dowson, A. Philpott, A. Mason, and A. Downward. A multi-stage stochastic optimization model of a pastoral dairy farm. Submitted to EJOR.

Nature of contribution by PhD candidate	Developed the model, performed the experiments, wrote the manuscript.
Extent of contribution by PhD candidate (%)	80

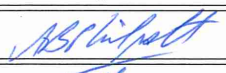
CO-AUTHORS

Name	Nature of Contribution
Andy Philpott	Discussions, advice, and editing.
Andrew Mason	Discussions, advice, and editing.
Anthony Downward	Discussions, advice, and editing.

Certification by Co-Authors

The undersigned hereby certify that:

- the above statement correctly reflects the nature and extent of the PhD candidate's contribution to this work, and the nature of the contribution of each of the co-authors; and
- that the candidate wrote all or the majority of the text.

Name	Signature	Date
Andy Philpott		25/5/18
Andrew Mason		25/5/18.
Anthony Downward		25/5/18

Last updated: 28 November 2017

Contents

Preface

This thesis is about decision-making under uncertainty. More specifically, it is about the problem of making a sequence of decisions in time when the future is unknown. These *multistage stochastic optimisation* problems arise in a variety of settings, many of which will be familiar to the reader. In this thesis, we explore examples in which factories produce products without knowing exact future demand; hydroelectric generators manage reservoir levels without knowing future inflows; and dairy farmers manage their herd without knowing future grass growth.

For several reasons, including the high-dimensionality of real-world problems and the necessity of making decisions before we know what information will arrive in the future, multistage stochastic optimisation problems are difficult to solve [164]. Even the problem of comparing two solutions is difficult. Should we compare the average or the worst-case outcome?

However, despite the difficulty of solving these problems to optimality, human decision makers can, and do, make good decisions based on years of experience and intuition. The dairy farmer that we meet in Part II of this thesis continually makes sequential decisions in the face of short- and long-term environmental uncertainty and price volatility. As we shall see, independently of our model, the farmer reduced the number of cows on their farm in line with our optimisation model's recommendation. To quote Kellogg [117], writing about oil drilling companies:

> Given the small size of many ... firms ..., it seems unlikely that they are formally solving Bellman equations. However, they may have developed decision heuristics that roughly mimic an optimal decision-making process. Moreover, the firms have a strong financial incentive to get their decision-making at least approximately right.

Although the farmer's decision-making was approximately right regarding the question of reducing the number of cows, the farmer is not infallible. The pioneers of behavioural psychology, Tversky and Kahneman, wrote that:

> Most important decisions [people] make are governed by beliefs concerning the likelihood of unique events. The "true" probabilities of such events are elusive, since they cannot be assessed objectively. The subjective probabilities that are assigned to unique events by knowledgeable and consistent people have been accepted as all that can be said about the likelihood of such events.

> Although the "true" probability of a unique event is unknowable, the reliance on heuristics such as availability or representativeness, biases subjective probabilities in knowable ways. (Tversky and Kahneman [196])

Our farmer is a good demonstration of this biased subjectivity. The majority (84%) of dairy farmers in New Zealand supply milk to Fonterra, a large milk processing co-operative [195]. However, they are not paid for their milk on delivery. Instead, at the end of each season (one year), they are back-paid an *end-of-season* milk price for each kilogram of milk supplied during the preceding season. This end-of-season milk price is calculated from a series of international auctions that occur during the year. Therefore, at the start of the season, there is some uncertainty surrounding the end-of-season milk price, since the auctions have not yet occurred. To help guide the farmers in their decision-making, Fonterra releases a sequence of *forecast* milk prices during the season. As the season progresses, this sequence of forecast milk prices converges towards the end-of-season milk price. A question naturally arises regarding the accuracy of the forecast milk price at the start of the season, compared with the end-of-season milk price. In Table 1, we give the initial forecast milk price issued by Fonterra, the end-of-season milk price, and the difference, for the last eight complete seasons in New Zealand.

Season	09/10	10/11	11/12	12/13	13/14	14/15	15/16	16/17
Initial	4.10	6.60	6.75	5.50	7.00	7.00	5.25	4.25
End-of-Season	6.10	7.60	6.08	5.84	8.40	4.40	3.90	6.12
Difference	$+2.00$	$+1.00$	-0.67	$+0.34$	$+1.40$	-2.60	-1.35	$+1.87$

Table 1: Comparison of Fonterra's initial milk price forecast against the end-of-season milk price for the last eight complete seasons. All units are NZ$/kg. Note that a season runs from June to the following May, so the seasons overlap across years (e.g. 09/10 refers to the 2009/10 season). Data sourced from Fonterra [75].

In an interview, we asked the farmer if the end-of-season milk price was more likely, or less likely, to be greater than the initial forecast milk price. He replied that the end-of-season milk price was more likely to be greater than the initial forecast milk price because "Fonterra want to give farmers good news during the year." Looking at the historical record over the last eight seasons (i.e. 2009/10 – 2016/17), there is a 5–3 split in favour of seasons in which the end-of-season milk price was greater than the initial forecast milk price. However, there is no evidence that this result is statistically significant (p=0.52). Therefore, despite the intention of Fonterra to release a sequence of forecast milk prices to help guide farmers' decision-making, biases on the part of the individual farmers may lead them to misinterpret the available information and make sub-optimal decisions.

Moreover, the decision by our farmer to reduce the number of cows was only made after several years of trial and error. This lag in decision-making, where the human must learn from experience, provides an opportunity for optimisation and modelling to improve their decision-making.

In our view, multistage stochastic optimisation offers a structured way to remove the bias of the decision maker (although the model itself may introduce alternative biases). It also tightens the feedback loop between action and experience, and it allows the decision maker to be proactive rather than reactive. Dantzig, one of the fathers of mathematical programming, remarked in 1991 that (our emphasis added):

> In retrospect it is interesting to note that the original problem that started my research is still outstanding – namely the problem of planning or scheduling dynamically over time, particularly *planning dynamically under uncertainty*. If such a problem could be successfully solved it could eventually through better planning contribute to the well-being and stability of the world. (Dantzig [53])

This thesis is structured in two distinct parts that are approximately equal in content. The first, *Multistage Stochastic Optimisation*, addresses the theory and computation of multistage stochastic optimisation. The second, *Applications in the Dairy Industry*, applies the theory and techniques developed in the first part of the thesis to the problems faced by pastoral dairy farmers in New Zealand. It is intended that each part is more or less self-contained, although readers unfamiliar with stochastic optimisation will benefit from reading Part I before beginning Part II. Within each part, the chapters should be read sequentially. Since the two parts are distinct, they each begin with an introductory chapter. For example, an introduction and literature review of the dairy industry can be found in Chapter 6 instead of in this chapter. In addition, each part finishes with a conclusion chapter that summarises the main results discussed in that part.

Thesis outline

The thesis is structured as follows.

Part I Chapters 1 – 5 form Part I of this thesis. In Chapter 1, we introduce the reader to multistage stochastic optimisation and outline the notation and vocabulary used in this thesis. The main contribution is the idea of a *policy graph* as a structured way of formulating a class of multistage stochastic optimisation problems. One way of solving such problems is the *stochastic dual dynamic programming* (SDDP) algorithm, which is the focus of Chapter 2. In that chapter, we describe the algorithm and survey the literature regarding the current state-of-the-art implementations. In Chapter 3, we introduce `SDDP.jl`, a Julia library we have developed to solve multistage stochastic optimisation problems using SDDP. This library forms the main contribution of Part I of this thesis. In Chapter 4, we introduce and analyse an extension to the SDDP algorithm to solve a specific class of multistage stochastic optimisation problems. Finally, in Chapter 5, we conclude with a summary of the main results and contributions of Part I and some avenues of future research.

Part II The next five chapters, Chapters 6 – 10, form Part II of this thesis. Chapter 6 is a short chapter that contains an introduction to the New Zealand dairy industry and a review of multistage stochastic optimisation in agriculture. The next three chapters all concern the formulation and application of multistage stochastic optimisation models to the New Zealand dairy industry. Since these chapters require some understanding of the New Zealand dairy industry, we delay our introduction of them to the end of Chapter 6. We conclude Part II in Chapter 10 with a summary of the main results and contributions of Part II and some avenues of future research.

We conclude the thesis with a short summary that integrates both parts.

Part I

Multistage Stochastic Optimisation

Chapter 1

Introduction

The following chapters in Part I of this thesis deal with the theory and computation of multistage stochastic optimisation. Optimisation under uncertainty has a long history dating back to Dantzig [54]. However, unlike the standardisation that has taken place in deterministic optimisation (where terms like decision variable, constraint, and objective function are widely accepted), the stochastic optimisation community has fragmented into different communities, each of which speaks a different arcane language [165]. Therefore, it is necessary to clearly define the terminology and notation that we shall be using in this thesis. Readers should be aware that compared with other approaches in the literature, there are some subtle (and some large) differences in our approach. We use the language of stochastic optimal control, with terms like *stage*, *state*, and *control*. We do not view the world as a scenario tree, nor do we explicitly express the uncertainty in terms of a filtration on some probability space. Instead, we follow an approach that is heavily inspired by Powell [165]: we decompose the problem into small atomic pieces, clearly define each piece, and then define how the pieces fit together. We begin with some terminology.

1.1 Terminology

First, let us define the *stage* in multistage.

Definition 1. A *stage* is a discrete interval of time in which the agent chooses a decision and any uncertainty is revealed.

Therefore, *multistage* refers to a problem that can be decomposed into a sequence of stages. This requires the assumption that time can be discretised. Second, the *stochastic* component of multistage stochastic optimisation refers to problems with uncertainty. In this thesis, we differentiate between two types of uncertainty; the first of which we term a *noise*. (We shall describe the second type in the next section, but it relates to how the problem transitions between stages.)

Definition 2. A *noise* is a stagewise-independent random variable in stage t.

In stage t, we denote a single observation of the noise with the lowercase ω_t, and the sample space from which it is drawn by the uppercase Ω_t. Ω_t can be continuous or discrete, although in this thesis we only consider the discrete case. Furthermore, we use the term *stagewise-independent* to refer to the fact that the noise in stage t is sampled independently of the noise terms in stages $1, 2, \ldots, t-1, t+1, t+2, \ldots, T$.

Next we define a *state*, taking the definition from Powell [166].

Definition 3. A *state* is the minimally dimensioned function of history that captures all the information we need to model a system from some point in time onward.

Expressed a different way, a state is the smallest piece of information that is necessary to pass between stage t and $t+1$ so that the optimal decision-making in stage $t+1$ onward can be made independently from the decisions that were made in stages 1 to t. Each dimension of the state is represented by a *state variable*. State variables can be continuous or discrete.

We denote the state variable at the start of stage t by the lowercase x_t. We refer to x_t as the *incoming state variable*. Then, during the stage: the agent chooses a control (action); a realisation of the noise is observed; and the state transitions to x'_t at the end of the stage according to the transition function. We refer to x'_t as the *outgoing state variable*. We now define a *control* and the *transition function*.

Definition 4. A *control* variable is an action or decision taken (explicitly or implicitly) by the agent during a stage.

Control variables in stage t can be continuous or discrete. In this thesis, we denote control variables with the lowercase u_t. Such controls must be feasible for the agent to implement, and therefore they belong to a set that depends on the incoming state variable and observation of the random noise; this is denoted $u_t \in U_t(x_t, \omega_t)$.

Definition 5. The *transition function* is a mapping of the incoming state x_t to the outgoing state x'_t, given the control u_t and the noise ω_t.

We denote the transition function as $x'_t = T_t(x_t, u_t, \omega_t)$. This function can be of any form. As a result of the state transitioning, a cost is incurred.

Definition 6. The *stage-objective* is the cost (if minimising, otherwise value) accrued in stage t as a consequence of taking the control u_t, given the incoming state x_t and realisation of the noise ω_t.

This is denoted $C_t(x_t, u_t, \omega_t)$. All that remains is to define how the agent chooses a control. We use the terminology of Puterman [169] and call this a *decision-rule*.

Definition 7. A *decision-rule*, π_t, for stage t, is a mapping of the incoming state variable x_t and observation of the noise ω_t, to a control u_t.

This is denoted $u_t = \pi_t(x_t, \omega_t)$. In cases where the decision-rule does not depend upon the noise, we denote the decision-rule as $\pi_t(x_t)$. We refer to a set of decision-rules, one for each stage t, as a *policy*.

Definition 8. A *policy* is a set of decision-rules $\pi = \{\pi_t : \ t = 1, 2, \ldots, T\}$, containing one element for each stage t.

Now that we have defined some basic terminology, we can construct the atomic building block of multistage stochastic optimisation, which we refer to as a *node*.

Definition 9. A *node* is a collection of the following components: incoming and outgoing state variables, a noise, some control variables, a transition function, a stage-objective, and a decision-rule.

In this thesis, we discriminate between two types of node:

(1) A *Hazard-Decision* node – also called Wait-and-See [30]; and
(2) A *Decision-Hazard* node – also called Here-and-Now [30].

We now define each type of node in turn.

Definition 10. In a *Hazard-Decision* node, the agent chooses a control u_t *after* observing a realisation of the noise $\omega_t \in \Omega_t$ according to the decision-rule $\pi_t(x_t, \omega_t)$. The state transitions from x_t to x'_t according to the transition function $T_t(x_t, u_t, \omega_t)$. The decision-rule respects the set of admissible controls so that $u_t \in U_t(x_t, \omega_t)$. In addition, a cost $C_t(x_t, u_t, \omega_t)$ is incurred. A schematic of this is shown in Figure 1.1.

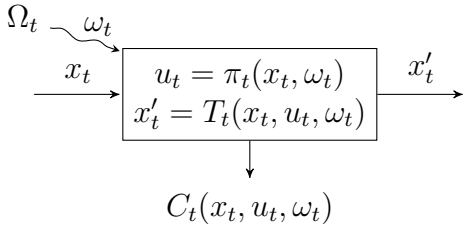


Figure 1.1: Schematic of a Hazard-Decision node.

Definition 11. In a *Decision-Hazard* node, the agent chooses a control u_t *before* observing a realisation of the noise $\omega_t \in \Omega_t$ according to the decision-rule $\pi_t(x_t)$. The state transitions from x_t to x'_t according to the transition function $T_t(x_t, u_t, \omega_t)$. The decision-rule respects the set of admissible controls so that $u_t \in U_t(x_t)$. In addition, a cost $C_t(x_t, u_t, \omega_t)$ is incurred. A schematic of this is shown in Figure 1.2.

When no uncertainty is realised in the node (i.e. $|\Omega_t| = 1$), the Decision-Hazard node becomes identical to the Hazard-Decision node. We denote this deterministic node by dropping the wavy line coming into the box depicting the node and dropping the ω_t function arguments.

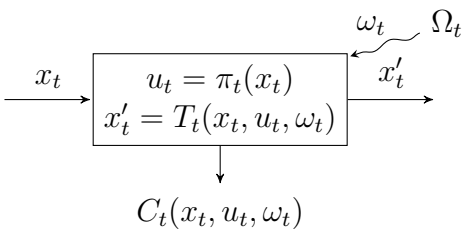


Figure 1.2: Schematic of a Decision-Hazard node.

Furthermore, readers should note that the two types of nodes are not immutable. Instead, they are a modelling choice. For example, it is possible to transform a Decision-Hazard node into a deterministic node followed by a Hazard-Decision node with an expanded state-space. An example of this is shown in Figure 1.3. Note that we pass x_t and u_t between the nodes, instead of just x_t.

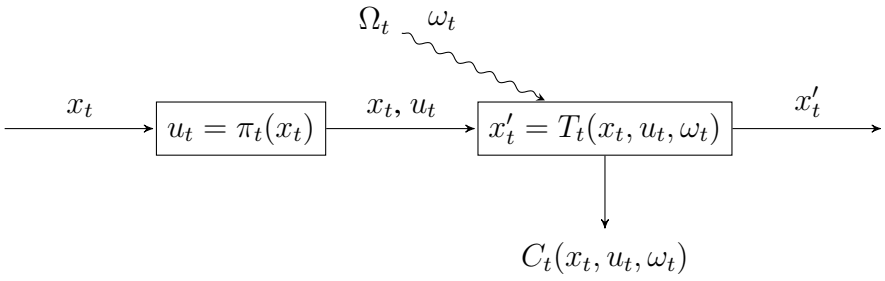


Figure 1.3: A Decision-Hazard node expanded into a deterministic node and a Hazard-Decision node.

The definitions above should be familiar to most readers versed in stochastic optimisation. However, notably excluded is a description of *how* the nodes are linked together. Typically, nodes are linked together as a linear sequence so that $x'_t = x_{t+1}$. In this case, we use the terms *node* and *stage* interchangeably, and we use x_t and x_{t+1} instead of x_t and x'_t. Implicit within this formulation is the idea that u_t cannot depend upon events in stages $t+1$ onward (i.e. nonanticipativity), since the decision-rule π_t depends only upon the incoming state variable x_t and the stagewise independent random variable ω_t.

In contrast to the linear case, it is possible to link nodes together in more complicated structures (e.g. such as the Markovian lattice used by Philpott and de Matos [160]). In this case, there can be many nodes corresponding to each stage (in time). Therefore, we cannot use the terms *node* and *stage* interchangeably. To clearly signal the difference, we will use the subscript i instead of t when referring to the components within each node. In the next section, we generalise this idea of linking nodes with what we term a *policy graph*.

1.2 Policy graphs

In the previous section, we defined two types of node: Hazard-Decision and Decision-Hazard. In this section, we describe how to link nodes together. Thus, we first need to define the initial conditions of the problem.

Definition 12. The *root node* is the current point-of-view of the agent in the decision-making process and stores an initial value x_R for the state variable.

It is important to note that the root node is neither a Hazard-Decision nor a Decision-Hazard node; it is just a convenient object to represent the initial point in the sequential decision-making process.

We now have all the terminology necessary to define the object that specifies the structure of how the nodes link together. We call the structure a *policy graph*.

Definition 13. A *policy graph* $\mathcal{G} = (R, \mathcal{N}, \mathcal{E}, \Phi)$ is defined by a tuple containing the root node R, along with the set of nodes $\mathcal{N}$ and directed edges $\mathcal{E}$. Φ is an $|\mathcal{N} + 1|$ by $|\mathcal{N}|$ matrix of the transition probabilities between nodes, with entries $\phi_{i,j}$, such that if there exists an edge (i, j) in $\mathcal{E}$ for two nodes $i \in \mathcal{N} \cup \{R\}$ and $j \in \mathcal{N}$, then $x_j = x_i'$ with probability $\phi_{i,j} \geq 0$. If no edge exists, then $\phi_{i,j} = 0$.

This definition encompasses the second type of uncertainty alluded to in the previous section: the probability of transitioning between nodes in the policy graph. (Recall the first type of uncertainty was the stagewise-independent *noise* within a node.) Before we elaborate on the transitions between nodes, it is useful to define the *children* of a node and the notion of two nodes being *connected*.

Definition 14. The *children* of node i is the set $i^+ = \{j : \phi_{i,j} > 0\}$.

Definition 15. Node i is *connected* to node j if $j \in i^+$.

In the remainder of this section we show, by graphical example, the various features of a policy graph. The graphical representation of each policy graph has a single circle that represents the root node. The square boxes represent Hazard-Decision or Decision-Hazard nodes. For simplicity, we drop the annotations used in Figures 1.1 and 1.2, since the node type can be inferred based on the presence (or absence) of a wavy arc representing the noise.

We note that this chapter is not the first to advocate for a graphical representation of the decision-making process. The System Dynamics community have developed a rich graphical framework for conveying dynamical systems [192]. However, the System Dynamics approach focuses on the (causal) relationships between the states (stocks) and controls (flows) and cannot express how uncertainty is revealed through the decision-making process. There are also well-known tools in the Operations Research community that use similar symbols such as decision trees, scenario trees, and flowcharts. (See Powell [164] for examples.) Our approach is

different in that it operates at a level of abstraction above the System Dynamics framework (for example, each node can be diagrammed using System Dynamics), and that (as we shall show) it supersedes the scenario tree approach.

Sequential Hazard-Decision problem In a sequential Hazard-Decision problem, the noise is observed at each square node before the control is chosen. A graphical representation of the policy graph is given in Figure 1.4. Note that the arcs can be interpreted as the flow of information (i.e. the states) between nodes.

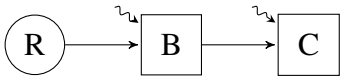


Figure 1.4: Two-stage Hazard-Decision problem.

The node graph is:

$$\mathcal{G} = (R, \mathcal{N}, \mathcal{E}, \Phi) = (R, \{B, C\}, \{(R, B), (B, C)\}, \{\phi_{R,B} = 1, \phi_{B,C} = 1\}) .$$

We call a policy graph of this form a *linear policy graph*.

Definition 16. A *linear policy graph* is composed of a finite set of nodes, where each node is connected to, at most, one other node.

Recall that when the policy graph is linear, we can use the terms *node* and *stage* interchangeably, and we denote the outgoing state variable in stage t as x_{t+1} instead of x'_t.

A policy graph can also be thought of as a compressed scenario tree. If each Hazard-Decision node in Figure 1.4 has two possible realisations for ω_t, then the scenario tree has six nodes. This can be described by the policy graph:

$$\begin{aligned}\mathcal{G} = \quad & (R, \mathcal{N}, \mathcal{E}, \Phi) = (\\ & R, \\ & \{B, C, D, E, F, G\}, \\ & \{(R, B), (R, C), (B, D), (B, E), (C, F), (C, G)\}, \\ & \{\phi_{R,B} = 0.5, \phi_{R,C} = 0.5, \phi_{B,D} = 0.5, \phi_{B,E} = 0.5, \phi_{C,F} = 0.5, \phi_{C,G} = 0.5\} \\ ).\end{aligned}$$

The graphical representation is shown in Figure 1.5.

The policy graphs in Figures 1.4 and 1.5 are equivalent representations of the same problem. However, the first example contains more information than the second example since it shows that node D is identical to node F, and node E is identical to node G. To clarify the point, any policy graph (with a finite sample space for the noise Ω) can be expanded into a scenario tree, but a scenario tree cannot be compressed into a linear policy graph without the additional information of which transitions between nodes are stagewise-independent.

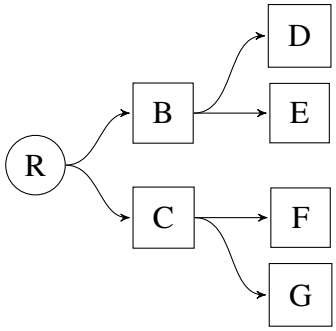


Figure 1.5: Explicit scenario tree representation.

The benefit of the graphical representation of a policy graph is most apparent when different types of nodes are mixed together. For example, a common problem in the literature is the two-stage problem with recourse (Figure 1.6). In this problem, the first stage is deterministic, and the second stage is Hazard-Decision.

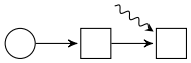


Figure 1.6: Two-stage problem with recourse.

Conditional dependence

So far, we have limited the uncertainty to two types: (1) a stagewise-independent *noise* within a node; and (2) probabilistic transitions between nodes. Importantly, the transition between the nodes is sampled first, and this determines the sample space from which the stagewise-independent noise is sampled. We now show how these two types of uncertainty can be combined to produce stagewise-dependent[1] noise. As a motivating example, consider a hypothetical ice-cream seller in New Zealand. The New Zealand summer climate is dominated by the El Niño-Southern Oscillation, an irregular, periodical climate pattern in the Pacific Ocean. It has two extremes: El Niño and La Niña. In El Niño years, the Eastern Pacific warms relative to average, and there is less rainfall than average in New Zealand. In La Niña years, the opposite is true [147]. Prior to learning if the year is El Niño or La Niña, an ice-cream seller must order a quantity of ice-cream to sell over the summer. Then, during the summer, their sales depend upon the number of sunny days. The number of sunny days is uncertain, although there is a higher probability of sunny days during El Niño years than La Niña.

We could model this problem by a linear, two-stage policy graph like Figure 1.6. However, we would need to add an additional binary state variable to code whether the system was in El Niño or La Niña. This formulation comprises one decision-rule for the first stage and one decision-rule for the second stage.

Alternatively, we can model the problem as shown in Figure 1.7 without the need for the additional state variable. This formulation comprises one decision-rule for the first stage (as

[1] We really mean *nodewise-dependent*, but this may confuse the matter too much.

before), but *two* decision-rules for the second stage: one to use if the year is El Niño and one to use if the year is La Niña.

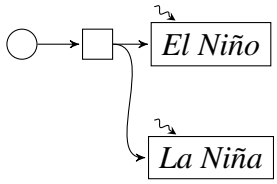


Figure 1.7: Two-stage problem with recourse and conditional dependence.

Importantly, the realisation of the random node transition (into either El Niño or La Niña) is exogenous to the remainder of the problem. Such a decomposition is hardly novel – indeed it is identical to the process of decomposing the problem into stages (and thereby solving for a decision-rule in each stage, instead of a single decision-rule that includes time as a state variable).

If we extend this structure to a series of stages, where all the nodes in a stage t are connected to all the nodes in the next stage $t+1$, we end up with a Markov chain structure. We call policy graphs with this structure *Markovian*.

Definition 17. A *Markovian policy graph* is composed of a finite set of stages, where each stage consists of a set of nodes, and all the nodes within a stage are connected to nodes within, at most, one other stage.

An example of a Markovian policy graph is shown in Figure 1.8. The recombination of the nodes prevents the number of leaf nodes in the expanded scenario tree from growing exponentially. We refer to the different nodes within a stage as the *Markov states* of the stage. Note that a linear policy graph is a Markovian policy graph with one Markov state in each stage.

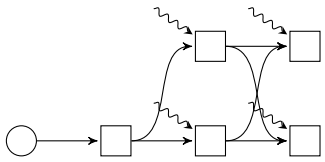


Figure 1.8: A Markovian policy graph.

The scenario tree in Figure 1.5 is also a Markovian policy graph. However, there are some structures which are neither linear nor Markovian; an example of which is shown in Figure 1.9. In that example, the policy graph is not Markovian as the node in the first stage has one child in the second stage and one in the third stage.

To convert the policy graph in Figure 1.9 into a Markovian policy graph, we could introduce a new Markov state in the second stage on the lower branch. The new Markov state would be deterministic, have no control variables, and have a stage-objective $C_i(x_i) = 0$. The transition function would map the incoming state variables directly to the outgoing state variables so that $x_i = x_i'$.

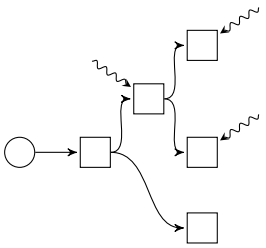


Figure 1.9: A non-Markovian policy graph.

Non-Markovian policy graphs demonstrate the need to distinguish between a node and a stage. However, once this distinction is made, we can easily express problems with random transitions between stages (i.e. if time is not sequential) such as those considered by Guigues [95]. An example is shown in Figure 1.10.

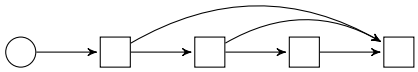


Figure 1.10: A policy graph with random transitions between stages.

We are almost ready to describe the optimisation problem of finding the optimal decision-rule for each node. However, before we proceed, it is necessary to introduce the concept of *risk*.

1.3 Risk

At every node in the policy graph of a multistage stochastic optimisation problem, the agent needs some way of aggregating the future cost of a control across the future uncertainty. They do so via a *risk measure*.

Definition 18. A *risk measure* $\mathbb{F}$ is a function that maps a random variable to a real number.

To elaborate upon this definition, we draw heavily from Shapiro et al. [188, Ch. 6.3]. In this thesis, we restrict our attention to random variables with a finite sample space $\Omega := \{z_1, z_2, \ldots, z_K\}$ equipped with a sigma algebra of all subsets of Ω and respective (strictly positive) probabilities $\{p_1, p_2, \ldots, p_K\}$. This greatly simplifies the analysis of risk and is a required assumption for our proposed solution technique (stochastic dual dynamic programming, which we discuss in the next chapter). We denote the random variable with the uppercase Z.

In this thesis, we shall use the following risk measures: Expectation ($\mathbb{E}[Z]$), Average Value-at-Risk (AV@R$_{1-\beta}[Z]$), and Worst-case ($\max[Z]$). We visualise these risk measures in Figure 1.11 given the probability density function of Z. We assume that we are minimising; thus, smaller values of Z are better.

The Expectation and Worst-case risk measures are self-explanatory. However, the Average Value-at-Risk is worth explaining for readers unfamiliar with it.

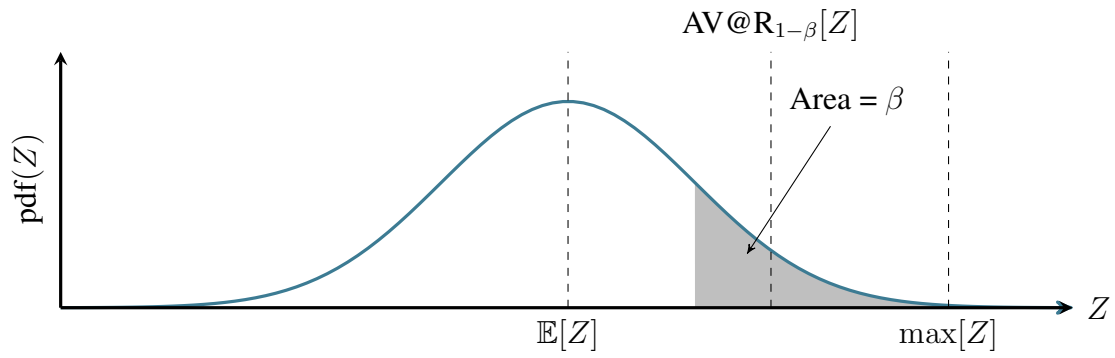


Figure 1.11: Graphical representation of the risk measures used in this thesis.

Definition 19. According to Rockafellar and Uryasev [179], the *Average Value-at-Risk*[2] at the β quantile (AV@R$_{1-\beta}$) is:

$$\text{AV@R}_{1-\beta}[Z] = \inf_{\zeta} \left\{ \zeta + \frac{1}{\beta} \sum_{k=1}^{K} p_k (z_k - \zeta)_+ \right\},$$

where $(x)_+ = \max\{0, x\}$.

As a simple approximation, the AV@R$_{1-\beta}$ can be thought of as the expectation of the worst β fraction of outcomes. However, if the distribution of the random variable is not continuous (e.g. the distribution is discrete), the interpretation is subtler since we may have to split a discrete probability atom. (See Rockafellar and Uryasev [179] for more details.) Also note that when $\beta = 1$, AV@R$_{1-\beta}[Z] = \mathbb{E}[Z]$, and $\lim_{\beta \to 0}$AV@R$_{1-\beta}[Z] = \max[Z]$.

We use these risk measures (Expectation, AV@R, and Worst-case) because they are *coherent* according to the following axioms.

Definition 20. A *coherent* risk measure is a risk measure $\mathbb{F}$ that satisfies the axioms of Artzner et al. [6]. For two discrete random variables Z_1 and Z_2, each with drawn from a sample space with K elements, the axioms are:

- **Monotonicity**: If $Z_1 \leq Z_2$, then $\mathbb{F}[Z_1] \leq \mathbb{F}[Z_2]$.
- **Sub-additivity**: For Z_1, Z_2, then $\mathbb{F}[Z_1 + Z_2] \leq \mathbb{F}[Z_1] + \mathbb{F}[Z_2]$.
- **Positive homogeneity**: If $\lambda \geq 0$ then $\mathbb{F}[\lambda Z] = \lambda\, \mathbb{F}[Z]$.
- **Translation equivariance**: If $a \in \mathbb{R}$ then $\mathbb{F}[Z + a] = \mathbb{F}[Z] + a$.

Positive homogeneity and sub-additivity give:

- **Convexity**: For $\lambda \in [0, 1]$, $\mathbb{F}[\lambda Z_1 + (1 - \lambda) Z_2] \leq \lambda\, \mathbb{F}[Z_1] + (1 - \lambda)\, \mathbb{F}[Z_2]$.

[2] Rockafellar and Uryasev [179] actually call this the *Conditional Value-at-Risk* (CV@R); however, we follow Shapiro et al. [188] and refer to it as AV@R.

We can also define coherent risk measures in terms of *risk sets* [6, 188]. That is, a coherent risk measure $\mathbb{F}$ has a dual representation that can be viewed as taking the expectation of the random variable with respect to the worst probability distribution within some set $\mathfrak{A}$ of possible distributions:

$$\mathbb{F}[Z] = \sup_{\xi \in \mathfrak{A}} \mathbb{E}_{\xi}[Z] = \sup_{\xi \in \mathfrak{A}} \sum_{k=1}^{K} \xi_k z_k, \tag{1.1}$$

where $\mathfrak{A}$ is a convex subset of:

$$\mathfrak{P} = \left\{ \xi \in \mathbb{R}^K \ : \ \sum_{k=1}^{K} \xi_k = 1, \ \xi \geq 0 \right\}.$$

Following Philpott et al. [159], we shall refer to the probability distribution ξ that attains the supremum as the *changed* probability distribution.

Definition 21. Given an original probability distribution $\{p_1, p_2, \ldots, p_K\}$ and a coherent risk measure $\mathbb{F}$, there exists a *changed* probability distribution $\{\xi_1, \xi_2, \ldots, \xi_K\}$ such that $\mathbb{F}[Z] = \mathbb{E}_{\xi}[Z]$.

The three risk measures described above (Expectation, AV@R, and Worst-case) can be expressed in terms of the set $\mathfrak{A}$ as follows:

- **Expectation:** If $\mathfrak{A}$ is a singleton, containing only the original probability distribution, then the risk measure $\mathbb{F}$ is equivalent to the expectation operator.
- **AV@R:** If $\mathfrak{A} = \left\{ \xi \in \mathfrak{P} \mid \xi_k \leq \frac{p_k}{\beta}, \ k = 1, 2, \ldots, K \right\}$, then the risk measure $\mathbb{F}$ is equivalent to AV@R$_{1-\beta}$.
- **Worst-case:** If $\mathfrak{A} = \mathfrak{P}$, then $\mathbb{F}$ is the Worst-case risk measure.

In this thesis we also use a convex combination of the Expectation and AV@R measures as proposed by Shapiro [187]:

$$\mathbb{F}[Z] = \lambda\, \mathbb{E}[Z] + (1 - \lambda) \text{AV@R}_{1-\beta}[Z]. \tag{1.2}$$

As the values of λ and β increase, the measure becomes less risk-averse.[3] It is easy to show that any convex combination of coherent risk measures is itself a coherent risk measure.

1.3.1 Dynamic risk measures

So far, we have only described a single-stage risk measure. In the multistage case, we refer to risk measures as *dynamic risk measures*. A thorough mathematical treatment can be found in

[3] This differs from other authors (e.g. Philpott and de Matos [160], Shapiro [187]) who prefer the $(1 - \lambda)$ on the expectation operator. We prefer this approach because λ and β both act in the same direction. In the alternative case, one has to remember that the measure becomes more risk-averse as λ increases or β decreases. Our approach has the nice property that as $\lambda \to 1$ or $\beta \to 1$, $\mathbb{F}[Z] \to \mathbb{E}[Z]$.

the following works [176, 184, 188, 189]; however, for this thesis, it is sufficient for the reader to have a more intuitive understanding.

To explain dynamic risk measures, we consider a scenario tree with three stages as shown in Figure 1.12 (ignore the dashed line for now). We define the set of nodes in the tree $n \in \mathcal{N}$ and leaves in $\mathcal{L}$. We denote children of node $n \in \mathcal{N} \setminus \mathcal{L}$ by n^+. With each node n, we associate a path $P(n)$, which gives the sequence of nodes from the root node to node n. For example, $P(8) = \{0, 1, 3, 8\}$. Each node n has an associated cost Z_n, which we have depicted on the incoming arc to the node in Figure 1.12. We assume that the cost of the root node is zero (i.e. $Z_0 = 0$). We also assume that the probability on each of the outgoing arcs from a node is 0.5.

To evaluate the risk of this scenario tree, we shall use the single-stage risk measure:

$$\mathbb{F}[X] = 0.5\,\mathbb{E}[X] + 0.5\max[X].$$

There are two main ways to evaluate the risk of this scenario tree: using either an *end-of-horizon* risk measure, or a *nested* risk measure. First, we explain an *end-of-horizon* risk measure. A graphical example is shown in Figure 1.12. The squares represent nodes in the scenario tree. The circle is the root node. The dashed square encompasses the nodes over which we are taking our risk measure.

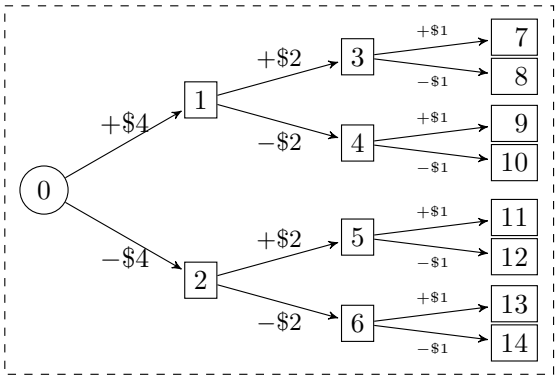


Figure 1.12: End-of-horizon risk measure acting on a scenario tree.

For each node n in the set of leaves $\mathcal{L}$, we define the *cumulative cost* $C(n) = \sum_{i \in P(n)} Z_i$. For example, $C(8) = Z_0 + Z_1 + Z_3 + Z_8 = 0 + 4 + 2 + -1 = \$5$. An *end-of-horizon* risk measure $\mathbb{F}$ takes the risk of the scenario tree by calculating the risk of the cumulative cost of all of the paths from the root node of the scenario tree to a leaf node. We denote this by $\mathbb{F}[\mathbf{C}]$, where $\mathbf{C}$ is the random variable drawn from the finite sample space $\{C(n) : n \in \mathcal{L}\}$ with uniform probability.

For our example, there are eight paths from the root node to a leaf node, each of which occurs with equal probability (i.e. $1/8$). These have corresponding cumulative costs of $-\$7$, $-\$5, \ldots, \$7$. Therefore, $\mathbb{E}[\mathbf{C}] = \$0$ and $\max[\mathbf{C}] = \$7$. Thus, the end-of-horizon risk measure

calculates the risk of the scenario tree as:

$$\mathbb{F}[\mathbf{C}] = 0.5\,\mathbb{E}[\mathbf{C}] + 0.5\max[\mathbf{C}] = 0.5 \times (0) + 0.5 \times (7) = \$3.5.$$

In contrast to the *end-of-horizon* risk measure, a *nested* risk measure takes the risk of the scenario tree according to a recursive definition. For each node n of the scenario tree, we define the *risk-adjusted cost* $V(n)$, which is the cost Z_n plus all of the future costs of the subtree rooted at node n:

$$V(n) = \begin{cases} Z_n & n \in \mathcal{L} \\ Z_n + \mathbb{F}_{\,m \in n^+}\left[V(m)\right] & n \in \mathcal{N} \setminus \mathcal{L}. \end{cases}$$

The nested risk measure takes the risk of the scenario tree as the risk-adjusted cost at the root node: $V(0)$.

Each of the risk measures in the recursion is evaluated at a node of the scenario tree and is *conditioned* on the events that have happened in the predecessor nodes of the tree. An example of this structure is given in Figure 1.13. We now explain how to compute the risk of a scenario tree using a nested risk measure. Compared with the end-of-horizon risk measure, the nested risk measure is more complicated to evaluate, so we do so in several steps. In the interest of brevity, we shall not detail the calculation of $V(n)$ at every node in the tree; however, we work through a few for illustrative purposes.

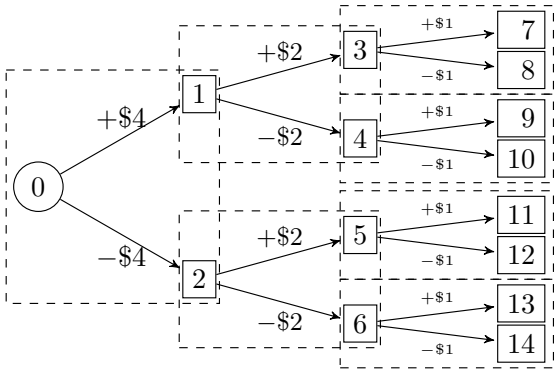


Figure 1.13: Nested risk measure acting on a scenario tree.

First, since node 7 is a leaf node, $V(7) = +\$1$. Now consider evaluating $V(3)$. We get:

$$V(3) = Z_3 + \mathbb{F}_{\,m \in 3^+}\left[V(m)\right] = Z_3 + 0.5\,\mathbb{E}_{\,m \in 3^+}\left[V(m)\right] + 0.5\max_{\,m \in 3^+}\left[V(m)\right].$$

The set of children $3^+ = \{7, 8\}$, which have corresponding costs of $V(7) = \$1$ and $V(8) = -\$1$. Therefore, $V(3) = 2 + 0.5 \times (0) + 0.5 \times (1) = \$2.5$. A similar argument can be constructed to show that $V(4) = -\$1.5$.

Now consider evaluating $V(1)$. We get:

$$V(1) = Z_1 + \mathbb{F}_{m \in 1^+}[V(m)] = Z_1 + 0.5\,\mathbb{E}_{m \in 1^+}[V(m)] + 0.5 \max_{m \in 1^+}[V(m)].$$

The set of children $1^+ = \{3, 4\}$, which have corresponding costs of $V(3) = \$2.5$ and $V(4) = -\$1.5$. Therefore, $V(1) = 4 + 0.5 \times (0.5) + 0.5 \times (2.5) = \$5.5$. A similar argument can be constructed to show that $V(2) = \$1.5$.

Finally, consider evaluating $V(0)$. We get:

$$V(0) = Z_0 + \mathbb{F}_{m \in 0^+}[V(m)] = Z_0 + 0.5\,\mathbb{E}_{m \in 0^+}[V(m)] + 0.5 \max_{m \in 0^+}[V(m)].$$

The set of children $0^+ = \{1, 2\}$, which have corresponding costs of $V(1) = \$5.5$ and $V(2) = \$1.5$. Therefore, $V(0) = 0 + 0.5 \times (5.5) + 0.5 \times (1.5) = \$3.5$.

In this case, the two methods of evaluating the risk of the scenario tree produced equivalent results. However, this is not always the case. In the following, we consider an example presented by Pflug and Pichler [155] using the scenario trees given in Figure 1.14.

In each of the sub-figures in Figure 1.14, we plot a scenario tree with three stages. The first stage is deterministic. For each node there is a 10% chance of transitioning upwards and a 90% change of transitioning downwards. In scenario tree (a), the possible end-of-horizon costs are: 7 with probability 0.01, 6 with probability 0.09, 3 with probability 0.09, and 2 with probability 0.81. In scenario tree (b), the corresponding outcomes are 8, 5, 4, and 1. As our risk measure we use AV@R with $\beta = 0.1$.

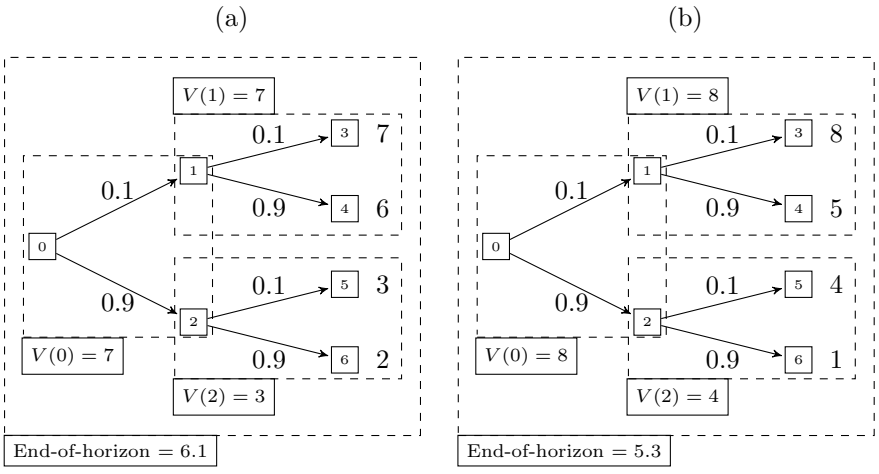


Figure 1.14: Different views of risk over two scenario trees (a) and (b) using the $\text{AV@R}_{1-0.1}$ risk measure. (This example is a slightly modified version of one given by Pflug and Pichler [155, Fig. 1].)

First, let us consider scenario tree (a). From the upper branch in the second stage, there are two possible future outcomes: 7 with probability 10% and 6 with probability 90%. Therefore, the average value-at-risk at the 10% quantile ($\text{AV@R}_{0.9}$) is 7. In the lower branch, the average

value-at-risk is 3. Stepping back to the first stage, the nested risk measure sees a cost-to-go of 7 with probability 10% and 3 with probability 90%. Therefore, the average value-at-risk is 7. However, if we calculate the average value-at-risk over the end-of-horizon outcomes, we find that the average value-at-risk is 6.1.

Now, let us consider scenario tree (b). From the upper branch in the second stage, there are two possible future outcomes: 8 with probability 10% and 5 with probability 90%. Therefore, the average value-at-risk at the 10% quantile (AV@$\mathrm{R}_{0.9}$) is 8. In the lower branch, the average value-at-risk is 4. Stepping back to the first stage, the nested risk measure sees a cost-to-go of 8 with probability 10% and 4 with probability 90%. Therefore, the average value-at-risk is 8. However, if we calculate the average value-at-risk over the end-of-horizon outcomes, we find that the average value-at-risk is 5.3.

Suppose that we are taking some action in stage one to determine the costs as given by scenario tree (a) or (b). From a nested viewpoint, we prefer scenario tree (a) over scenario tree (b). However, from an end-of-horizon viewpoint, we prefer scenario tree (b) over scenario tree (a). Thus, if we use the nested formulation of risk to guide our decision-making in stage one, we may make sub-optimal decisions from an end-of-horizon point-of-view.

Despite this problem, in this thesis we use nested risk measures because they are computationally efficient to include in stochastic dual dynamic programming, our solution method that we introduce in the next chapter. However, the difference between a nested and end-of-horizon risk measure is particularly relevant in Chapter 9 when we modify the single-stage risk measures in a nested risk measure with the view to modifying the end-of-horizon distribution of outcomes.

1.4 Standard form

To recap what we have discussed so far. We can decompose a multistage stochastic optimisation problem into a series of nodes. The nodes are linked together by state variables and we can describe the linkages by a policy graph $\mathcal{G}$. Associated with each node i is a decision-rule $\pi_i(x_i, \omega_i)$ ($\pi_i(x_i)$ for Decision-Hazard), which maps the incoming state variable x_i and realisation of a random noise ω_i, to a feasible control $u_i \in U_i(x_i, \omega_i)$ (in the Decision-Hazard case: $u_i \in U_i(x_i)$). As a result of taking the control u_i, the state transitions to the outgoing state x_i' according to the transition function $x_i' = T_i(x_i, u_i, \omega_i)$, and a cost of $C_i(x_i, u_i, \omega_i)$ is incurred. Then, the system transitions to a new node in the policy graph according to the probability distribution of Φ. All that remains is to define an optimisation problem that can be used to find the optimal decision-rule π_i for each node i. To do this, we utilise Bellman's *principle of optimality* and form the *cost-to-go* for each node i, given the incoming state x_i and realisation of the noise ω_i.

Definition 22. *Principle of Optimality*: An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions. (Bellman [22])

Definition 23. Given a policy π, the *cost-to-go* for node i is the stage-objective as a result of taking the control u_i, plus the risk-adjusted cost-to-go of the node's children:

$$V_i(x_i, \omega_i) = C_i(x_i, u_i, \omega_i) + \underset{j \in i^+;\ \omega_j \in \Omega_j}{\mathbb{F}} \left[V_j(T_i(x_i, u_i, \omega_i), \omega_j) \right],$$

where $u_i = \pi_i(x_i, \omega_i)$ if the node is Hazard-Decision, and $u_i = \pi_i(x_i)$ if the node is Decision-Hazard.

Note that we when we use the notation $\mathbb{F}_{j \in i^+;\ \omega_j \in \Omega_j}[\cdot]$, we mean that we apply the risk measure to the distribution of outcomes after sampling all of the noise realisations in all of the children of node i. Since the noise realisation and node transition are independent, from node i, the probability of sampling ω_j in the next node is $\phi_{i,j} \times p_{\omega_j}$ (i.e. the probability of transitioning from node i to node j, multiplied by the probability of observing ω_j in node j). Also note that when maximising, we refer to the *value-to-go* instead of the cost-to-go.

Putting together all that we have discussed so far, we can now define the class of multistage stochastic optimisation problems that we consider in the remainder of this thesis.

Definition 24. Given a policy graph $\mathcal{G}$, a *multistage stochastic optimisation problem* is an optimisation problem of the form:

$$\min_{\pi} \left\{ \underset{i \in R^+;\ \omega_i \in \Omega_i}{\mathbb{F}} \left[V_i(x_R, \omega_i) \right] \right\}, \tag{1.3}$$

where x_R is the initial condition of the state variable at the root node.

Readers should note that this definition does not encompass all multistage stochastic optimisation problems. For example, it does not consider continuous time problems, and due to the recursive definition of the cost-to-go function, we only use nested risk measures. However, this framework also allows single-stage risk measures within each subproblem (e.g. expected conditional risk measures [109]), and end-of-horizon risk measures by dynamically changing the single-stage risk measures in each subproblem (e.g. Baucke et al. [14], Pflug and Pichler [156]). In addition, the nonanticipative constraints usually associated with multistage stochastic programming are satisfied by our definition of a policy.

The goal of multistage stochastic optimisation is to find the policy π that minimises Eq. 1.3. If the policy π is not given explicitly, the cost-to-go from the start of node i can be formulated as an optimisation problem. The formulation is different depending upon whether the node is Hazard-Decision or Decision-Hazard. We call the optimisation problem associated with each node a *subproblem*.

Definition 25. A *Hazard-Decision subproblem* is the optimisation problem:

$$\begin{aligned} \mathbf{HD}_i(x_i, \omega_i): \quad V_i(x_i, \omega_i) = \min_{u_i} \quad & C_i(x_i, u_i, \omega_i) + \mathop{\mathbb{F}}_{j \in i^+;\ \omega_j \in \Omega_j} [V_j(x_i', \omega_j)] \\ \text{s.t.} \quad & x_i' = T_i(x_i, u_i, \omega_i) \\ & u_i \in U_i(x_i, \omega_i), \end{aligned} \tag{1.4}$$

where the decision-rule $\pi_i(x_i, \omega_i) \in \arg\min_{u_i} \mathbf{HD}_i(x_i, \omega_i)$.

Definition 26. A *Decision-Hazard subproblem* is the optimisation problem:

$$\begin{aligned} \mathbf{DH}_i(x_i): \quad V_i(x_i) = \min_{u_i} \quad & \mathop{\mathbb{F}}_{\omega_i \in \Omega_i;\ j \in i^+} \left[C_i(x_i, u_i, \omega_i) + V_j\left(x'_{i,\omega_i}\right) \right] \\ \text{s.t.} \quad & x'_{i,\omega_i} = T_i(x_i, u_i, \omega_i), \quad \forall \omega_i \in \Omega_i \\ & u_i \in U_i(x_i) \end{aligned} \tag{1.5}$$

where the decision-rule $\pi_i(x_i) \in \arg\min_{u_i} \mathbf{DH}_i(x_i)$.

Finally, a few last pieces of terminology. Given a policy π, we can *simulate* the multistage stochastic optimisation problem by the procedure given in Algorithm 1. As a result, we end up with a sequence of nodes i^n, states x^n, controls u^n, noise terms ω^n, and costs c^n for $n \in \{1, \ldots, N\}$.

Algorithm 1: Simulating the policy.

set $x^1 = x_R$
set $n = 1$
set $i^0 = R$
while $i^{n-1^+} \neq \varnothing$ **do**
  sample i^n from i^{n-1^+}
  sample ω^n from Ω_{i^n}
  set $u^n = \pi_{i^n}(x^n, \omega^n)$
  set $c^n = C_{i^n}(x^n, u^n, \omega^n)$
  set $x^{n+1} = T_{i^n}(x^n, u^n, \omega^n)$
  set $n = n + 1$
end

We also need to define a *scenario* and the *cumulative cost* of a scenario.

Definition 27. A *scenario* is a sequence of node and noise realisations $(i^1, \omega^1), (i^2, \omega^2), \ldots, (i^N, \omega^N)$.

Definition 28. The *cumulative cost* of a scenario is $\sum_{n=1}^{N} c^n$.

1.5 Summary

This chapter introduced the reader to multistage stochastic optimisation. We began the chapter by clearly defining the terminology that we shall use in this thesis. Then, we identified the two building blocks in multistage stochastic optimisation: the Hazard-Decision node, and the Decision-Hazard node. Next, we introduced the concept of a policy graph. A policy graph encapsulates the structure of the multistage stochastic optimisation problem by defining the relationships between nodes. We also introduced a graphical representation of the policy graph as a shorthand for communicating the structure of the problem.

We now have all the tools necessary to formulate multistage stochastic optimisation problems. However, there can be many ways of modelling the same problem. For example, the modeller can choose between a compact representation like Figure 1.4 and an expanded representation like Figure 1.5, and even between Decision-Hazard and Hazard-Decision nodes like we showed in Figures 1.2 and 1.3. Therefore, it is useful to consider the solution approach at the modelling step in order to formulate the problem in a way that is best suited to the particular solution approach. The solution approach that we use in this thesis is called *stochastic dual dynamic programming* (SDDP). In the next chapter, we provide an overview of the SDDP algorithm and survey the current state-of-the-art.

Chapter 2

Stochastic dual dynamic programming

In the previous chapter, we introduced multistage stochastic optimisation. Such problems are notoriously difficult to solve [164]. There are a few contributing reasons for this. First, many real-world applications, such as the ones we discuss in Chapters 8 and 9 of this thesis, have multiple state and control variables. Therefore, they quickly run into dynamic programming's well-known "curse of dimensionality" [23]. Compounding this curse is the stochastic element of the problem. Since the noise term is sampled independently in each subproblem, a linear policy graph with T stages has $\prod_{t=1}^{T} |\Omega_t|$ possible realisations for the sequence of noise terms $\omega_1, \omega_2, \ldots, \omega_T$. Therefore, the total number of possible realisations increases exponentially as the number of stages increases, and polynomially (with order T) as the number of elements in the sample space for each noise term increases. (The example in Chapter 8 has on the order of 10^{68} scenarios!) A number of solution methods are available for solving multistage stochastic optimisation problems (e.g. approximate dynamic programming [164], progressive hedging [178], and stochastic dynamic programming, which we describe in Appendix A). However, in this thesis, we focus on one particular method: *stochastic dual dynamic programming* (SDDP).

2.1 Introduction

The stochastic dual dynamic programming (SDDP) algorithm dates back to the seminal work of Pereira and Pinto [153]. However, the idea can be traced back to Benders decomposition [25] and the L-shaped method of Van Slyke and Wets [199] (originally for two-stage problems, it was extended to the multistage case by Louveaux [130]). The method has been studied extensively in the literature [161, 187, 189]. We can also recommend the accessible introductions to SDDP provided by Newham [145, Ch. 4] and Guan [91, Ch. 5]. Although the basic method itself is simple, many refinements and modifications have been suggested since its inception. In this chapter, we introduce the basic algorithm for readers unfamiliar with its workings and survey the current state-of-the-art implementations. Readers should also be aware that, in addition to the original algorithm, the term SDDP can refer to a broad class of sampling algorithms, all

of which employ similar techniques (e.g. AND [58], CUPPS [42], DOASA [161], and ReSa [107]). Though we call the algorithm described in this section "SDDP", it is perhaps closest to the DOASA[1] algorithm of Philpott and Guan [161].

For multistage stochastic linear programs, Philpott and Guan [161] have shown that the SDDP algorithm will converge to an optimal policy almost surely in a finite number of iterations. (Earlier proofs, such as Chen and Powell [42] and Linowsky and Philpott [127], made use of an unstated assumption regarding the application of the second Borel-Cantelli lemma [89].) The convergence result has since been extended to more general convex problems by Girardeau et al. [84] and to convex, risk-averse problems by Guigues [93].

There are two main variants of the SDDP algorithm: average-cut and multi-cut. In this thesis, we only consider the average-cut variant. Consequently, we shall also restrict our attention to Hazard-Decision nodes (the details of why should not concern the reader), and we use the term *node* interchangeably with *Hazard-Decision node* (and subproblem interchangeably with Hazard-Decision subproblem). However, we note that this is not a limiting assumption; recall that we showed earlier how any Decision-Hazard node can be expressed as two Hazard-Decision nodes with an expanded number of state variables.

The chapter is laid out as follows. First, in Section 2.2 we introduce the basic SDDP algorithm. Then, in Section 2.3 we describe various ways of terminating the algorithm. In Section 2.4 we describe *cut selection*, an important heuristic for improving the computational performance of the SDDP algorithm. We describe a variety of other computational improvements in Section 2.5. In the final two sections, we explore two issues with SDDP: numerical issues in Section 2.6 and degeneracy in Section 2.7. We conclude with a short summary in Section 2.8.

2.2 The algorithm

Recall that a Hazard-Decision subproblem is the optimisation problem:

$$\begin{aligned} \mathbf{HD}_i(x_i, \omega_i): \quad V_i(x_i, \omega_i) = \min_{u_i} \quad & C_i(x_i, u_i, \omega_i) + \mathop{\mathbb{F}}_{j \in i^+;\ \omega_j \in \Omega_j} [V_j(x_i', \omega_j)] \\ \text{s.t.} \quad & x_i' = T_i(x_i, u_i, \omega_i) \\ & u_i \in U_i(x_i, \omega_i), \end{aligned} \tag{2.1}$$

where the decision-rule $\pi_i(x_i, \omega_i) \in \arg\min_{u_i} \mathbf{HD}_i(x_i, \omega_i)$.

Evaluating the future cost component of the objective (which we call the *cost-to-go* function) in each subproblem $\mathbf{HD}_i$ is non-trivial. However, the recursive definition of a multistage stochastic program makes the problem amenable to dynamic programming [23]. One possible solution method is *backward recursion*, which we describe in Appendix A (along with `DynamicProgramming.jl`, the Julia library that we have developed to solve multistage stochastic optimisation problems using backward recursion). This method works by recursively

[1] Dynamic Outer Approximation Sampling Algorithm.

evaluating the risk-adjusted cost-to-go at a discretised set of points in the state-space. To evaluate the cost-to-go at a point in the state-space that is not in the discretised set of points, an interpolation scheme is used. However, this method quickly runs into the "curse of dimensionality." (We will observe this curse in Chapter 7.)

Instead of evaluating the cost-to-go function at a set of discretised points, SDDP builds an outer approximation of the cost-to-go function using hyperplanes. For this reason, a number of technical assumptions are required. Given a problem represented by the policy graph $\mathcal{G} = (R, \mathcal{N}, \mathcal{E}, \Phi)$, these assumptions are:

(A1) given a fixed ω_i, the function $V_i(x_i, \omega_i)$ is convex with respect to x_i at each node $i \in \mathcal{N}$;
(A2) the sample space Ω_i of random noise outcomes is finite at each node $i \in \mathcal{N}$;
(A3) the policy graph $\mathcal{G}$ is acyclic with a finite number of subproblems; and
(A4) the feasible region of each subproblem $i \in \mathcal{N}$ is non-empty, and there exists an optimal solution for every reachable incoming state x_i and realisation of the noise ω_i.

Readers should note that missing from this list is the requirement that random quantities in different stages are independent; this is implicitly satisfied by the definition of a policy graph.

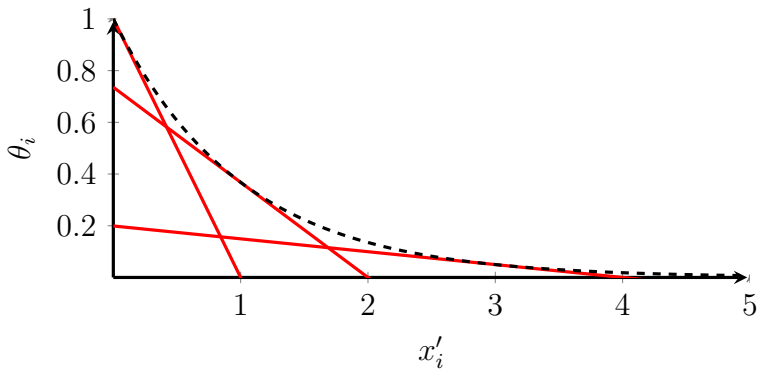


Figure 2.1: Outer approximation of the function $\theta_i \geq e^{-x'_i}$ using cuts.

Due to assumption (A1), the cost-to-go function (i.e. $\mathbb{F}_{j \in i^+;\ \omega_j \in \Omega_j}\left[V_j(x'_i, \omega_j)\right]$) is convex with respect to the state variable x'_i. Therefore, it can be replaced by a variable θ_i and approximated by the maximum of a set of linear functions, which we refer to as *cuts* (an example is shown in Figure 2.1). SDDP constructs the set of cuts iteratively. Each iteration consists of two phases: a forward pass, which samples a sequence of subproblems and values for the state variables $\mathcal{S} = [(i_1, x'_{i_1}), (i_2, x'_{i_2}), \dots]$, and a backward pass, which refines the approximation of the cost-to-go function by adding a new cut to each of the subproblems visited in the forward pass. The

resulting approximated subproblem at node i after K iterations can be expressed as:

$$\begin{aligned} \mathbf{SP}_i^K(x_i, \omega_i): \quad V_i^K(x_i, \omega_i) = \min_{u_i} \quad & C_i(\bar{x}_i, u_i, \omega_i) + \theta_i \\ \text{s.t.} \quad & x_i' = T_i(\bar{x}_i, u_i, \omega_i) \\ & \bar{x}_i = x_i, \quad [\lambda_i] \\ & u_i \in U_i(\bar{x}_i, \omega_i) \\ & \theta_i \geq \alpha_i^k + \beta_i^k x_i', \quad k \in \{1, 2, \ldots, K\}. \end{aligned} \tag{2.2}$$

Note that λ_i is the vector of dual variables associated with the constraints $\bar{x}_i = x_i$. In other words, λ_i is a valid subgradient for the function $V_i^K(x_i, \omega_i)$ with respect to x_i. In the literature many authors (e.g. [153, 161]) compute the subgradient of $V_i^K(x_i, \omega_i)$ with respect to the incoming state variables x_i by computing a transformation of the dual of the transition constraints (i.e. $x_i' = T_i(\bar{x}_i, u_i, \omega_i)$), taking into account the feasible set of actions U_i. This approach is overly complicated. Our solution (also used by Girardeau et al. [84]) is to introduce a dummy variable $\bar{x}_i$ with the constraint:

$$\bar{x}_i = x_i, \quad [\lambda_i].$$

This has downside of adding one extra variable and constraint for each state variable, but results in a simpler subgradient calculation.

2.2.1 Computing cut coefficients

The description above does not detail how to calculate the cut coefficients α and β. We address this question below.

Consider a node j, given an incoming state variable $\hat{x}$ and a realisation of the noise ω_j. We can solve $\mathbf{SP}_j^K$ and record the optimal objective value $V_j^K(\hat{x}, \omega_j)$, which we denote $\bar{V}_{j,\omega_j}^K$, and the optimal value of the dual variable λ_j, which we denote $\bar{\lambda}_{j,\omega_j}^K$. Since V_j^K is convex with respect to x_j, we have that:

$$V_j^K(x_j, \omega_j) \geq \bar{V}_{j,\omega_j}^K + \langle \bar{\lambda}_{j,\omega_j}^K, (x_j - \hat{x}) \rangle,$$

where $\langle , \rangle$ is the inner product. Re-arranging terms we get:

$$V_j^K(x_j, \omega_j) \geq \left(\bar{V}_{j,\omega_j}^K - \langle \bar{\lambda}_{j,\omega_j}^K, \hat{x} \rangle \right) + \langle \bar{\lambda}_{j,\omega_j}^K, x_j \rangle.$$

However, recall that we wish to approximate the cost-to-go function $\mathbb{F}_{j \in i^+;\, \omega_j \in \Omega_j}[V_j(x_i', \omega_j)]$. To do so, consider the following proposition from Philpott et al. [159, Proposition 4], which we restate using our notation without proof. (Recall that ξ is the *changed* probability distribution from Section 1.3.)

Proposition 1. *Suppose for each $\omega \in \Omega$, that $g(\hat{x}, \omega)$ is a subgradient of $Z(x, \omega)$ at $\hat{x}$. Then, given $\mathbb{F}[Z(x, \omega)] = \mathbb{E}_\xi[Z(x, \omega)]$, $\mathbb{E}_\xi[g(\hat{x}, \omega)]$ is a subgradient of $\mathbb{F}[Z(x, \omega)]$ at $\hat{x}$.*

Using this result, we can construct a valid cut via the method given in Algorithm 2.

Algorithm 2: Cut calculation algorithm.

Solve $\mathbf{SP}_j^K(\hat{x}, \omega_j)$ for all $j \in i^+$ and $\omega_j \in \Omega_j$
Compute ξ so that $\mathbb{E}_\xi\left[\bar{V}_{j,\omega_j}^K\right] = \mathbb{F}_{j\in i^+;\omega_j\in\Omega_j}\left[\bar{V}_{j,\omega_j}^K\right]$
Set $\beta_i^{K+1} = \mathbb{E}_\xi\left[\bar{\lambda}_{j,\omega_j}^K\right]$
Set $\alpha_i^{K+1} = \mathbb{E}_\xi\left[\bar{V}_{j,\omega_j}^K\right] - \langle\beta_i^{K+1}, \hat{x}\rangle$
Obtain the inequality $\mathbb{F}_{j\in i^+;\ \omega_j\in\Omega_j}\left[V_j(x_i', \omega_j)\right] = \theta_i \geq \alpha_i^{K+1} + \langle\beta_i^{K+1}, x_i'\rangle$

We now describe an iteration of the SDDP algorithm in more detail.

2.2.2 An iteration

Each iteration consists of a forward pass, in which a sequence of nodes and states is sampled, and a backward pass, in which the cost-to-go function at each node visited on the forward pass is refined at the point in the state-space sampled on the forward pass.

Forward pass

To begin the forward pass in iteration K, we set our current state x to the value of the outgoing state variable at the root node x_R, and we set the list of visited node-state pairs $\mathcal{S}$ to the empty list. Then, we apply the following procedure.

(a) Sample node i from the children of node R according to the policy graph transition matrix Φ, and then transition to that node.
(b) If i^+ is empty, terminate the forward pass; otherwise continue to step (c).
(c) Sample a noise ω_i from Ω_i.
(d) Solve $\mathbf{SP}_i^K(x, \omega_i)$.
(e) Append (i, x_i') to the list $\mathcal{S}$.
(f) Set $x = x_i'$.
(g) Sample a new i from i^+ according to the transition matrix Φ, then go to step (b).

Backward pass

On the backward pass, we backtrack up the sequence of node-state pairs $\mathcal{S}$ that we visited on the forward pass. For each node-state pair (i_n, x_{i_n}') in $\mathcal{S}$, we set $i = i_n$ and $\hat{x} = x_{i_n}'$, and then construct the cut according to the cut-calculation method given in Algorithm 2. The resulting cut is added to the subproblem $\mathbf{SP}_{i_n}^K$ so that it becomes $\mathbf{SP}_{i_n}^{K+1}$. Pseudo-code for the full SDDP algorithm is given in Algorithm 3.

Algorithm 3: The complete SDDP algorithm.

set $K = 0$
while *not stopped* **do**
  /* Forward Pass */
  set $x = x_R$, $\mathcal{S} = [\,]$
  sample i from R^+ according to the transition matrix Φ
  while $i^+ \neq \varnothing$ **do**
    sample ω_i from Ω_i
    solve $\mathbf{SP}_i^K(x, \omega_i)$
    append (i, x_i') to the list $\mathcal{S}$
    set $x = x_i'$
    sample new i from i^+ according to the transition matrix Φ
  end
  /* Backward Pass */
  for $(i, \hat{x})$ *in* $reverse(\mathcal{S})$ **do**
    for $j \in i^+$ **do**
      for $\omega_j \in \Omega_j$ **do**
        solve $\mathbf{SP}_j^K(\hat{x}, \omega_j)$
        set $\bar{V}_{j,\omega_j}^K$ to the optimal objective value
        set $\bar{\lambda}_{j,\omega_j}^K$ to the value of λ_j in the optimal solution
      end
    end
    compute ξ so that $\mathbb{E}_\xi\left[\bar{V}_{j,\omega_j}^K\right] = \mathbb{F}_{j\in i^+;\omega_j\in\Omega_j}\left[\bar{V}_{j,\omega_j}^K\right]$
    set $\beta_i^{K+1} = \mathbb{E}_\xi\left[\bar{\lambda}_{j,\omega_j}^K\right]$
    set $\alpha_i^{K+1} = \mathbb{E}_\xi\left[\bar{V}_{j,\omega_j}^K\right] - \langle\beta_i^{K+1}, \hat{x}\rangle$
    add the cut $\theta_i \geq \alpha_i^{K+1} + \langle\beta_i^{K+1}, x_i'\rangle$ to $\mathbf{SP}_i^K$
  end
  increment K
end

Initialisation

Note that in the first iteration (i.e. $K = 0$), there are no cuts. As a consequence, the cost-to-go variable θ_i is free and the subproblem $\mathbf{SP}_i^0$ is unbounded. Therefore, it is necessary to add a valid lower bound to the cost-to-go variable so that $\theta_i \geq -M$, where M is some sufficiently large number. The question of "sufficient" is somewhat ill-defined. The bound should not be as large as possible (since this will help convergence and avoid some numerical issues), but not so small as to cut off some of the optimal region. In most cases, the modeller can bring domain knowledge to identify a good bound. However, if this is not the case, we recommend solving the model using a few different bounds and confirming that the results are the same. In addition, when simulating the policy, the modeller should check if the cost-to-go variable ever takes the solution $\theta_i = -M$ at any node i in the policy graph, since this could indicate that the problem has not yet converged or that the bound M is too small.

2.3 Stopping rules

Recall that Philpott and Guan [161] (among others) have shown that the SDDP algorithm will converge to an optimal policy almost surely in a finite number of iterations. In practice, this number is likely to be very large. Moreover, they do not specify a convergence criterion. Therefore, we need some way of pre-emptively terminating SDDP when the solution is "good enough." Several ways have been proposed in the literature, which we refer to as *stopping rules*, and we review them in this section. However, before we begin, observe that since the cuts form an outer-approximation of the cost-to-go function, we can obtain a valid lower bound $\underline{z}$ to the original multistage stochastic optimisation problem (i.e. Eq. 1.3) by evaluating:

$$\underline{z} = \underset{i \in R^+;\ \omega_i \in \Omega_i}{\mathbb{F}} \left[V_i^K(x_R, \omega_i) \right].$$

We shall refer to this as the *lower bound*. We now proceed to describe the *statistical stopping rule*.

2.3.1 The statistical stopping rule

One way to obtain an upper bound for the problem is to simulate every possible sequence of subproblem and noise realisations (recall that we refer to one single sequence as a *scenario*). However, in practice, the number of scenarios is very large. (The models we solve in Chapter 8 of this thesis have on the order of 10^{68} scenarios.) As a consequence, this approach of evaluating the upper bound is intractable. If the model is risk-neutral, then Monte Carlo simulation can be used to obtain an estimate for the upper bound. However, Monte Carlo simulation is less straightforward for risk-averse models. Each (iid) replication of the Monte Carlo simulation samples a scenario from the full sample space of scenarios, solves each subproblem in sequence, then sums the stage-objectives to produce a single cumulative cost for the scenario. Pseudo-code for this is given in Algorithm 1 (from Section 1.4).

Using the cumulative costs from the N Monte Carlo replications y_1, y_2, ..., y_N, Pereira and Pinto [153] form a confidence interval for the mean of the expected cumulative cost (i.e. the upper bound). Based on the lower bound $\underline{z}$ and this confidence interval, they propose the following stopping rule, which we call the *statistical stopping rule*.

Definition 29. *Statistical stopping rule*: Given a lower bound $\underline{z}$ and cumulative costs y_1, y_2, ..., y_N for the N Monte Carlo replications, terminate SDDP when the lower bound $\underline{z}$ lies above the lower limit of a confidence interval constructed for the mean of the Monte Carlo replications. That is, when:

$$\underline{z} \geq \mu_y - z\sqrt{\frac{1}{N(N-1)}\sum_{i=1}^{N}(\mu_u - y_i)^2},$$

where $\mu_y = \frac{1}{N}\sum_{i=1}^{N} y_i$, and z is the z-score for a certain level of confidence (e.g. $z = 1.96$ for a 95% confidence interval).

There are two problems with this stopping rule. First, if the cumulative costs have a large variance, or we conduct too few simulations (i.e. N is small), then the confidence interval will be very wide. Therefore, we may terminate too early. As a result, choosing when to simulate the policy, how many replications to conduct, and the level-of-confidence at which to build the confidence interval all have an impact on the quality of the policy. It is up to the practitioner to determine what they deem is "good enough." Second, this method can only be used when the risk measure is the expectation operator. This rules out a large class of problems.

However, given the risk measure is the expectation operator, several improvements have been proposed. Most of these improvements are focused on reducing the variance of the estimator for the expected cost (i.e. the upper bound). Therefore, instead of sampling independent and identically distributed (iid) scenarios for the Monte Carlo replications, variance reduction strategies (such as antithetic variates, control variates, importance sampling, Latin hypercube sampling, and quasi-Monte Carlo sampling) are used [21, 56, 108, 110]. These strategies reduce the variance of the estimator, but do not address the problem of when to perform the Monte Carlo simulation and how many replications to conduct. The problem of how many replications to conduct is addressed by Bayraksan and Morton [20] in their method of *sequential sampling*.

Sequential sampling

The basic idea of sequential sampling is to conduct a small number of Monte Carlo replications and then construct the confidence interval. If the confidence interval shows no evidence of convergence, we restart the iterations of SDDP. Otherwise, we perform more replications and re-check. Once we have sampled a large number of replications and the statistical stopping rule still holds, we terminate the SDDP algorithm. In Algorithm 4, we give the pseudo-code for a modified version of the *sequential sampling* method. Compared with Bayraksan and Morton [20], the key differences are that we place an upper bound $N_{\max}$ on the number of replications (typically very large), and that we choose a fixed step size of N_Δ.

Sequential sampling overcomes many of the problems associated with the statistical stopping rule. Since we terminate the simulation step early if there is no evidence of convergence (thereby avoiding unnecessary computation), the stopping rule can be applied frequently. However, since we can make the upper bound $N_{\max}$ very large, we can be sure that we only terminate the SDDP algorithm after conducting a large number of replications. Moreover, the sequential sampling algorithm can be used in conjunction with the variance reduction techniques mentioned previously to further improve the performance.

One problem that sequential sampling does not overcome is the limitation to risk-neutral problems. To terminate risk-averse problems, different stopping rules must be used.

Algorithm 4: Modified sequential sampling algorithm.

input : $N_{\min}$, $N_{\max}$, N_{Δ}, $\underline{z}$
set $N^0 = 0$, $N^1 = N_{\min}$, $k = 1$, $converged = true$
while $(N^k \leq N_{\max}) \wedge (converged)$ **do**
  conduct $N^k - N^{k-1}$ replications to obtain estimates $y_{N^{k-1}+1}, y_{N^{k-1}+2}, \ldots, y_{N^k}$
  set $\mu_y = \frac{1}{N^k} \sum_{i=1}^{N^k} y_i$
  set $\overline{z} = \mu_y - z\sqrt{\frac{1}{N^k(N^k-1)} \sum_{i=1}^{N^k} (\mu_y - y_i)^2}$
  if $\underline{z} \leq \overline{z}$ **then**
    set $converged = false$
  else
    set $N^{k+1} = N^k + N_{\Delta}$
    set $k = k + 1$
  end
end
output: The Boolean $converged$ indicating if the algorithm has converged.

2.3.2 The bound stalling stopping rule

Instead of computing an upper bound, we can use the evolution of the lower bound as an estimator of convergence. Some authors (e.g. [91, 145]) suggest terminating the SDDP algorithm if the lower bound fails to change by more than a certain tolerance for a given number of consecutive iterations. We refer to this as *bound stalling*.

Definition 30. *Bound stalling stopping rule*: Given a sequence of lower bounds $\underline{z}_1, \underline{z}_2, \ldots, \underline{z}_K$, a parameter $n \in \{1, 2, \ldots, K-1\}$, and a tolerance $\varepsilon \geq 0$, terminate SDDP if:

$$\underline{z}_i - \underline{z}_{i-1} \leq \varepsilon, \quad \forall i \in \{K, K-1, \ldots, K-n\}.$$

The benefits of the bound stalling stopping rule are that it is simple to calculate and that it can be used on risk-averse problems. However, like the statistical stopping rule, it has some flaws. The largest is that the lower bound could continue to improve at a very slow rate for many iterations. Therefore, there is no guarantee concerning how far the lower bound is from the upper bound when it "stalls". The bound may also stall early in the algorithm if, through "bad luck", we sample a short sequence of scenarios which do not improve the lower bound.

2.3.3 Other limits

The previous two stopping rules (statistical and bound stalling) both have the risk of terminating the algorithm with a sub-optimal solution. Therefore, some authors (e.g. [85, 158]) advocate choosing either a fixed number of iterations to perform, or a time limit before terminating.

Definition 31. *Time limit stopping rule*: Terminate SDDP after s seconds of computation.

Definition 32. *Iteration limit stopping rule*: Terminate SDDP after k iterations.

There are two reasons for preferring these stopping rules over the statistical and bound stalling stopping rules. First, the statistical stopping rule is not suitable for risk-averse problems. Second, the bound stalling stopping rule does not consider the quality of the policy. The approach we use in practice is to set an iteration limit, solve and then simulate the policy, and then plot the solution. It is usually easy to identify if the solution has not converged as the policy will make "dumb" decisions such as selling items at a low price or incurring large penalties for violating certain constraints. If sub-optimal decisions are identified, we perform more iterations. This is a highly subjective stopping rule. However, we find that it works well in practice. (We discuss this further in Chapter 9.)

Before we finish our discussion on stopping rules for SDDP, it is worth briefly mentioning three deterministic methods for evaluating the upper bound.

2.3.4 Deterministic convergence

First, Philpott et al. [159] develop a deterministic upper bound for risk-averse problems. This is an attractive feature as it can be used as a deterministic stopping rule. It can also be used to direct the search towards areas of the state-space that require improvement, rather than the random sampling that occurs in the traditional algorithm. However, in their experiments, the computational performance of the method was poor, requiring an order of magnitude more time to construct the upper bound than the lower bound. Moreover, the computational effort increases dramatically with the number of state variables, limiting the method to low-dimensional problems [159].

Baucke et al. [13] construct the upper bound via an outer approximation of the dual problem. Moreover, the algorithm they propose does not randomly sample the noise terms in each subproblem on the forward pass. Instead, they use the upper and lower bounds to direct the search towards the area of the state-space with the largest gap between the upper and lower bounds.

Leclère et al. [124] propose a similar method to Baucke et al. and also construct the upper bound via an outer approximation of the dual problem. However, their algorithm samples states in both the primal- and dual-SDDP subproblems. This allows both problems to be solved in parallel.

2.4 Cut selection

As the SDDP algorithm progresses, many cuts (typically thousands) are added to each subproblem. This increases the computational effort required to solve each subproblem. (A real-world model may begin with subproblems that contain tens of variables and constraints, only to add 10,000 constraints!) In addition, many of the cuts discovered early in the solution process may

be redundant once additional cuts are added. This issue has spurred a vein of research (e.g. [11, 55, 142, 154]) into heuristics for choosing cuts to keep or remove; henceforth referred to as *cut selection*. These heuristics attempt to select a subset of cuts to keep, while trading off two opposing forces:

1. minimising the number of cuts in order to reduce the solve time of each subproblem; and
2. maximising the number of cuts in order to improve the approximation of the cost-to-go function.

In addition, since any time spent computing the set of cuts to keep is time not spent discovering cuts, the heuristics should be computationally cheap to evaluate.

Various authors (e.g. [11, 55, 115]) have reported that cut selection can offer an order of magnitude reduction in the solution time required to conduct a fixed number of iterations with minimal degradation in solution quality.

2.4.1 Heuristics

One approach to selecting cuts is to maintain all the cuts except those that can be proven to be never binding in the optimal solution of $\mathbf{SP}_i^K$ for any reachable values of x_i and ω_i. For the j^{th} cut in subproblem i, this can be done by solving the following LP (which was called the *test of usefulness* by Pfeiffer et al. [154]):

$$\begin{aligned} D_i(j) = \min_{x_i', \theta_i} \quad & \theta_i - \left(\alpha_i^j + \langle \beta_i^j,\ x_i' \rangle\right) \\ \text{s.t.} \quad & \theta_i \geq \alpha_i^k + \langle \beta_i^k,\ x_i' \rangle, \quad k \in \{1, 2, \ldots, K\}. \end{aligned} \tag{2.3}$$

If the objective value $D_i(j) > 0$, then we can conclude that cut j is not binding anywhere in the state-space, and it can be safely deleted. We refer to this heuristic as *exact cut selection*. Since we only delete cuts that are proven to be never binding, this cut selection heuristic maintains the best approximation of the cost-to-go function. However, it is expensive to compute since it involves solving an LP for every cut in every subproblem.

Perhaps the most comprehensive exploration of cut selection has been conducted by de Matos et al. [55], who experimentally analysed the different performance of various cut selection heuristics. We now describe the best performing heuristic from that analysis, which they call *Level One cut selection*.

Level One cut selection

The intuition behind Level One cut selection is that we want to keep all the cuts that are binding in some part of the state-space. However, instead of solving the expensive *test of usefulness*, we can discretise the state-space and record the best cut at each of the discretised points. Unfortunately, this suffers from the same curse of dimensionality as dynamic programming. Therefore,

Level One cut selection only records the best cut at the states visited on the forward pass. We call the best cut the *dominating cut*. At a point in the state-space $\hat{x}$ in node i, the *value* of the dominating cut is given by:

$$\bar{\theta} = \max_{k=1,2,\ldots,K} \alpha_i^k + \langle \beta_i^k, \hat{x} \rangle. \tag{2.4}$$

We call the argument k^* which achieves the maximum the *index* of the dominating cut. The Level One cut selection method works as follows (for notational simplicity, we drop the subproblem index i). After K iterations, we store for $k = 1, 2, \ldots, K$:

1. the list of visited states (i.e. x_i'), which we denote $\bar{x}^1, \ldots, \bar{x}^K$;
2. the index of the dominating cut at each sampled state $\bar{x}^k$, which we denote $j^1, \ldots, j^K$;
3. the value of the dominating cut at each state, which we denote $\bar{\theta}^1, \ldots, \bar{\theta}^K$; and
4. the number of states at which the cut k is dominant, which we denote $d^1, \ldots, d^K$.

After adding the cut to subproblem i in iteration $K+1$, we update the data listed above as follows. First, we assume that the new cut is dominant for the sampled point. (This is not necessarily true if other cuts are dominating but have been removed in a previous iteration.) Then, we loop through the previously visited states and evaluate the new cut at those points. If we find that the new cut is dominating, we update j^k and $\bar{\theta}^k$ and increment our counter for d^{K+1}. Having looped through the previously visited states, we perform a similar loop comparing previously discovered cuts evaluated at the new state, since a cut that was removed in a previous iteration may now be the dominating cut. Pseudo-code is given in Algorithm 5. At any time, the set of cuts that we should keep in subproblem i is:

$$\{k : \ d_i^k > 0, \ k = 1, 2, \ldots, K\}.$$

We favour Level One cut selection over other approaches (e.g. [142, 154]) since it is computationally cheap to evaluate and out-performs many other cut selection heuristics [55]. Another approach, proposed by Löhndorf et al. [129], is to reject cuts if they do not improve the cost-to-go approximation by more than some small value ϵ; however, we have not tested this method.

2.4.2 Implementation

Notably missing from papers such as de Matos et al. [55] is a description of how to implement, computationally, the addition and removal of cuts from the subproblems. For example, should we apply the cut selection heuristic every time we find a new cut? Or should we cache a number of cuts and then process them in batch form. At the solver-level, should we delete cuts that are no longer selected and add new cuts? Or should we modify, in-place, the coefficients α and β of a constraint $\theta \geq \alpha^\top x + \beta$ that is not selected with new coefficients α^* and β^*?

`SDDP.jl`, a library that we have developed for solving multistage stochastic optimisation problems using SDDP (described in the next chapter), has some technical limitations which

Algorithm 5: Level One update step.

```
/* Assume new cut is dominant at new state.                    */
set d^{K+1} = 1
set j^{K+1} = K + 1
set θ̄^{K+1} = α^{K+1} + ⟨β^{K+1}, x̄^{K+1}⟩
/* Test cut (K+1) against previously visited points.           */
for k = 1, 2, ..., K do
    set y = α^{K+1} + ⟨β^{K+1}, x̄^k⟩
    if y > θ̄^k then
        set d^{K+1} = d^{K+1} + 1
        set d^{j^k} = d^{j^k} − 1
        set j^k = K + 1
        set θ̄^k = y
    end
end
/* Test old cuts against new state.                            */
for k = 1, 2, ..., K do
    set y = α^k + ⟨β^k, x̄^{K+1}⟩
    if y > θ̄^{K+1} then
        set d^k = d^k + 1
        set d^{j^{K+1}} = d^{j^{K+1}} − 1
        set j^{K+1} = k
        set θ̄^{K+1} = y
    end
end
```

prevent it from deleting constraints or modifying constraint coefficients (the specifics of these limitations should not concern the reader). The approach we take in `SDDP.jl` is to add each newly discovered constraint, and then periodically rebuild each subproblem using a subset of the cuts chosen using the cut selection heuristic. Since this implementation includes a complete rebuild of the model, there is a trade-off between the reduction in solution time as a result of solving a model with fewer constraints and the fixed cost of rebuilding the model. Tuning this "rebuild-frequency" parameter is problem-specific but can result in an order of magnitude decrease in solution time [115].

Finally, although we have no experience with modifying the coefficients in-place, we have anecdotal evidence (J. Dias Garcia, personal communication, June, 2017) that the modification in-place strategy is more efficient than deleting and adding constraints. This strategy is also used in `DOASA` [158], a C++ implementation of SDDP that we benchmark `SDDP.jl` against in the next chapter. If technically feasible, we believe this technique should be implemented and tested in any new SDDP implementation.

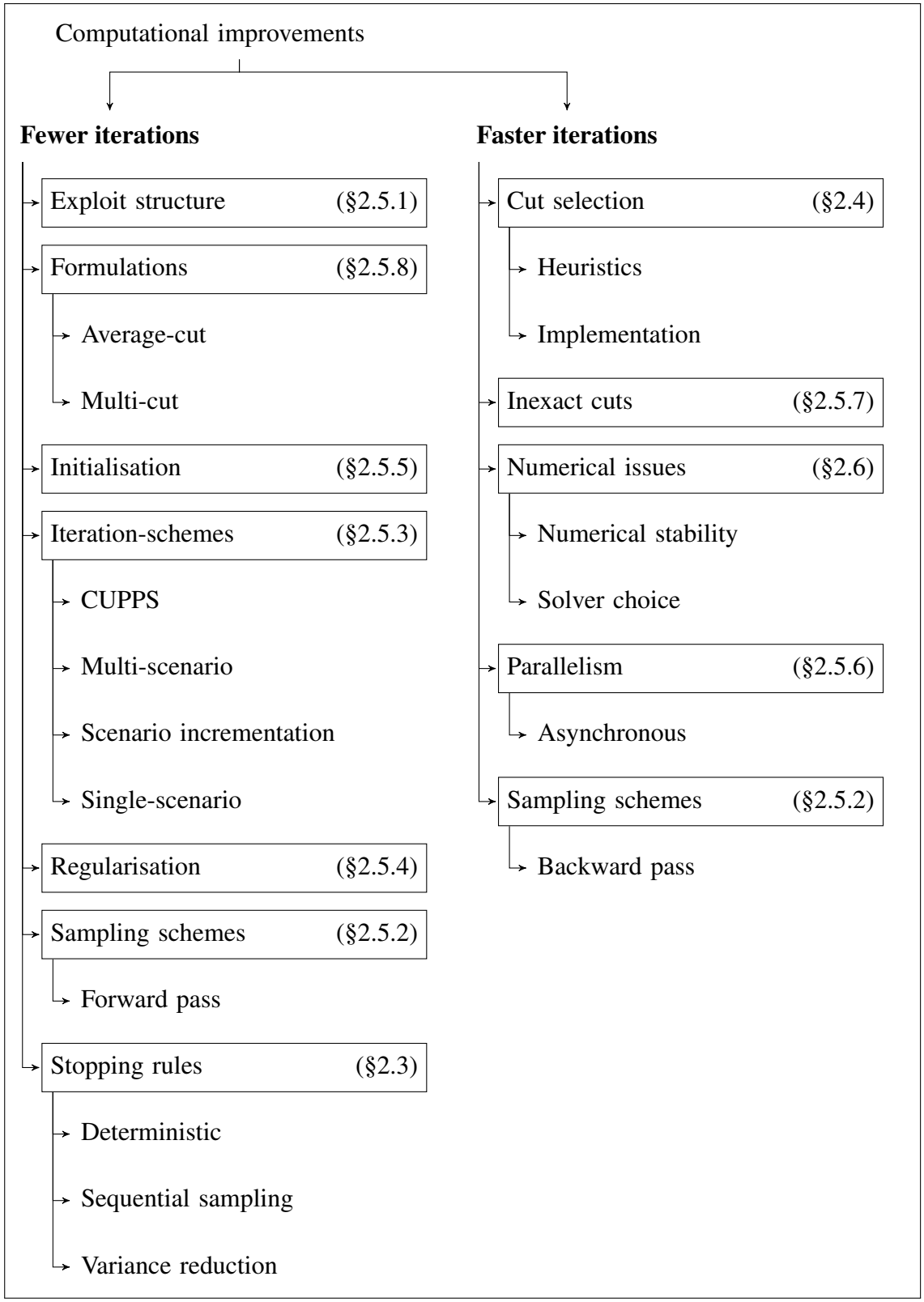


Figure 2.2: Organisation of computational improvements with section references.

2.5 Other computational improvements

In addition to cut selection and variance reduction techniques, various computational improvements have been suggested. Broadly speaking, these improvements can be clustered into two main types: improvements that result in *faster* iterations, and improvements that result in *fewer* iterations. In Figure 2.2, we present a chart that categorises the computational improvements suggested in the literature. We have already discussed stopping rules (Section 2.3) and cut selection (Section 2.4). In this section, we review the other improvements.

2.5.1 Exploit structure

It is possible to exploit structure in the policy graph on the backward pass. For example, assume subproblem i_1 is sampled on the forward pass, and there exists another subproblem i_2 such that $i_1^+ = i_2^+$. (This property is common in Markovian policy graphs.)

On the backward pass, we use the cut calculation algorithm (Algorithm 2) to derive a valid cut to add to the subproblem i_1. This involves solving $\mathbf{SP}_j^K(\hat{x}, \omega_j)$ for all $j \in i_1^+$ and $\omega_j \in \Omega_j$, and then computing ξ so that:

$$\mathbb{E}_{\xi}\left[\bar{V}_{j,\omega_j}^K\right] = \mathbb{F}_{j\in i_1^+;\, \omega_j\in\Omega_j}\left[\bar{V}_{j,\omega_j}^K\right].$$

However, if $i_1^+ = i_2^+$, then in doing so, we have also solved $\mathbf{SP}_j^K(\hat{x}, \omega_j)$ for all $j \in i_2^+$ and $\omega_j \in \Omega_j$, and obtained values for $\bar{V}_{j,\omega_j}^K$ and $\bar{\lambda}_{j,\omega_j}^K$. Therefore, we can derive a valid cut for subproblem i_2 by computing the new changed probability distribution $\{\xi_k\}$ such that:

$$\mathbb{E}_{\xi}\left[\bar{V}_{j,\omega_j}^K\right] = \mathbb{F}_{j\in i_2^+;\, \omega_j\in\Omega_j}\left[\bar{V}_{j,\omega_j}^K\right],$$

without having to re-solve the subproblems. Compared with the cost of solving the subproblems, computing this changed probability distribution is cheap, and we obtain an additional cut with minimal computational effort.

2.5.2 Sampling schemes

Forward pass

If we sample the forward pass using the true probabilities, then, in an out-of-sample simulation, we tend to get a good approximation of the cost-to-go function at the points we visit often and a bad approximation at the points we visit rarely. To overcome this problem, Guan [91] suggests that there is no need to sample trajectories on the forward pass with their true probability (so long as when the cuts are added on the backward pass the original probability distribution is used). With this technique, we adjust the probabilities on the forward pass so that we visit rare states more often than we otherwise would.

Backward pass

If the noise ω_i in each subproblem i only appears in the right-hand side of the constraints, then solving a subproblem with any realisation of the noise in the backward pass of the SDDP algorithm will result in a cut that forms a valid outer approximation. This is because the optimal dual solution λ^* for the primal problem:

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \quad [\lambda], \end{aligned}$$

is feasible (but sub-optimal) for the dual problem:

$$\begin{aligned} \max \quad & \lambda^\top b \\ \text{s.t.} \quad & A^\top \lambda = c \quad [x], \end{aligned}$$

with any value of b. Philpott and Guan [161] use this fact to show that on the backward pass, it is not necessary to sample all $\omega_j \in \Omega_j$. The algorithm will still converge, provided the sampling method satisfies their *Backward Pass Sampling Property*[2].

2.5.3 Iteration-schemes

An *iteration-scheme* defines the sequence of forward and backward passes that are conducted in the SDDP algorithm. For a given model, different iteration-schemes will have different rates of convergence. In the following, we detail four different iteration-schemes. However, it is possible to create other, hybrid, iteration-schemes based on combinations of those presented below.

Single-scenario

In the SDDP algorithm, as previously described, each iteration consists of a single forward pass, followed by a single backward pass along the sequence of nodes visited. This is the *single-scenario* iteration-scheme described by Philpott and Guan [161]. A schematic is shown in Figure 2.3 for two iterations. The numbers on the arcs denote the order in which the steps are taken.

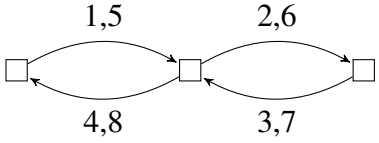


Figure 2.3: Schematic of the single-scenario iteration-scheme.

[2]The Backward Pass Sampling Property states we sample all of the noise realisations ω_j infinitely many times with probability 1 in the backward pass.

Multi-scenario

One iteration in the original SDDP algorithm (i.e. of Pereira and Pinto [153]) consists of 200 forward passes, followed by a backward pass along each of the forward passes. Philpott and Guan [161] called this the *multi-scenario* iteration-scheme. A schematic is shown in Figure 2.4.

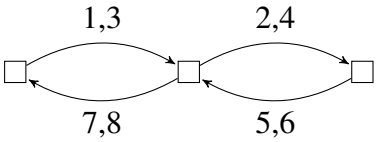


Figure 2.4: Schematic of the multi-scenario iteration-scheme.

CUPPS

Another scheme is proposed by Chen and Powell [42] in their CUPPS algorithm: instead of having separate forward and backward passes, the CUPPS algorithm combines the two by simulating a node, and then adding a cut before it transitions to the next node. A schematic is shown in Figure 2.5.

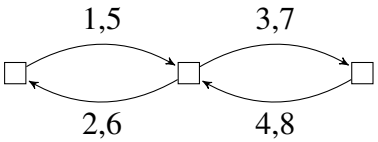


Figure 2.5: Schematic of the CUPPS iteration-scheme.

Scenario incrementation

As a trade-off between the single-scenario and multi-scenario iteration-schemes, de Matos et al. [55] propose a method of *scenario incrementation*. They suggest that as the algorithm proceeds, the number of forward passes per iterations should increase. In the computational benchmarks they performed, this strategy was shown to out-perform both the single-scenario and multi-scenario iteration-schemes. However, de Matos et al. [55] did not compare their method with a CUPPS style iteration-scheme.

2.5.4 Regularisation

Initial iterations of the SDDP algorithm have the tendency to jump between solutions in very different parts of the state-space due to a poor approximation of the cost-to-go function. As an example, consider a problem with a single state variable. In the first iteration, a cut is added with a negative dual, and the subproblem optimises towards the upper bound of the state variable. Then, in the second iteration, a cut with a positive dual is added, and the subproblem optimises

towards the lower bound of the state variable. To overcome this problem, various authors (e.g. [8, 96, 198]) augment the objective function of subproblem i in iteration k of the forward pass with a quadratic regularisation term:

$$\min\ C_i(x_i, u_i, \omega_i) + \rho_i^k (x_i' - \hat{x}_i^k)^\top Q_i (x_i' - \hat{x}_i^k),$$

where $\hat{x}_i^k$ is the centre of the regularisation term that may vary between iterations, and ρ_i^k belongs to the sequence of coefficients $\rho_i^1, \rho_i^2, \ldots$, such that $\rho_i^k \geq 0\ \forall k \in \mathbb{N}$, and $\lim_{k\to\infty} \rho_i^k = 0$. Q_i is a diagonal matrix with non-negative coefficients that is used to scale the penalties on the different state variables. On the backward pass, the original objective function is used.

Various authors (i.e. [8, 96, 198]) report promising results, but the performance of the method is likely to be problem-specific and dependent upon the choice of ρ_i^k and $\hat{x}_i^k$. We conjecture that this method will work well if there is a strong temporal component to the optimal state trajectory. This is often observed in hydro-thermal scheduling problems, where the reservoir levels (i.e. the states) follow a strong temporal pattern over the year [157]. Therefore, regularising the state along this trajectory is likely to improve the rate of convergence. In contrast, the price-state models we explore in Chapter 9 of this thesis do not follow a strong temporal pattern. Instead, the state evolves according to an auto-regressive process. Therefore, regularising the state to a previously visited state may hinder convergence.

2.5.5 Initialisation

Recall that each subproblem approximates the cost-to-go function via a set of cuts. However, in the first iteration before any cuts are generated, we bound the cost-to-go variable θ_i in each subproblem by some known bound (or a large negative value if we are unsure of a bound). The goal of an initialisation phase is to cheaply construct a better approximation of the cost-to-go function than assuming $\theta_i \geq -M$.

If the noise ω_i in each subproblem i only appears in the right-hand side of the constraints, then solving the expected-value subproblems (i.e. $\mathbf{SP}_i^k(x_i, \mathbb{E}[\omega_i])$) in the SDDP algorithm will result in cuts that form a valid outer approximation. This is because the optimal dual solution λ^* for the primal problem:

$$\begin{aligned} \min\quad & c^\top x \\ \text{s.t.}\quad & Ax = b \quad [\lambda], \end{aligned}$$

is feasible (but sub-optimal) for the dual problem:

$$\begin{aligned} \max\quad & \lambda^\top b \\ \text{s.t.}\quad & A^\top \lambda = c \quad [x], \end{aligned}$$

with any value of b.

Since we only need to solve one realisation of the noise per subproblem on the backward

pass instead of $|\Omega_i|$, each iteration of the expected-value problem will be faster than solving the full problem, at the expense of discovering cuts that are sub-optimal.

After solving a given number of these *expected-value* iterations, we can revert to the original SDDP algorithm using the discovered cuts as a good initial approximation for the cost-to-go function. Some authors [58, 154] report good results with this technique, but the method is limited to subproblems where the noise only appears in the right-hand side of the constraints, and the computational benefit will depend upon the number of possible realisations for the noise in each subproblem.

2.5.6 Parallelism

The SDDP algorithm is highly parallelisable. This observation dates back to Pereira and Pinto [153] who noted that the algorithm could be conducted asynchronously, with processes performing iterations independently and sharing cuts periodically. Different authors [103, 163] have proposed different methods for efficiently parallelising the algorithm. However, we choose to outline an approach that minimises inter-process communication. We call this *asynchronous* SDDP.

In our implementation, one process is designated the "master" process and the remainder are designated as "slaves". Each slave receives a full copy of the SDDP model and is set to work, performing iterations. At the end of each iteration, the slave passes the master the cuts it discovered during the iteration and receives any new cuts discovered by other slaves. The slave also queries the master as to whether it should terminate, perform another iteration, or perform a simulation. If the master requests a simulation (for example, to calculate a confidence interval in order to test for convergence), the slave returns the objective value of the simulation rather than a new set of cuts. A diagram of the process layout is shown in Figure 2.6. We benchmark an implementation of this approach in Section 3.3.3.

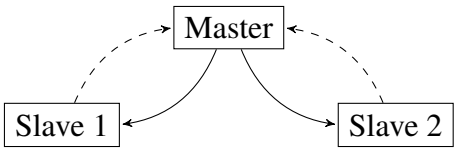


Figure 2.6: Structure of processes in asynchronous mode showing slaves passing new cuts to the master (dashed arrows) and the master passing cuts to slaves (solid arrows).

2.5.7 Inexact cuts

We explained earlier how it is not always necessary to solve all of the subproblems in the backward pass since any feasible dual can be used to construct a valid cut (at the expense of being sub-optimal). Some authors [94, 210] have expanded upon this idea and proposed that we terminate the solution to subproblems on the backward pass when a feasible, but sub-optimal,

solution is reached. This results in cuts that are *inexact*. This approach results in faster iterations (since less time is spent solving the subproblems to optimality) but may require more iterations before convergence. Therefore, the performance improvement is likely to be problem-specific.

2.5.8 Average-cut vs. multi-cut

In the average-cut formulation of SDDP (i.e. the one we have described so far), we replace the cost-to-go function with a single variable θ_i and add cuts that are the risk-adjusted expectation of the next stage's outcomes. In contrast, the multi-cut version of SDDP adds a cost-to-go variable θ_i^{j,ω_j} for every possible realisation in the next stage, and the expectation is calculated in the objective rather than in the cut coefficients. (See [11, 29, 212] for details.) Therefore, the approximated subproblem i after K iterations is:

$$\begin{aligned} V_i^K(x_i, \omega_i) = \min_{u_i} \quad & C_i(\bar{x}_i, u_i, \omega_i) + \sum_{j \in i^+} \phi_{i,j} \sum_{\omega_j \in \Omega_j} p_{\omega_j} \theta_{i,j,\omega_j} \\ \text{s.t.} \quad & x_i' = T_i(\bar{x}_i, u_i, \omega_i) \\ & \bar{x}_i = x_i, \quad [\lambda_i] \\ & u_i \in U_i(\bar{x}_i, \omega_i) \\ & \theta_{i,j,\omega_j} \geq \alpha_{i,j,\omega_j}^k + \langle \beta_{i,j,\omega_j}^k,\ x_i' \rangle, \quad j \in i^+,\ \omega_j \in \Omega_j,\ k \in \{1, 2, \ldots, K\}, \end{aligned}$$

where $\phi_{i,j}$ is the probability of transitioning from node i to node j in the policy graph and p_{ω_j} is the probability of sampling ω_j from Ω_j. The forward pass of the algorithm is identical to the average-cut SDDP. However, on the backward pass, we add a cut for every realisation of the next subproblem and noise. As a consequence, risk is also handled differently in the multi-cut formulation compared to the average-cut formulation. Instead of the "change-of-probability" approach, risk can be modelled using auxiliary variables. (See [109, 189, 197, 212] for details.)

The biggest draw-back to the multi-cut method is that, compared with the average-cut formulation, it involves adding many more variables and constraints to each subproblem. However, cut selection techniques could be applied to each cost-to-go variable independently in order to reduce this problem [11]. Despite the larger subproblems, some authors (e.g. Zhang et al. [212]) have reported that the multi-cut version is faster than the average-cut version. This is because the bound on the cost-to-go function using the multi-cut formulation is tighter than the average-cut formulation. For example, the cost-to-go function at point x_i' in the multi-cut formulation is:

$$\text{multi-cut}(x_i') = \sum_{j \in i^+} \left[\phi_{i,j} \sum_{\omega_j \in \Omega_j} \left[p_{\omega_j} \max_{k=1,2,\ldots,K} \left\{ \alpha_{i,j,\omega_j}^k + \langle \beta_{i,j,\omega_j}^k,\ x_i' \rangle \right\} \right] \right].$$

In contrast, the value of the cost-to-go function at the point x_i' in the average-cut formulation is:

$$\text{average-cut}(x_i') = \max_{k=1,2,\ldots,K} \left\{ \sum_{j \in i^+} \left[\phi_{i,j} \sum_{\omega_j \in \Omega_j} \left[p_{\omega_j} \left(\alpha_{i,j,\omega_j}^k + \langle \beta_{i,j,\omega_j}^k,\ x_i' \rangle \right) \right] \right] \right\}.$$

Therefore, it is easy to show that the inequality:

$$\text{multi-cut}(x'_i) \geq \text{average-cut}(x'_i)$$

holds for all x'_i. At the point in the state-space where the cuts were produced, the inequality above holds as an equality. The performance of the average-cut formulation compared with the multi-cut formulation depends on the relative trade-off between the gap in this inequality and the number of additional variables and constraints added to each subproblem.

2.6 Numerical issues

Solving a real-world model via SDDP typically involves solving hundreds of thousands or even millions of subproblems. (The models we solve in Chapter 9 involve the solution of over 200 million LPs!) In addition, these subproblems, and the SDDP solution process, have unique characteristics that challenge commercial solvers such as CPLEX [112] and Gurobi [98]. As a result, it is not uncommon to find: solution times that differ by 50% between these two state-of-the-art commercial solvers (we observe this in Chapter 3); feasible models incorrectly classified as infeasible or unbounded; and in extreme cases, incorrect solutions. In this section, we describe some of the numerical issues associated with SDDP and review potential solutions. We can also recommend the work of Klotz [118] as a comprehensive treatment of the issues associated with numerical instability in mathematical optimisation.

2.6.1 Scaling

The root cause of nearly all numerical issues in mathematical optimisation is poor problem scaling [68, 118]. Therefore, users should attempt to scale their problems to minimise the difference in the order of magnitude in the problem coefficients (paying particular attention to the cut coefficients). A common example is in the hydroelectricity setting, where it might be common to state that a reservoir has a capacity on the order of 10^9m^3 of water and a cost per unit on the order of \$$10^{-2}$/m^3 (a difference of 11 orders of magnitude). If we instead re-scale the problem to consider units of million m^3 (Mm^3), we now have a reservoir capacity on the order of 10^3Mm^3 and a cost per unit on the order of \$$10^4$/Mm^3. This has a difference of only one order of magnitude.

2.6.2 Tolerances

A long-standing issue in computer science is how to compare two values in floating-point arithmetic [118]. Typically, two values a and b are said to be equal if $|a - b| \leq \epsilon$ where ϵ is some small positive tolerance. In the optimisation context, these tolerances are applied to ensure: that constraints are satisfied; that a variable value is within its bounds; and that integer variables

are sufficiently close to an integer value [68]. Therefore, the constraint $x \geq 0$ can really be thought of as $x \geq -\epsilon$. In the case of most linear solvers (e.g. Gurobi [98]), it seems the default is $\epsilon = 10^{-6}$ [118].

Although this is a perfectly acceptable solution in general optimisation problems, numerical tolerances can lead to counter-intuitive outcomes when the solution to one problem is used as an input to a subsequent model. We demonstrate this in the following example.

Consider a simple linear program with two variables and three constraints:

$$\begin{aligned} \min \quad & x + y \\ \text{s.t.} \quad & 10x - y \geq 0 \\ & x \geq 0 \\ & y \geq 0. \end{aligned} \tag{2.5}$$

We can decompose the problem using Benders decomposition into the first-stage problem:

$$\begin{aligned} \min \quad & x + \theta \\ \text{s.t.} \quad & x \geq 0 \\ & \theta \geq 0, \end{aligned} \tag{2.6}$$

and the second-stage subproblem:

$$\begin{aligned} \min \quad & y \\ \text{s.t.} \quad & y \leq 10\hat{x} \quad [\lambda] \\ & y \geq 0, \end{aligned} \tag{2.7}$$

where $\hat{x}$ is the optimal value of x in the first stage problem. The question for the reader is: is it possible for a feasible first-stage (i.e. Eq. 2.6) solution of x to cause infeasibility when set as the right-hand side $\hat{x}$ in the second-stage problem (i.e. Eq. 2.7)?

The intuitive answer is no. The second-stage problem has constraints such that $0 \leq y \leq 10x$, and we know from the first-stage problem that $x \geq 0$. Therefore, the feasible set for y is never empty as it always contains the value $y = 0$ for any feasible value for $x \geq 0$. However, given a floating-point tolerance of $\epsilon = 10^{-6}$ in Eq. 2.6, the solver may produce a (numerically) feasible and optimal solution to Eq. 2.6 of $(x, \theta) = (-10^{-6}, 0)$. This can (and should) be expected by the user. An issue arises when the approximately optimal value for x is set as the right-hand side value $\hat{x}$ in Eq. 2.7. This produces a set of constraints such that:

$$-\epsilon \leq y \leq -10^{-5} + \epsilon.$$

As a result, Eq. 2.7 is infeasible for $\epsilon < 5 \times 10^{-6}$. Since the default tolerance is $\epsilon = 10^{-6}$, it can be the case that a (numerically) feasible solution to the Benders first-stage problem leads to an infeasible Benders second-stage subproblem.

We (and others, e.g. [8, 115]) have frequently observed this problem in different real-world models, using a variety of different solvers, and on different machines. However, there is a simple mitigation. The solution is to modify the forward pass of the SDDP algorithm (Algorithm 3) so that instead of passing the solution of x'_i forward as x_j, we first round the variable back onto its domain (if it has finite bounds and the numerical solution is outside these bounds). Although this is not guaranteed to fix the problem in all instances (it is possible to construct a much more elaborate set of constraints that will result in infeasibility for a numerically optimal first-stage variable that is not at one of its bounds), in our experience, it alleviates a significant cause of numerical instability.

2.6.3 Solver choice

A critical component of the numerical stability of the SDDP algorithm is the choice of solver. As will be seen in Chapter 3, the variance in solution time between commercial solvers such as Gurobi [98] and CPLEX [112] on the same problem can be in excess of 50%. Moreover, due to the scaling and tolerance issues described above, it is not uncommon to find that solvers encounter numerical difficulties that prevent them from finding the optimal solution. In our experience, commercial solvers such as Gurobi and CPLEX are more robust than open-source solvers, such as Clp [43] which frequently fails to find the optimal solution. However, even the commercial solvers are not immune to numerical error. Gurobi will occasionally return a status `Numeric`, which indicates that it has encountered un-recoverable numerical difficulties, and CPLEX will return the status `CPX_OPT_INFEAS`, which indicates that it solved a scaled version of the primal problem to optimality, but when the result was un-scaled, the solution was infeasible. We have even found bugs in Gurobi that result in sub-optimal solutions being returned to the user but reported as optimal. (See Appendix B for details.)

Considering these issues, we highly recommend that users solve their model using different solvers and parameters in order to compare solution times and validate the solutions. Without performing this step when solving a new real-world model, users should be wary of the solution they obtain.

2.6.4 Potential solutions

Finally, we note that numerical issues in optimisation are not limited to SDDP. Modern MIP solvers such as Gurobi already include a number of techniques for counteracting numerical issues [136]. However, many of these techniques are designed to deal with the cutting planes that are added as part of the solution process for integer programs. Therefore, they are not used when solving linear programs. Some of these techniques include discarding redundant cuts (e.g. cut selection – Section 2.4), as well as discarding cuts whose coefficients differ significantly in magnitude [136]. Many of these techniques are unexplored in the context of SDDP.

2.7 Multiple optimal solutions

In some cases, the subproblems in an SDDP model have multiple optimal primal solutions. For example, in hydro-thermal scheduling (a common application area), the marginal value of water (i.e. the slope of the cost-to-go function) is sometimes equal to the marginal cost of thermal fuel [115]. Therefore, the optimal solution can be any combination of thermal and hydro generation [157]. This phenomenon of multiple optimal solutions occurs because the dual of the subproblem is degenerate. Furthermore, recall that SDDP is based on Benders decomposition. Therefore, problems that affect Benders decomposition will also affect SDDP.

In this section, we illustrate how multiple optimal primal solutions of a Benders first-stage problem can lead to a sub-optimal solution when the policy is simulated, even after the algorithm has terminated due to an optimality criterion. This issue has serious implications for all practitioners implementing Benders decomposition, but in particular, those implementing nested Benders decomposition in algorithms such as SDDP.

Let us consider a simple linear program that demonstrates multiple optimal solutions when solved via Benders decomposition:

$$\begin{aligned}
\min\ & x + & & & y_2 & & \\
\text{s.t. } & x & & & & \leq & 1 \\
& x - & y_1 & & & = & 0 \\
& & y_1 & + & y_2 & \geq & 0 \\
& - & 3y_1 & + & y_2 & \geq & -2 \\
& x & & & & \geq & 0.
\end{aligned} \tag{2.8}$$

It is apparent upon inspection that Eq. 2.8 has the optimal solution $(x, y_1, y_2) = (0, 0, 0)$, with the corresponding objective value $x + y_2 = 0$. However, we can decompose problem Eq. 2.8 using Benders decomposition into the first-stage problem (assuming a known lower bound on the second-stage problem of -1):

$$\begin{aligned}
\min\ & x + \theta & & \\
\text{s.t. } & x & \leq & 1 \\
& \theta & \geq & -1 \\
& x & \geq & 0,
\end{aligned} \tag{2.9}$$

and the second-stage problem:

$$\begin{aligned}
\bar{\theta} = \min\ & & y_2 & & & \\
\text{s.t. } & y_1 & & = & \hat{x} & [\lambda] \\
& y_1 + & y_2 & \geq & 0 & \\
& -3y_1 + & y_2 & \geq & -2. &
\end{aligned} \tag{2.10}$$

We begin by solving the first-stage problem (Eq. 2.9). This has the unique solution $(x, \theta) = (0, -1)$. Next, recall that in Benders decomposition, a valid lower bound for the objective of the original problem (Eq. 2.8) is the objective of the first-stage problem. Therefore, the current lower bound is -1. Then we solve the second-stage problem (Eq. 2.10). This has the unique solution $(y_1, y_2, \lambda) = (0, 0, -1)$, and an objective value of $\bar{\theta} = 0$.

Using the solutions to these two subproblems, we can obtain a feasible solution to Eq. 2.8 by combining the optimal value of x in the first-stage problem, and the optimal values of y_1 and y_2 in the second-stage problem. Therefore, we obtain the feasible solution $(x, y_1, y_2) = (0, 0, 0)$. As observed earlier, this is the optimal solution to Eq. 2.8.

We can also calculate an upper bound for Eq. 2.8 by summing the optimal objective value of the first-stage problem (less the optimal value of the cost-to-go variable θ) and the optimal objective value of the second-stage problem. This gives an upper bound for the original problem of $x + y_2 = 0 + 0 = 0$. Despite having found the optimal solution, we have not proved optimality as the lower bound (-1) is not equal to the upper bound (0).

Therefore, we add a Benders optimality cut to the first-stage problem. These cuts have the form:

$$\theta \geq \bar{\theta}^k + \bar{\lambda}^k(x - \hat{x}^k),$$

where $\bar{\theta}^k$ is the objective of the second-stage problem, and $\bar{x}^k$ is the value x takes in the optimal solution to the first-stage problem in the k^{th} iteration. We therefore add the cut:

$$\theta \geq 0 + -1(x - 0).$$

The first-stage subproblem is now:

$$\begin{array}{lrcrcr} \min & x & + & \theta & & \\ \text{s.t.} & x & & & \leq & 1 \\ & & & \theta & \geq & -1 \\ & x & + & \theta & \geq & 0 \\ & x & & & \geq & 0. \end{array} \tag{2.11}$$

There are multiple optimal primal solutions to this problem as any solution:

$$(x, \theta) \in \{(a, -a) : \ a \in [0, 1]\}$$

is an optimal solution to the primal first-stage problem with an objective value of 0.

We start a new Benders iteration by re-solving the new first-stage problem. This has a non-unique solution; however, if we assume that the solver hot-starts from the previous solution value, we obtain the solution $(x, \theta) = (0, 0)$. The objective value of the subproblem is 0, which is therefore also a valid lower bound for the original problem.

As before, we solve the second-stage problem and obtain the unique solution $(y_1, y_2, \lambda) =$

$(0, 0, -1)$, and an objective value of $\bar{\theta} = 0$. From this, we calculate an upper bound for the original problem of 0, which is also equal to the lower bound. Therefore, the Benders decomposition algorithm terminates with the optimal solution $(x, y_1, y_2) = (0, 0, 0)$ and the optimal objective value of 0.

Now assume that we take the converged problem and attempt to reconstruct the solution. We solve the first-stage problem, but this time the solver might return the (optimal) solution $(x, \theta) = (1, -1)$. The different solution may result from the use of a different solver, different solve parameters, different hot-start solutions, numerical error, or the use of a non-deterministic solution algorithm. The solution is also feasible and provides a lower bound to the original problem that is equal to the optimal objective value (as expected). Therefore, we set $\hat{x} = 1$ and solve the second-stage problem. This has the optimal solution $(y_1, y_2, \lambda) = (1, 1, 1)$.

By combining the solutions of the two stage problems, we obtain a feasible solution to the original problem of $(x, y_1, y_2) = (1, 1, 1)$. This has an objective value of 2 and is sub-optimal.

In other words, taking an optimal first-stage decision in a converged Benders decomposition can lead to a sub-optimal solution in the second-stage problem. In this example, it is trivial to observe that the solution is sub-optimal. In a more complicated setting such as SDDP, it may not be so easy to observe. This has a serious implication for SDDP and we cannot understate its importance.

Remark 1. *When we simulate an "optimal" policy, we may obtain a sequence of sub-optimal controls, even in a converged, deterministic model.*

As a consequence, the decision-rule at each node i should be considered to be a random variable. The Benders optimality cuts alone (as represented by their variable coefficients and intercept) are not sufficient to reproduce the optimal control given an incoming state variable x_i and realisation of the noise ω_i. This conflicts with the widespread belief that it is possible to pre-compute the cuts and then use them as an operational policy in the future. (For examples of this, see Philpott and Pritchard [158] and Asamov and Powell [8]. We highlight these papers to demonstrate the belief, rather than as a comment on any of the individual papers.) Moreover, authors such as Liu et al. [128], Rougé and Tilmant [183], and Kapelevich [115] have observed this problem in real-world instances. Rougé and Tilmant [183] called the tendency of the simulated realisations of the policy produced by the SDDP algorithm to have unstable and unpredictable behaviour with small variations in the input "algorithmic chaos".

Degeneracy in other settings

Dual degeneracy is also found in other cutting plane methods such as Gomory's fractional cutting plane method for solving integer linear programs [9, 211]. As a solution, Zanette et al. [211] propose a lexicographic simplex method for solving the linear subproblems. Their method ensures that the same optimal basis is obtained each time the algorithm is run, ensuring that we

always obtain the same optimal solution. They found the method to be effective and stable in practice.

Degeneracy is also discussed in the context of Benders decomposition in a review paper by Rahmaniani et al. [172]. However, the literature is focused on the problem of choosing a strong cut from a set of feasible cuts when the primal is degenerate (i.e. there are multiple dual solutions). Aside from the papers mentioned above (i.e. [115, 128, 183]), we are unaware of any work that recognises that simulating a converged policy can result in a sub-optimal sequence of controls. (It is also interesting to note that few of the computational techniques for dealing with primal degeneracy have flowed through into the SDDP literature; however, it is an area that we leave to future research.)

2.8 Summary

In this chapter we introduced SDDP, the main algorithm that we use in this thesis to solve multistage stochastic optimisation problems. In particular, we highlighted four technical assumptions that restrict the class of multistage stochastic optimisation problems that SDDP can solve. We also conducted a review of the current literature that represents the state-of-the-art for SDDP implementations. A summary of the computational improvements is presented in Figure 2.2. In the next chapter we introduce `SDDP.jl`, a software package that we have developed to solve multistage stochastic optimisation problems using SDDP.

Chapter 3

SDDP.jl: a Julia package for stochastic dual dynamic programming

In the previous chapter, we introduced stochastic dual dynamic programming (SDDP) as a solution technique for solving multistage stochastic optimisation problems. In this chapter we present `SDDP.jl`, an open-source library we have developed that implements the SDDP algorithm. `SDDP.jl` is built upon JuMP, an algebraic modelling language in Julia. This enables `SDDP.jl` to have a high-level interface for the user, while simultaneously providing performance that is similar to implementations in low-level languages. We benchmark the performance of `SDDP.jl` against a C++ implementation of SDDP for the New Zealand Hydro-Thermal Scheduling Problem. On the benchmark problem, `SDDP.jl` is approximately 30% slower than the C++ implementation. However, this performance penalty is small when viewed in context of the generic nature of the `SDDP.jl` library compared with the single-+purpose C++ implementation.

3.1 Introduction

Solving any mathematical optimisation problem requires four steps: the formulation of the problem by the user; the communication of the problem to the computer; the efficient computational solution of the problem; and the communication of the computational solution back to the user. Over time, considerable effort has been made to improve each of these four steps for a variety of problem classes. For example, consider the evolution from early file formats such as MPS [140] to modern algebraic modelling languages embedded in high-level languages such as JuMP [66], or the 73-fold speed-up in solving difficult mixed-integer linear programs in seven years by Gurobi [99].

However, the same cannot be said for stochastic problem classes. This is particularly true of multistage stochastic optimisation problems, which are the focus of this thesis. As discussed in Chapter 1, there is considerable debate about how to best formulate a stochastic program [165,

166]. Moreover, when it comes to communicating the problem to the solver, various file formats have been proposed [31, 81], but owing to the lack of agreement about the problem formulation, acceptance of these is not widespread. Instead, what happens is the development of an *ad-hoc*, problem-specific format that is often tightly coupled to individual solver implementations on a case-by-case basis. The visualisation of a stochastic policy is also difficult due to the high-dimensionality of the state- and control-spaces and the inherent uncertainty. Thus, policy visualisation is also problem-specific.

Where progress has been made however, is in the development of efficient computational solution algorithms. The state-of-the-art solution technique for convex multistage stochastic optimisation problems is stochastic dual dynamic programming (SDDP), which we introduced in the previous chapter. Since the seminal work of Pereira and Pinto [153], SDDP (and its variants) has been widely used to solve a number of problems in both academia and industry. However, until recently, no open-source, generic implementations of the algorithm existed in the public domain. (We discuss these implementations in Section 3.5.) Instead, practitioners were forced to code their own implementations in a variety of languages and styles. Research implementations have been reported in a variety of languages, including AMPL [91], C++ [103, 158], GAMS [37, 149], Java [8], and MATLAB [151], as well as in commercial products such as the seminal SDDP [168], QUASAR [171], and DOASA [161]. In our opinion, this "re-invention of the wheel" has limited the adoption of the SDDP algorithm in areas outside the electricity industry (which is the focus of most researchers) since there is a large up-front cost to development. Moreover, many researchers develop and test new algorithmic improvements without being able to easily compare their ideas against other implementations or the current state-of-the-art.

This chapter presents `SDDP.jl` – a Julia package for solving multistage stochastic optimisation problems using SDDP. The chapter is laid out as follows. First, in Section 3.2, we introduce `SDDP.jl` and explain many of its basic features by working through a simple example. Then, in Section 3.3, we detail some of the more advanced features of `SDDP.jl`. In Section 3.4, we benchmark `SDDP.jl` against a C++ implementation of the SDDP algorithm for the New Zealand Hydro-Thermal Scheduling problem. Finally, we conclude with a comparison of `SDDP.jl` to other libraries in Section 3.5.

This chapter is not intended to be a comprehensive tutorial for `SDDP.jl`. Instead, readers are directed to `https://github.com/odow/SDDP.jl` for source-code, examples, and documentation.

3.2 Example: the air conditioner problem

To illustrate the basic features of `SDDP.jl`, we consider the air conditioner problem proposed by Papavasiliou [150]. In this problem, the manager of a factory seeks a production plan for producing air conditioners over a period of three months. During standard working hours, the

factory can produce 200 units per month at a cost of \$100/unit. Unlimited overtime can be scheduled; however, the cost increases to \$300/unit during those hours. In the first month, there is a known demand of 100 units. However, in each of months two and three, there is an equally likely demand of either 100 or 300 units. Air conditioners can be stored between months at a cost of \$50/unit, and all demand must be met.

We now walk through each of the four steps in the solution process, beginning with the problem formulation.

3.2.1 Formulating the problem

We can formulate the air conditioner problem as a multistage stochastic optimisation problem. It has three stages – one for each month of operation. This can be represented by the linear policy graph shown in Figure 3.1.

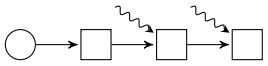


Figure 3.1: Policy graph of the air conditioner problem.

We now identify the states, controls, and noise terms in the model. There is one state variable in the model: x_t, the quantity of air conditioners in storage at the start of stage (i.e. month) t. As the policy graph is linear, x_t can also be thought of as the quantity of air conditioners in storage at the end of stage $t-1$. Therefore, we have $x'_t = x_{t+1}$ for $t = 1, 2$. There is one noise in the model: ω_t, the demand for air conditioners in month t. In addition, there are three control variables:

1. p_t: the number of air conditioners produced during standard working hours in month t;
2. o_t: the number of air conditioners produced during overtime in month t; and
3. s_t: the number of air conditioners sold[1] in month t.

We can formulate each subproblem in the policy graph indexed by $t = 1, 2, 3$ as:

$$\begin{aligned}
\mathbf{SP}_t: \quad V_t(x_t, \omega_t) = \max_{p_t, o_t, s_t, x_{t+1}} \quad & 100p_t + 300o_t + 50x_{t+1} + \mathbb{E}_{\omega_{t+1}}[V_{t+1}(x_{t+1}, \omega_{t+1})] \\
\text{s.t.} \quad & x_t + p_t + o_t - s_t = x_{t+1} \\
& s_t = \omega_t \\
& 0 \le p_t \le 200 \\
& x_{t+1}, o_t, s_t \ge 0,
\end{aligned}$$

where $V_4(\cdot, \cdot) = 0$. Although air conditioners are discrete, to maintain convexity for SDDP we solve the LP relaxation. However, as we shall later see, the solution is naturally integer. In each

[1] In the following formulation, the sales are trivially equal to the demand. We include an explicit control variable for the sales in order to demonstrate some aspects of `SDDP.jl`.

stage t, ω_t is independently sampled from a finite discrete distribution so that:

$$\omega_1 = \left\{100, \text{ with probability } 1,\right.$$

$$\omega_t = \begin{cases} 100, \text{ with probability } 0.5 \\ 300, \text{ with probability } 0.5 \end{cases} \quad \text{for } t = 2, 3.$$

The root node of the policy graph has the initial condition $x_1 = 0$. Therefore, the full optimisation problem faced by the factory manager is:

$$\max_{\pi} \quad \mathbb{E}_{\omega_1}\left[V_1(0, \omega_1)\right]. \tag{3.1}$$

In the next section, we describe how to communicate the formulation of the subproblems $\mathbf{SP}_t$ to `SDDP.jl`.

3.2.2 Communicating the problem to the solver

In order to introduce `SDDP.jl`, we must first introduce JuMP [66], an algebraic modelling language for the Julia programming language [28] that we use as the basis for creating and manipulating the subproblems in the SDDP algorithm. JuMP supports a wide range of problem classes including linear, mixed-integer, conic-quadratic, and nonlinear. In particular, a large effort has been placed on abstracting multiple steps in the typical optimisation modelling process in a way that is open to extension by third parties. This has allowed us to create an SDDP modelling library that builds on the functionality of both JuMP and Julia. The expressiveness of JuMP's modelling syntax is available to the user with minimal implementation effort, while Julia's multiple dispatch, parallelism, and macro-programming features enable a fast, yet generic, implementation of the SDDP algorithm.

Overview Before we dive into the specifics, we provide the reader with a brief overview of the design behind the user-interface of `SDDP.jl`. The driving principle is to represent each subproblem $\mathbf{SP}_t$ as a JuMP model parameterised by the incoming state variable x_t and the realisation of the stagewise-independent random variable ω_t, staying as close as possible to the formulation of $\mathbf{SP}_t$ described above. This is achieved by extending some of the features of JuMP to introduce state variables, and to allow random variables in the right-hand side of constraints and in the objective. Where new methods and macros have been added (for example, adding state variables), we have tried to stay close to the syntactic feel of JuMP. These modifications typically produce standard JuMP constructs that are visible to the user (for example, a state variable is just a JuMP variable), along with some additional SDDP specific information that is hidden from the user (for example, how to link state variables between subproblems). Behind the scenes, `SDDP.jl` handles the cost-to-go function in the objective, manages the realisations

of the random variables, and passes the values of the state variables between subproblems.

Finally, due to technical reasons, `SDDP.jl` requires three new assumptions in addition to assumptions (A1)–(A4) outlined in the previous chapter. These are:

(A5) for each node $i \in \mathcal{N}$, the noise ω_i only appears in the right-hand side of the constraints in the transition function T_i and feasibility set U_i, but *not* as a coefficient;
(A6) every subproblem $i \in \mathcal{N}$ is Hazard-Decision; and
(A7) the policy graph $\mathcal{G}$ is *Markovian*.[2]

We now describe how to communicate the formulation of the air conditioner subproblems to `SDDP.jl`. We give the complete code as an appendix item in Listing C.1. However, before we can run the example, we need to install `SDDP.jl` (this requires Julia 0.5 or later):

```
julia> Pkg.clone("https://github.com/odow/SDDP.jl")
```

The appendix example can be run by saving the file to the disk, opening a Julia REPL, and then running:

```
julia> include("path/to/airconditioner/file")
```

In the remainder of this section, we walk through the file and explain points of interest in Listing C.1.

Loading packages First, we load the relevant packages needed by the example: SDDP, JuMP, and Clp.

```
using SDDP, JuMP, Clp
```

Users are free to use any of the other solvers in the JuliaOpt ecosystem that support MathProgBase instead of Clp. (See Dunning et al. [66] for more details.)

The SDDPModel object The core type in `SDDP.jl` is the `SDDPModel` object. This stores a single JuMP model for each subproblem, as well as the information needed to parameterise each model based on the incoming state variable x_t and realisation of the random variable ω_t. It also stores other information, including the linkages between subproblems. If the policy graph is linear, then the constructor for the `SDDPModel` object has the following form:

```
m = SDDPModel(keyword arguments...) do sp, t
    ... subproblem definition ...
end
```

[2] More exotic policy graphs, while permissible from a theoretical perspective, are not yet solve-able using `SDDP.jl`.

There are two key features to discuss. First, we describe the `keyword arguments...`. There are many optional keyword arguments which we shall not discuss as they are not necessary for our simple example. However, the four given in Listing C.1 are required. They are: `stages`, the number of stages in the model; `objective_bound`, a known lower bound (if minimising) for every subproblem; `sense`, the optimisation sense `:Min` or `:Max`; and `solver`, any MathProgBase compatible solver.

Second, the unusual `do sp, t ... end` syntax is a Julia construct that is equivalent to writing:

```
function foo(sp::JuMP.Model, t::Int)
    ... subproblem definition ...
end
m = SDDPModel(foo, keyword arguments...)
```

where `foo(sp, t)` is a function that takes an empty JuMP model `sp` and the stage `t` and builds the subproblem $\mathbf{SP}_t$ using a mix of JuMP functionality and `SDDP.jl`-specific functions. We now describe how to construct the subproblem definition.

Control variables `sp` is a standard JuMP model. Therefore, the user is free to add any JuMP variables to the model via the `@variable` and `@variables` macros. In the air conditioner problem, we add: the number of units to produce during standard production hours `p`; the number of units to produce during overtime production `o`; and the number of air conditioners to sell `s`:

```
@variables(sp, begin
    0 <= p <= 200
         o >= 0
         s >= 0
end)
```

All these control variables are non-negative, and the standard production capacity `p` has an upper bound of 200 units.

State variables `SDDP.jl` provides a new macro called `@state` that can be used to add state variables to the subproblem `sp`. This macro is a variant of the JuMP `@variable` macro with a few differences. It takes three arguments. The first argument is the JuMP model `sp`. The second argument is an expression for the outgoing state variable (i.e. x_{t+1}) and can be any valid syntax for the second argument of the JuMP `@variable` macro (e.g. specifying upper and lower bounds). The third argument is for the incoming state variable (i.e. x_t) and must be a name for the incoming variable, followed by ==, and then the value of the state variable in the first stage (i.e. x_1). For the air conditioner problem, we can create the state variable x_t as:

```
@state(sp, xtp1 >= 0, xt == 0)
```

This will add two standard JuMP variables (`xtp1` and `xt`) to the model `sp` that are visible to the user, as well as some additional information that is hidden from the user describing how to link the value of `xtp1` (i.e. x_{t+1}) in stage t to `xt` (i.e. x_t) in stage $t+1$. The reader should note that the syntax of the third argument is how we specify the value of the state variable at the root node (i.e. x_R). It does not enforce the constraint $x_t = 0$ in all stages. The following is also valid syntax and demonstrates how the index i can be shared between `ytp1` and `yt`:[3]

```
y1 = [0.0, 1.0] # initial value
@state(sp, 0 <= ytp1[i=1:2] <= 1, yt == y1[i])
```

This code will add four variables to the JuMP model `sp`: `ytp1[1]`, `ytp1[2]`, `yt[1]`, and `yt[2]`. In constraints and in the objective (detailed below), state variables (i.e. `xt` and `xtp1`) behave just like any other JuMP variable.

Constraints The user is also free to add any arbitrary JuMP constraint via the `@constraint` macro. For the air conditioner problem, we add the balance constraint that the number of air conditioners in storage at the end of a month `xtp1` is equal to the number in storage at the start of the month `xt`, plus any production `p` and overtime production `o`, less any sales `s`:

```
@constraint(sp, xtp1 == xt + p + o - s)
```

Right-hand side uncertainty `SDDP.jl` supports random variables in the right-hand side of constraints.[4] Instead of using `@constraint` to add a constraint with a random variable, `SDDP.jl` provides the macro `@rhsnoise`. This macro takes three arguments. The first is the JuMP model `sp`. The second is a keyword argument of the form `name = realisations`, where `name` is an arbitrary name for the random variable and `realisations` is a vector containing the finite discrete list of realisations that the random variable can take. In the air conditioner problem, these realisations are the singleton 100 in stage $t = 1$, and either 100 or 300 in stages $t = 2$ and $t = 3$. The third argument is any valid JuMP `@constraint` syntax that contains `name` as part of the right-hand side term (i.e. not a variable coefficient). For the air conditioner problem, the agent must sell exactly the quantity of units demanded. Therefore, we add the `@rhsnoise` constraint:

```
D = [ [100], [100, 300], [100, 300] ]
@rhsnoise(sp, w=D[t], s == w)
```

[3] This construct will be familiar to users of JuMP.

[4] As noted before, `SDDP.jl` does *not* support uncertainty in the constraint coefficients. This is a known limitation and will hopefully be resolved in a future release.

`D[t]` is the list of possible demand realisations in stage t. (Note that we use `t` which was defined in the *SDDPModel Object* section.) We can set the probability of the demand realisations in stage t using the `setnoiseprobability!` function:

```
P = [ [1.0], [0.5, 0.5], [0.5, 0.5] ]
setnoiseprobability!(sp, P[t])
```

In this example, we used a list of lists to store the realisation and probability data in a compact manner. However, the following is also valid:

```
if t == 1
    @rhsnoise(sp, w=[100], s == w)
    setnoiseprobability!(sp, [1.0])
else
    @rhsnoise(sp, w=[100, 300], s == w)
    setnoiseprobability!(sp, [0.5, 0.5])
end
```

Readers should note that this use of an `if` statement can be used more generally to conditionally add stage-specific features to the subproblems.

The stage-objective All that remains is to define the objective of the subproblem. `SDDP.jl` handles the $\mathbb{F}[V_{t+1}(x_{t+1}, \omega_{t+1})]$ term behind the scenes, so the user only has to provide the stage-objective via the `@stageobjective` macro:

```
@stageobjective(sp, 100 * p + 300 * o + 50 * xtp1)
```

We now describe how to solve the model and visualise the solution.

3.2.3 Solving the problem

As we have previously mentioned, one area in which stochastic programming has made significant progress is in the efficient computational solution of the problem. `SDDP.jl` includes many of the state-of-the art features that have appeared in the literature. (We discussed these in the previous chapter.) In addition, we heavily utilise Julia's type-system and ability to overload generic methods in order to provide an interface that is extensible by the user without them having to modify the package's source-code. We briefly summarise three of the most important solution features of `SDDP.jl` but elaborate upon them in Section 3.3.

1. User-defined cut selection routines.

   Without modifying the `SDDP.jl` source-code, users are able to define new cut selection heuristics (see Section 2.4) to reduce the size of the subproblems.

2. User-defined risk measures.

   Without modifying the `SDDP.jl` source-code, users are able to define new risk measures (see Section 1.3) that seamlessly integrate with the entire `SDDP.jl` library. This functionality has been used by Philpott et al. [162] to develop a distributionally robust SDDP algorithm without having to implement the full SDDP algorithm.

3. Parallelism.

   `SDDP.jl` leverages Julia's built-in parallel functionality to perform SDDP iterations in parallel (see Section 2.5.6). `SDDP.jl` has been successfully scaled from the single core of a laptop to tens of cores in high-performance computing environments without the user needing to modify a single line of code.

Returning to the air conditioner problem, we can solve it using the SDDP algorithm via the `solve` method:

```
status = solve(m, max_iterations=10)
```

There are many different options that can be passed to the solve command, so in the interests of conciseness, we omit them here. However, the argument `max_iterations` causes the algorithm to terminate after the provided number of SDDP iterations have been conducted. Once the solution process has terminated, we can query the lower bound of the solution using the function `getbound(m)`. This returns the optimal objective bound (which can be verified by solving the deterministic equivalent) of $62,500. `status` is a symbol describing why the algorithm terminated. In this example, it will be `:max_iterations`.

3.2.4 Understanding the solution

Unlike deterministic mathematical programming, SDDP does not provide a solution containing an explicit optimal value for every variable. Instead, it constructs a *decision-rule* for each node of the policy graph in the form of an optimisation problem. To obtain the optimal control for a stage, the user sets the value of the incoming state variable x_t and the realisation of the random variable ω_t, then solves the approximated optimisation program. It can be difficult to understand the policy due to the uncertainty and the high dimensionality of the state- and control-spaces. Therefore, one commonly used approach is Monte Carlo simulation. In `SDDP.jl`, this can be done using the `simulate` function:

```
sims = simulate(m, 40, [:xtp1, :p, :o, :s])
```

The first argument is the `SDDPModel` `m`. The second argument is the number of iid replications to simulate (in this case, 40), and the third argument is a list of variables to record the value of at each stage. For the air conditioner problem, we record `xtp1`, the number of air conditioners

in storage at the end of stage t, as well as the three control variables: standard production `p`, overtime production `o`, and sales `s`. `sims` is a vector of dictionaries (one element for each simulation) that can be manipulated or saved to a file for later analysis. For example, the user can query the number of units produced during standard production hours during the second month in the tenth Monte Carlo replication by:

```
julia> sims[10][:p][2]
```

In Figure 3.2, we plot four of the Monte Carlo replications (chosen because they sample different demand realisations) for the four variables recorded by `simulate`. In all scenarios, 200 units are produced during normal production hours during the first stage (Figure 3.2b). This is despite the fact that the demand of 100 units is known ahead of time. Therefore, 100 units are in storage at the end of the first stage (Figure 3.2c). If the demand (Figure 3.2a) is high (i.e. 300 units) in the second stage, then production remains at 200 units per stage. In addition, 100 units are sold from storage to meet the demand of 300 units. If demand is low (i.e. 100 units) in the second stage, then standard production drops to 100 units and no units are sold from storage. In all scenarios, there is no overtime during the first two stages (Figure 3.2d).

At the beginning of the third stage, the system can be in one of two states: 100 units in storage if the previous stage's demand was 100 units, or zero units in storage if the previous stage's demand was 300 units. If there are zero units in storage and the demand in the third stage is high, then the optimal solution is to produce 200 units during standard production hours and 100 units during overtime hours. In all other cases, it is possible to meet the demand using a combination of the units in storage and standard production.

3.3 Unique design features

In this section, we provide a deeper discussion of some of the unique features of `SDDP.jl`.

3.3.1 Cut oracles

As the SDDP algorithm progresses, many cuts (typically thousands) are added to each subproblem. This increases the computational effort required to solve each subproblem. In addition, many of the cuts created early in the solution process may be redundant once additional cuts are added. This issue has spurred a vein of research into heuristics for choosing cuts to keep or remove. (We discussed this method of *cut selection* in Section 2.4.) To facilitate the development of cut selection heuristics, `SDDP.jl` features the concept of a *cut oracle* associated with each subproblem. A cut oracle has two jobs: it should store the complete list of cuts created for that subproblem; and when asked, it should provide a subset of those cuts to retain in the subproblem. This can be done by defining a new subtype of the

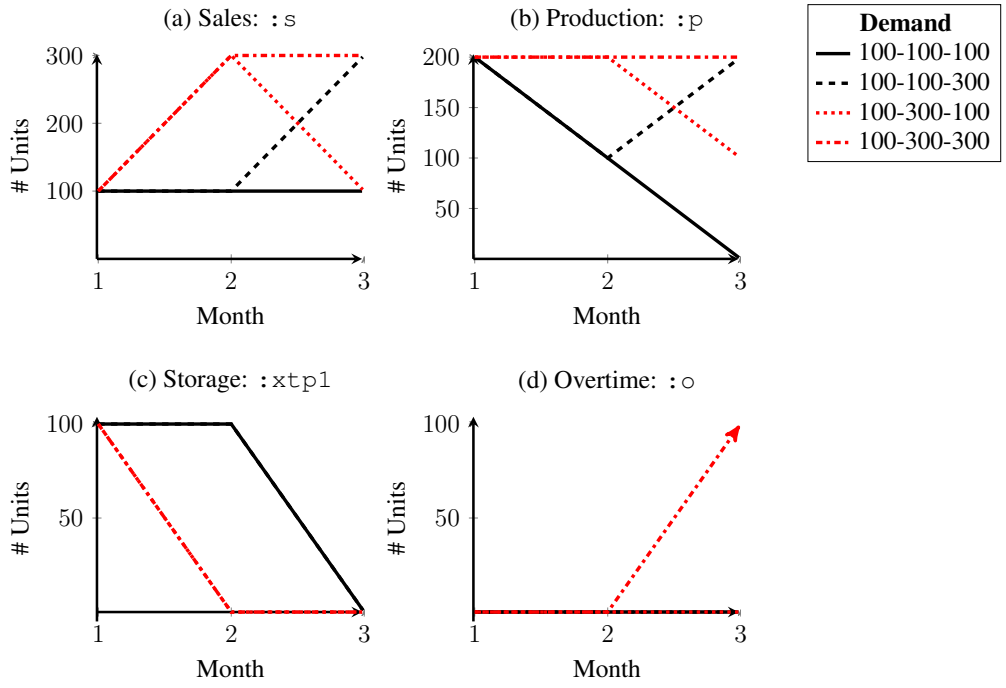


Figure 3.2: Four simulations (each sampling a different demand scenario) of the air conditioner problem using the optimal policy. Red lines sample high demand in stage two ($\omega_2 = 300$ units).

abstract type `AbstractCutOracle` (defined in `SDDP.jl`), and then overloading two methods: `SDDP.storecut!` and `SDDP.validcuts`. To illustrate this feature, we now give the code to implement a new cut oracle that is not implemented in `SDDP.jl`. This oracle only stores the N most recently discovered cuts (called the *last-cuts* strategy in de Matos et al. [55]).[5] First, we define a new Julia type that is a subtype of the abstract type `AbstractCutOracle` defined by `SDDP.jl`:

```
struct LastCutOracle{N} <: SDDP.AbstractCutOracle
    cuts::Vector{SDDP.Cut}
end
LastCutOracle(N::Int) = LastCutOracle{N}(SDDP.Cut[])
```

`LastCutOracle` has the type parameter[6] `N` to store the maximum number of the most-recent cuts to return. The type has the field `cuts` to store a vector of discovered cuts. More elaborate cut selection heuristics may need additional fields to store other information. Next, we overload the `SDDP.storecut!` method. This method should store the cut `c` in the oracle `o` so that it can be queried later. In our example, we append the cut to the list of discovered cuts inside the oracle:

```
function SDDP.storecut!{N}(o::LastCutOracle{N},
        m::SDDPModel, sp::JuMP.Model, c::SDDP.Cut)
```

[5] Readers should note that this is not necessarily an oracle that one would want to use.
[6] A Julia construct.

```
    push!(o.cuts, c)
end
```

Lastly, we overload the `SDDP.validcuts` method. In our example, the strategy is to return the N most recent cuts. Therefore:

```
function SDDP.validcuts{N}(o::LastCutOracle{N})
    return o.cuts[max(1,end-N+1):end]
end
```

By creating a new type that is a subtype of `AbstractCutOracle`, users can leverage Julia's multiple dispatch mechanisms to easily integrate new functionality into the library without having to dive into the internals. At present, only the default behaviour of no cut selection and the Level One cut selection method [55] have been implemented. The Level One cut selection oracle can be created as follows:

```
cut_oracle = LevelOneCutOracle()
```

Cut oracles can be added to the model with the `cut_oracle` keyword in the `SDDPModel` constructor.[7] For example:

```
m = SDDPModel(
    # ... other keyword arguments ...
        cut_oracle = LastCutOracle(500)
                    ) do sp, t
    # ... subproblem definition will go here ...
end
```

Due to the design of JuMP, we are unable to delete cuts from the model.[8] Therefore, selecting a subset of cuts involves rebuilding the subproblems from scratch. The user can control the frequency by which the cuts are selected and the subproblems rebuilt with the `cut_selection_frequency::Int` keyword argument of the `solve` method. Frequent cut selection (i.e. when `cut_selection_frequency` is small) reduces the size of the subproblems that are solved but incurs the overhead of rebuilding the subproblems. However, infrequent cut selection (i.e. when `cut_selection_frequency` is large) allows the subproblems to grow large (by adding many constraints), leading to an increase in the solve time of individual subproblems. Ultimately, this will be a model-specific trade-off. As a rule of thumb, simpler (i.e. few variables and constraints) models benefit from more frequent cut selection compared with complicated (i.e. many variables and constraints) models.

[7] Readers may argue that a cut oracle belongs to the solution process rather than the model object. However, this design is necessary to satisfy some of Julia's quirks regarding type stability.

[8] This may be rectified in future re-writes of JuMP.

3.3.2 Risk measures

Recently, significant efforts have been made to incorporate risk in the SDDP algorithm [120, 121, 159, 160, 187, 189]. In `SDDP.jl`, we use the "change-of-probability" approach of Philpott et al. [159] that we outlined in Section 1.3. For each risk measure, we create a function that takes the K realisations of the random variable $\{z_k\}_1^K$ with corresponding probabilities $\{p_k\}_1^K$ and returns the risk-adjusted probability distribution $\{\xi_k\}_1^K$. For example, when minimising, the worst-case risk measure is:

$$\xi_k = \begin{cases} 1, \ k = \underset{i \in \{1,2,\ldots,K\}}{\arg\max} \ \{z_i\} \\ 0, \ otherwise. \end{cases} \tag{3.2}$$

In a manner similar to the *cut oracle* design, `SDDP.jl` allows a core function (`SDDP.modifyprobability!`) to be overloaded to allow the development and testing of new risk measures. The `modfiyprobability!` function takes a number of arguments. The first is an instance of the risk measure. The second is a vector of `Float64` corresponding to the risk-adjusted probability of each scenario. This vector should be modified in-place by the function. The third argument is a vector of the un-adjusted probabilities (i.e. the original probability distribution). The fourth argument is a vector of observations (one for each realisation of the noise ω). The fifth and sixth arguments are the `SDDPModel` and JuMP subproblems to allow for other risk measures that require additional information.

To illustrate this feature, we give the code necessary to create the worst-case risk measure. Our implementation assumes that the worst outcome in `observations` has a positive probability of occurring. A proper implementation accounting for outcomes with zero probability is too verbose for this simple example. The worst-case risk measure places all of the probability on the scenario with the greatest objective value if minimising, or the smallest objective value if maximising:

```
struct WorstCase <: SDDP.AbstractRiskMeasure end

function SDDP.modifyprobability!(::WorstCase,
        riskadjusted_distribution,
        original_distribution::Vector{Float64},
        observations::Vector{Float64},
        m::SDDPModel,
        sp::JuMP.Model
    )
    if getsense(sp) == :Min
        # indmax returns the index of the maximum element
        idx = indmax(observations)
    else
```

```
        # indmin returns the index of the minimum element
        idx = indmin(observations)
    end
    # reset the distribution
    riskadjusted_distribution .= 0.0
    # place all the weight on the worst outcome
    riskadjusted_distribution[idx] = 1.0
end
```

After defining this code, the user could use `risk_measure = WorstCase()` in their definition of the `SDDPModel` object *as if it were a risk measure defined in `SDDP.jl`*. This again highlights one of the core benefits of building an SDDP framework in Julia: that the user can easily extend the underlying library without modifying the library itself.

3.3.3 Parallelisation

In the previous chapter, we described how the SDDP algorithm is amenable to parallelisation. In particular, we described an asynchronous solution scheme that minimises the amount of inter-process communication. In `SDDP.jl`, this functionality can be controlled by the `solve_type` keyword to the `solve` method. For example, `solve_type=Serial()` will solve the problem utilising one process, while `solve_type=Asynchronous()` will utilise all of the processes available to Julia. Typically, users should start Julia with the same number of processes as they have cores. For example, on a typical 4-core workstation with 16Gb of RAM, we find starting Julia with `julia -p 4` works the best.

To demonstrate the parallel scaling properties of `SDDP.jl`, we solve an instance of the POWDER model from Chapter 8 (the details of the model are unimportant at present) for 2000 iterations, using a virtual machine with 20 Intel E5-2698 CPUs and 64GB of memory, for different numbers of Julia processes. The results are plotted in Figure 3.3. The dashed grey line represents perfect scaling (i.e. doubling the number of processes halves the solution time).

These results show that doubling the number of slaves from 1 to 2 more than halves the solution time. This is unexpected since we would expect that there is some overhead. However, we repeated the experiment a number of times and always observed identical results. As we double the number of slaves to 4, and then to 8 processes, the solution time approximately halves each time as expected. However, once we add more than eight workers, the solution time begins to plateau. This indicates that there is a bottleneck in the system. One bottleneck could be due to a limitation on the speed of communication between processes (since we used a virtual machine, this can vary), or because the master process is becoming a bottleneck in the system: it must collect and then distribute the cuts from each slave at every iteration, as well as writing the cuts to file and deciding whether to terminate the algorithm.

Figure 3.3: Solution time against the number of slave processes showing actual (solid) and ideal (dashed) scaling rates.

3.3.4 Other extensions

Since `SDDP.jl` is built upon JuMP, arbitrary nonlinear convex subproblems are supported by `SDDP.jl`. This enables the user to solve problems incorporating quadratic objectives such as the problem of Guan and Philpott [92].

In addition, one core goal of Dunning et al. [66] in designing JuMP was to make it extensible. Therefore, JuMP contains functionality for injecting user-code into many of the points in the solution process. By building on top of JuMP and Julia, we have leveraged these features to allow other authors to implement features on top of `SDDP.jl` that seamlessly integrate into the existing infrastructure. The first such example of this is the Julia package `SDDiP.jl` [116], which implements the method of Zou et al. [213] to solve multistage stochastic integer programs. We believe the modularity and extensibility of the `SDDP.jl` framework is a rich starting point for future researchers to build upon.

3.4 Benchmark: hydro-thermal scheduling

In Section 3.2, we showed how to solve a multistage stochastic optimisation problem using `SDDP.jl`. In this section, we use a more complicated model to benchmark the performance of `SDDP.jl` against an existing C++ implementation of the SDDP algorithm.

The most common application of the SDDP algorithm (dating back to the original paper of Pereira and Pinto [153]) is the Hydro-Thermal Scheduling Problem (HTSP). In the HTSP, an operator owns a number of hydro-reservoirs that are connected by a series of rivers and hydroelectric generators. Water can be released from the reservoirs and directed through hydroelectric turbines to produce electricity. However, future inflows into the reservoirs from rainfall or snow-melt are uncertain. Any unmet electricity demand is met by thermal generation. Therefore, the objective of the operator is to find a strategy for controlling the release of water over a planning horizon that minimises the total cost of thermal generation.

Software for solving this problem using the SDDP algorithm has been successfully com-

mercialised by the Brazilian company PSR [168].[9] However, for this chapter, we used `DOASA`, a C++ implementation of the SDDP algorithm for the New Zealand HTSP.[10] In contrast to the generic `SDDP.jl`, `DOASA` is hard-coded to solve the New Zealand HTSP. This allows it to gain some efficiency over `SDDP.jl`. Our copy of `DOASA` was provided by the Electric Power Optimisation Centre at the University of Auckland, New Zealand.

To implement the New Zealand HTSP in `SDDP.jl`, we referred only to the description of the model given in Philpott and Pritchard [158]. At no point did we refer to any of the C++ source-code. We now test the correctness and performance of `SDDP.jl` by comparing it with `DOASA`. In the following, all experiments were conducted on a Windows 7 machine with an Intel i7-4770 CPU and 16GB of memory, and all solvers used the default parameter settings.

3.4.1 Correctness

To test the correctness of our `SDDP.jl` implementation of the HTSP model, five deterministic instances of the New Zealand HTSP were created using historical inflows, demand, and pricing for the years 2005 – 2009. These problems were solved for 100 SDDP iterations using `DOASA` and `SDDP.jl`. In all years, both implementations converged to identical lower bounds (Table 3.1). This experiment strongly suggests that the two implementations of the model and deterministic SDDP algorithm are functionally equivalent.

	Year				
	2005	2006	2007	2008	2009
`DOASA`	493,125,281	423,420,729	575,859,349	446,507,222	340,096,459
`SDDP.jl`	493,125,281	423,420,729	575,859,349	446,507,222	340,096,459

Table 3.1: Lower bound ($) after 100 SDDP iterations.

To test the correctness of `SDDP.jl` and `DOASA` on a stochastic problem, an instance of the New Zealand HTSP was created using historical inflows from 1970–2007 and demand and pricing data from 2008. This problem was solved for 5000 SDDP iterations using both implementations. In Figure 3.4, we plot the lower bound against the number of iterations for both implementations. Due to the different random number generators used by `DOASA` and `SDDP.jl`, different random inflows are sampled. This can lead to a difference in the bound at any particular iteration. However, we see clear evidence that both implementations converge towards an identical bound at approximately the same rate. When combined with the deterministic experiments, there is very strong evidence that the SDDP algorithms in `DOASA` and `SDDP.jl` are correct, and that the implementations of the New Zealand HTSP are equivalent. To the best

[9] Somewhat confusingly, the software is named SDDP.

[10] To add to the naming confusion, the term DOASA was also used to refer to a class of algorithms related to SDDP (the algorithm) by Philpott and Guan [161]. We shall refer to `DOASA` the software by stylising it in typewriter font.

of our knowledge, this is the first time that the correctness of an SDDP implementation in a real-world setting has been demonstrated in the literature.

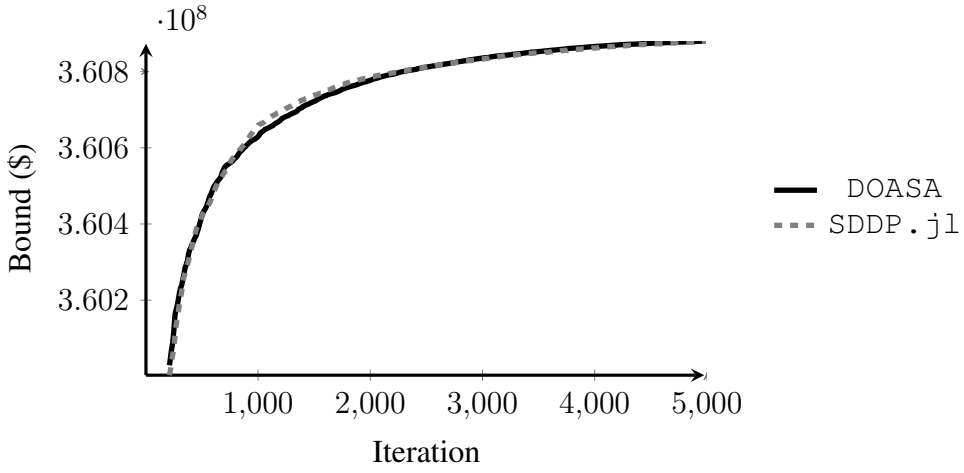


Figure 3.4: Convergence of the lower bound for `DOASA` (solid) and `SDDP.jl` (dashed) on a stochastic instance of the New Zealand HTSP.

3.4.2 Performance

To compare the performance of `SDDP.jl` and `DOASA`, an instance of the New Zealand HTSP was created using 38 years of historical inflows (1970–2007), with data from 2008 for demand and thermal pricing. Four different solver configurations were setup: `DOASA` using Gurobi version 6.5.0 [97], `SDDP.jl` using Gurobi version 6.5.0, `SDDP.jl` using Gurobi version 7.0.0 [98], and `SDDP.jl` using CPLEX version 12.6.1 [112]. The problem was solved 20 times for each of the configurations. The SDDP algorithm was terminated after 500 cuts had been generated for the first subproblem.

In Table 3.2, we summarise the results of these experiments. There are four columns of interest: *External*, *JuMP*, *SDDP*, and *Total*. In the *External* column, we report the time spent in the external solver libraries (i.e. CPLEX and Gurobi) solving the subproblems. In the *JuMP* column, we report the time that `SDDP.jl` spends in calls to the JuMP method `solve`, excluding the call to the external solver. This measures the overhead of solving the problem via the generic JuMP interface rather than directly through the solver API. In the *SDDP* column we measure the time that is spent performing tasks related to the SDDP algorithm, excluding the solve time. These include sampling the random variables, calculating cut coefficients, and writing information to file. We have aggregated the *JuMP* and *SDDP* columns for the `DOASA` configuration as it does not use JuMP. Finally, in the *Total* column, we report the total time spent solving the 500 iterations.

After 500 iterations, the lower bound of all configurations was similar ($\sim$\$360.4 million). However, solution times varied between `DOASA` and `SDDP.jl`, Gurobi and CPLEX, and even between different Gurobi versions. Of the four configurations, `SDDP.jl` with CPLEX

	Solver	Time (s) External	JuMP	SDDP	Total
`DOASA`	Gurobi v6.5.0	458.1 (1.1)	60.5 (0.8)		518.6 (1.4)
`SDDP.jl`	Gurobi v6.5.0	559.0 (1.0)	68.7 (0.7)	41.5 (0.5)	669.2 (1.5)
`SDDP.jl`	Gurobi v7.0.0	580.2 (1.1)	69.8 (0.7)	41.6 (0.6)	691.6 (1.4)
`SDDP.jl`	CPLEX v12.6.1	319.6 (3.6)	75.9 (1.4)	41.3 (0.9)	436.8 (5.4)

Table 3.2: Solution time after 500 iterations. All values are reported as the mean (standard deviation) of twenty repetitions.

v12.6.1 performed the fastest (mean of 436.8 seconds over the 20 repetitions to solve 500 iterations), followed by `DOASA` (518.6 seconds), `SDDP.jl` with Gurobi v6.5.0 (669.2 seconds), and `SDDP.jl` with Gurobi v7.0.0 (691.6 seconds), with the result that `SDDP.jl` was faster than `DOASA`.

However, despite solving an identical number of linear programs, the `SDDP.jl` configuration with Gurobi v6.5.0 spent 22% longer in the external solver than `DOASA` with Gurobi v6.5.0 (and 29% longer overall). This is because `DOASA` can use more efficient cut selection heuristics than `SDDP.jl` due to the tight coupling between `DOASA` and Gurobi. In particular, as we discussed in Section 2.4.2, `DOASA` uses a form of Level One cut selection and modifies the coefficients of existing constraints in-place. The 20% difference in computation time between `DOASA` and `SDDP.jl` is one reason why we believe that modifying the coefficients in-place is computationally beneficial.

In contrast to the tight coupling between `DOASA` and Gurobi, the combination of `SDDP.jl` and JuMP allows the solver to be changed by the user with a single line of code. Thus, it is easy to show that despite solving larger subproblems, `SDDP.jl` with CPLEX v12.6.1 was 15% faster than `DOASA` with Gurobi v6.5.0. This suggests that for this particular model, CPLEX is faster than Gurobi, and that this speedup outweighs the overhead of going through the JuMP abstraction layer. This overhead is approximately 10–20% of the total solution time. However, using JuMP allows the user to specify the subproblems using the simple input syntax described in the previous two sections. In contrast, `DOASA` builds the constraint matrix directly using the Gurobi C++ API. This significantly increases the development time needed to modify and debug the subproblems (for example, when a new feature is added).

Interestingly, `SDDP.jl` with Gurobi v7.0.0 is 3.7% slower than `SDDP.jl` with Gurobi v6.5.0. This could indicate that the improvements between Gurobi version 6.5.0 and Gurobi version 7.0.0 benefit difficult problems that take a long time to solve at the expense of small LP's. (On average, the subproblems in the New Zealand HTSP take on the order of 50μs to solve.)

Finally, less than 10% of the total solution time is spent performing tasks in the `SDDP.jl` library. This highlights the efficiency of the Julia implementation and demonstrates that further performance gains are more likely to be found by reducing the solve time of the individual subproblems than by improving the code in the `SDDP.jl` library.

3.5 Comparison with other libraries

`SDDP.jl` is not the only library for solving multistage stochastic optimisation problems using SDDP. We briefly describe six alternatives and contrast their abilities to `SDDP.jl`. A summary can be found in Table 3.3.

Four of the alternatives are open-source: StochDynamicProgramming.jl [125], StructDualDynProg.jl [126], FAST (Finally, An SDDP Toolbox) [38], and StOpt [83]. The remaining two are commercial software: QUASAR [171], and the seminal SDDP by PSR [168].

Perhaps the most interesting observation is that (including the PSR re-write), four of the seven implementations are in Julia. An even more interesting observation is that each of the authors chose to develop an SDDP library in Julia independently, and all within a few months of each other. This supports our belief that the combination of Julia's metaprogramming and multiple dispatch abilities, as well as the JuMP [66] package and wider JuliaOpt ecosystem, are the ideal foundation upon which to build an SDDP library.

Since all of the Julia libraries are based on the JuliaOpt ecosystem, they share many of the same features. StochDynamicProgramming.jl and StructDualDynProg.jl even share the same (user-extensible) cut selection routines. Moreover, since `SDDP.jl` and StructDualDynProg.jl expose the full functionality of JuMP, they support arbitrary convex subproblems. StochDynamicProgramming.jl does not expose the full functionality of JuMP. Instead it implements a different modelling layer. This allows it to support noise terms in the constraint coefficients (something notably missing from `SDDP.jl`). However, this comes at the expense of a less user-friendly modelling interface. For example, the following is an implementation of the air conditioner problem in StochDynamicProgramming.jl:

```
m = LinearSPModel(
    3,                      # stages
    [(0, 200), (0, Inf)], # control bounds
    [0.0],                  # initial state
    # cost function
    (t,x,u,w) -> 100u[1] + 300u[2] +50(x[1] + u[1] + u[2] - w[1]),
    # dynamics function
    (t,x,u,w) -> [ x[1] + u[1] + u[2] - w[1] ],
    [ # stagewise-independent noise terms
        NoiseLaw([100], [1.0]),
        NoiseLaw([100, 300], [0.5, 0.5]),
        NoiseLaw([100, 300], [0.5, 0.5])
    ]
)
set_state_bounds(m, [(0, Inf)])
solve_SDDP(m,
    SDDPparameters(ClpSolver(), max_iterations = 10)
```

Name	SDDP.jl	StochDynamicProgramming.jl	StructDualDynProg.jl	FAST	StOpt	QUASAR	PSR-SDDP[4]
Availability License	Free MPL-2.0	Free MPL-2.0	Free MIT	Free GPLv3	Free LGPLv3	Commercial	Commercial
Language	Julia	Julia	Julia	MATLAB	C++[2]	Java[3]	Fortran
General?	Yes	Yes	Yes	Yes	Yes	Yes	No
Policy Graph	Markovian	Linear	Acyclic[1]	Markovian	Markovian	Markovian	Markovian
Subproblems	Convex	Linear	Convex	Linear	Linear	Quadratic	Linear
Noise	RHS/Obj.	Any	No	No	RHS	Any	Any
Risk	Custom	Custom	No	No	No	Yes	Yes
Cut Selection	Custom	Custom	Custom	No	No	Yes	Yes
Solvers	Many	Many	Many	Many	User	Many	Xpress

Table 3.3: SDDP library comparison. "Language" refers to the host language. "General" refers to whether the library can be used for a variety of problems. "Subproblems" refers to whether the subproblems must be linear, quadratic, or arbitrary convex programs. "Noise" refers to whether the library supports stagewise-independent noise terms. "Risk" refers to whether the library supports nested risk measures. [1]StructDualDynProg.jl supports arbitrary cyclic graphs *only* if the stage-objective is zero in every stage. [2]StOpt also has a Python interface. [3]QUASAR also has MATLAB and Python interfaces. [4]PSR is currently re-writing their implementation in Julia. The new version will support convex subproblems and many different solvers (J. Dias Garcia, personal communication, April, 2018).

```
)
```

The `SDDP.jl` equivalent is:

```
m = SDDPModel(
              stages = 3,
    objective_bound = 0.0,
               sense = :Min,
              solver = ClpSolver()
                         ) do sp, t
    @state(sp, x_out >= 0, x_in == 0)
    @variables(sp, begin
        0 <= p <= 200
        0 <= o <= Inf
    end)
    DEMAND      = [ [100], [100, 300], [100, 300] ]
    @rhsnoise(sp, w=DEMAND[t], x_out == x_in + p + o - w)
    PROBABILITY = [ [1.0], [0.5, 0.5], [0.5, 0.5] ]
    setnoiseprobability!(sp, PROBABILITY[t])
    @stageobjective(sp, 100p + 300o + 50x_out)
end
solve(m,
    max_iterations=10
)
```

One reason for the design choice of StochDynamicProgramming.jl is so that the same `LinearSPModel` definition can be solved a variety of ways, including stochastic dynamic programming, and by an LP as the deterministic equivalent.

Unlike the other libraries, StructDualDynProg.jl can model problems with an arbitrary (acyclic) policy graph. However, it does not support stagewise-independent noise terms. Therefore, the user is forced to expand the policy graph in a manner similar to our example in Figure 1.5. FAST also lacks support for stagewise-independent noise terms. This is a large limitation since an increase in the number of subproblems causes a corresponding increase in memory requirements. Moreover, computation time may also increase.

Compared with the other open-source libraries, StOpt is implemented in a low-level language (C++). Moreover, it provides no modelling capability. Instead, the user codes their own subproblems, including the interface with any solver. This significantly adds to the development cost of new models. StOpt also lacks many of the features supported by the other libraries, such as risk aversion and cut selection.

Finally, the two commercial libraries (QUASAR and PSR's SDDP) contain many of the advanced features in SDDP such as risk aversion, cut selection, and parallel implementations.

However, since these libraries are focused on the end-user, they offer fewer customisation options to the user. PSR's SDDP [168] is also unique as the only non-general library we compare; it is exclusively for hydro-thermal scheduling.

From this analysis, we conclude that `SDDP.jl` is a valuable contribution to the suite of libraries for SDDP. Its biggest strengths are the first-class support for risk measures and cut selection techniques, as well as the user-friendly interface. The biggest weakness of `SDDP.jl` is the lack of support for noise terms in the constraint coefficients. However, these can be modelled by a Markovian policy graph.

3.6 Summary

In this chapter we introduced `SDDP.jl`, a Julia package for stochastic dual dynamic programming. In addition to describing the convenient user-interface, we showed that for the New Zealand HTSP, the overhead of using JuMP and the generic library `SDDP.jl` compared with a hard-coded C++ implementation is less than the difference between two commercial LP solvers. We also provided strong evidence that the implementation is correct by showing that two independently developed implementations give identical solutions.

There are many more features to `SDDP.jl` that we have not discussed in the interest of conciseness. For example, we have not discussed how to implement Markovian policy graphs or noise terms in the objective function. Instead, readers are directed to the online documentation at `github.com/odow/SDDP.jl` for more information.

We believe the unique features of `SDDP.jl` (that it is written entirely in a high-level language and built upon the state-of-the-art mathematical optimisation library JuMP) provide an excellent platform upon which to build and test new improvements and extensions to the SDDP algorithm. In the next chapter, we discuss one such extension to the SDDP algorithm that enables the solution of problems with stagewise-dependent noise in the objective function. In addition, we implement and test the extension in `SDDP.jl`.

Chapter 4

SDDP with stagewise-dependent objective uncertainty

The previous two chapters have focused on the stochastic dual dynamic programming (SDDP) algorithm of Pereira and Pinto [153] (including various extensions and computational improvements) as a solution technique for multistage stochastic optimisation problems. However, SDDP is limited to models where the cost-to-go function is convex. In this chapter we present two extensions of the SDDP algorithm that enable the solution of a class of problems where the cost-to-go function is a saddle function (i.e. convex in some state dimensions and concave in the others). The first method is a hybrid of SDDP and the stochastic dynamic programming method. The second is based on recent work by Downward et al. [65]. In addition to describing and providing practical implementations of both methods, we extend the convergence result of Downward et al. [65] to a broader class of problems. We demonstrate the computational performance of both methods through a simple example.

4.1 Introduction

Since its inception, SDDP has been widely applied to a variety of problems in energy [135], finance [120], and agriculture [92]. However, one significant downside to the SDDP algorithm is the requirement that the cost-to-go function is convex. This has ruled out the use of SDDP on many applications with a price process that exhibits stagewise dependence.

To avoid the convexity requirement, Gjelsvik et al. [85] proposed a hybrid of the SDDP and stochastic dynamic programming (SDP) algorithms. However, the price process they modelled only took a limited number of discrete values. This enabled the exact discretisation of the price dimension. This approach was later extended by Philpott and de Matos [160] and Rebennack [173] to allow more flexibility in the types of uncertainties that could be represented. (Using the terminology introduced in Chapter 1, the above authors modelled the stochastic processes using Markovian policy graphs.) However, these methods retained the significant downside that

the uncertainty process had to take a limited number of discrete values (i.e. Markov states) to maintain tractability of the problem.

In traditional SDP, a common approach to avoid this restriction is to use an interpolation scheme to evaluate points in the state-space that are not in the discretised lattice. In the SDDP setting, this corresponds to interpolating between the cost-to-go functions of different Markov states. The first mention of this idea appears to be in Gjelsvik et al. [86]. However, the reference to the interpolation idea is made in passing and likely existed before then. We are aware of one other implementation of the idea in the thesis of Wahid [202], who explored the method in a hydroelectric setting. Henceforth we shall refer to this method as the *static interpolation method*.

Recently, a new method has been proposed by Downward et al. [65], henceforth called the *dynamic interpolation method*. The dynamic interpolation method adds new points in the discretisation lattice via sampling. In contrast to the static interpolation method, which may have some interpolation error and thus not converge to the optimal solution, Downward et al. [65] provide a formal proof that their method converges to the optimal policy, almost surely, in a finite number of iterations.

Their convergence result applies to the case where the price process is linear and the stage-objective (in stage t) has the form ${p'_t}^\top Q_t x_t$, where p'_t is a vector of *price-state* variables, Q_t is an appropriately dimensioned matrix, and x_t is a vector of state variables. This class of problems includes, for example, the (one-dimensional) mean-reverting auto-regressive price process with lag-one:

$$p'_t = \alpha p_t + (1-\alpha)\mu + \varepsilon_t,$$

with an objective function of $p'_t \times x_t$.

However, in many applications, the price process can be modelled more accurately by a multiplicative auto-regressive process [70, 111]:

$$\log(p'_t) = \alpha \log(p_t) + (1-\alpha)\mu + \varepsilon_t.$$

This clearly breaks the assumption of Downward et al. that the price process is linear. One way of overcoming this is to perform a change-of-variables by taking the log of the price-state. We denote this as $y_t = \log(p_t)$. The transition of the y_t state variable is linear ($y'_t = \alpha y_t + (1-\alpha)\mu + \varepsilon_t$); however, the price p'_t will appear in the objective as $e^{y'_t}$. This case is not covered by the convergence proof of Downward et al. [65], and thus serves as the motivation for this chapter.

The chapter is laid out as follows. In Section 4.2, we formulate a simple example that we use as the basis for introducing the two interpolation methods. Then, in Section 4.3, we introduce the static interpolation method. In Section 4.4, we introduce the dynamic interpolation method. This includes an extension in Section 4.4.2 to the convergence result of Downward et al. [65] so that it applies to a broader class of problems, including the multiplicative auto-regressive

process described above. Finally, in Section 4.5, we propose a new method that is a hybrid of the static and dynamic interpolation methods.

4.2 Example: the widget producer

To motivate our discussion in this chapter, we consider the problem faced by a widget producer over a period of five days. They have a stock of widgets (denoted by x_t at the start of day t) and need to choose the quantity of widgets to sell (denoted u_t on day t) each day. At the start of each day, and before the sales decision is made, some widgets are produced. However, the process by which the widgets are produced is stochastic. The quantity ω_t^w of widgets produced on day t is drawn from the sample space $\Omega_t^w = \{0, 5, 10, \ldots, 50\}$ with uniform probability.

After learning how many widgets were produced on day t, the agent chooses the quantity u_t of widgets to sell. They earn the price of the widgets p_t, multiplied by the quantity u_t of widgets they sell. The price p_t at which the widgets can be sold on day t follows a stochastic process. Specifically, we model it using the log auto-regressive process:

$$\log(p_t) = \log(p_{t-1}) + \omega_t^p,$$

where ω_t^p is uniformly drawn from the sample space:

$$\Omega_t^p = \{\log(w) : w \in \{0.9, 0.95, 0.99, 1, 1.01, 1.05, 1.1\}\}.$$

Importantly, this price process evolves independently of the actions of the agent and the quantity of widgets they have unsold (i.e. x_t). The new price, p_t', is revealed before the agent chooses the quantity of widgets to sell. The agent cannot sell more than 100 widgets per day, and cannot buy widgets, so $u_t \in [0, 100]$ for all $t = 1, 2, 3, 4, 5$. In addition, the agent can only sell widgets from storage or that have been produced during the stage, so $x_t' \geq 0$ for all $t = 1, 2, 3, 4, 5$. They are also limited in the quantity of widgets they can store, so $x_t' \leq 350$ for all $t = 1, 2, 3, 4, 5$. Initially, the agent has 200 widgets, and the price during the previous day was \$1.50.

This problem can be modelled as a multistage stochastic optimisation problem with five Hazard-Decision subproblems that form a linear policy graph (Figure 4.1).

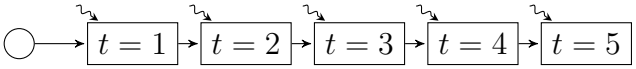


Figure 4.1: Policy graph of widget producer problem.

There are two state variables: (1) the quantity of widgets x_t in storage at the start of day t; and (2) the price of a widget p_t during previous day $t - 1$. In each stage, the noise terms are sampled from the Cartesian product of Ω_t^w and Ω_t^p (i.e. independently). We denote $(\omega_t^w, \omega_t^p) \in \Omega_t^w \times \Omega_t^p$ by $\omega_t \in \Omega_t$. We can express the Hazard-Decision subproblem t as:

$$\begin{aligned}\textbf{HD}_t: \quad V_t(x_t, p_t, \omega_t) = \min_{x'_t, u_t} \quad & -p'_t \times u_t + \mathcal{V}_{t+1}(x'_t, p'_t) \\ \text{s.t.} \quad & x'_t = x_t - u_t + \omega_t^w \\ & \log(p'_t) = \log(p_t) + \omega_t^p \\ & u_t \in [0, 100] \\ & x'_t \in [0, 350],\end{aligned}$$

where $\mathcal{V}_{t+1}(x'_t, p'_t) = \mathbb{E}_{\omega_{t+1} \in \Omega_{t+1}}\left[V_{t+1}(x'_t, p'_t, \omega_{t+1})\right]$, $\mathcal{V}_4(\cdot) = 0$, and the initial conditions are $(x_1, p_1) = (200, 1.50)$. As formulated, $\textbf{HD}_t$ has a bilinear term in the objective function ($p'_t \times u_t$).

One approach to solve the widget producer example is to expand the policy graph (Figure 4.1) into a Markovian policy graph with a different Markov state for every possible price in each stage. However, as the number of possible price values increases, this approach quickly becomes intractable. Traditional dynamic programming overcomes this limitation by interpolating between points on the sampled lattice. This corresponds to estimating the cost-to-go at a Markov state by interpolating between the cost-to-go functions of adjacent Markov states. An example of this is shown in Figure 4.2, where the cost-to-go function when the price is \$1.5 is constructed by a convex combination of the cost-to-go functions when the price is \$1 and \$2.

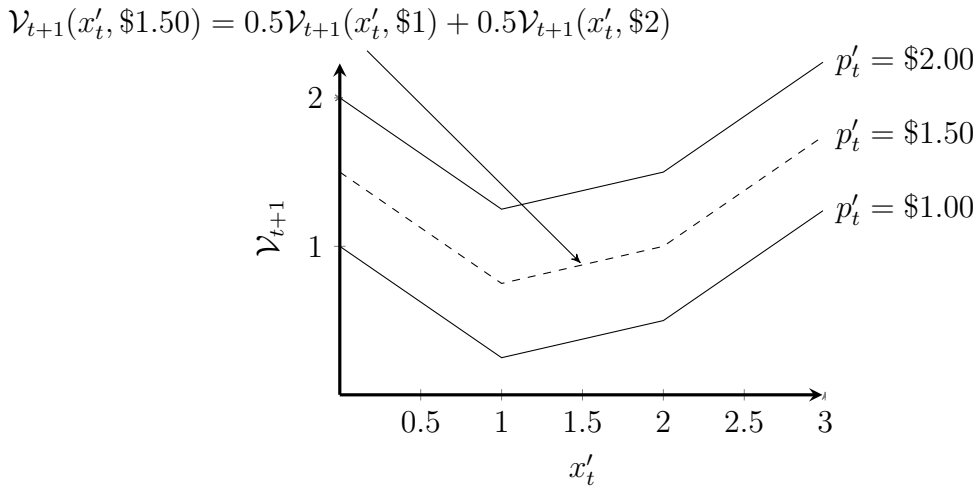


Figure 4.2: Interpolating the cost-to-go function between two existing cost-to-go functions.

To the best of our knowledge, this hybrid method of the stochastic dynamic programming (SDP) and stochastic dual dynamic programming (SDDP) algorithms (without interpolation) was first proposed by Gjelsvik et al. [85]. However, the only subsequent analyses of the interpolation variant were in Gjelsvik et al. [86] and Wahid [202]. We will refer to this method as the *static interpolation method*.

4.3 The static interpolation method

Before we introduce the static interpolation method, we discuss some preliminaries. First, given a function $f(x)$ and a discrete set of points $\bar{x} = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_N\}$, we can construct a piecewise linear interpolation of the function f at the point $\hat{x}$ by the convex combination of the function evaluations at closest discretised points above and below $\hat{x}$. An example for the sine function is shown in Figure 4.3.

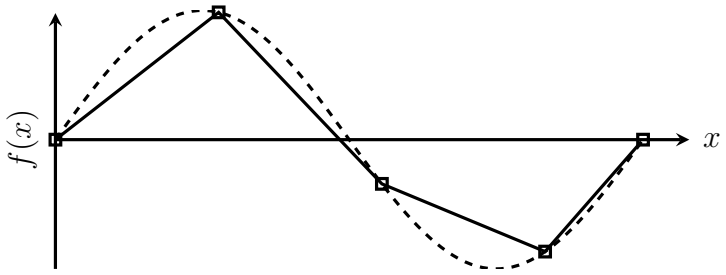


Figure 4.3: Piecewise linear interpolation of a one-dimensional function.

Furthermore, note that if the function f is concave, then by definition, this interpolation results is always a lower bound for the function. Moreover, the convex combination of discretised points that estimates $f(\hat{x})$ can be evaluated according to the following linear program, which we call the Linear Interpolating Problem (**LIP**):

$$\begin{aligned}
\textbf{LIP:} \quad f(\hat{x}) = \max_{\gamma} \quad & \sum_{i=1}^{N} \gamma_i f(\bar{x}_i) \\
\text{s.t.} \quad & \sum_{i=1}^{N} \gamma_i = 1 \\
& \sum_{i=1}^{N} \gamma_i \bar{x}_i = \hat{x} \\
& \gamma_i \geq 0,\ i \in \{1, 2, \ldots, N\}.
\end{aligned}$$

Second, the formulation of $\mathbf{HD}_t$ contains a nonlinear transition function (with respect to the p_t state variable). We perform the substitution $y_t = \log(p_t)$ to obtain the equivalent representation:

$$\begin{aligned}
\mathbf{HD}_t: \quad V_t(x_t, y_t, \omega_t) = \min_{x'_t, u_t} \quad & -e^{y'_t} \times u_t + \mathcal{V}_{t+1}(x'_t, y'_t) \\
\text{s.t.} \quad & x'_t = x_t - u_t + \omega_t^w \\
& y'_t = y_t + \omega_t^p \\
& u_t \in [0, 100] \\
& x'_t \in [0, 350].
\end{aligned}$$

In this form, given fixed values of y_t and ω_t, V_t is convex with respect to x_t. Therefore, we can approximate the expected cost-to-go function with respect to x'_t using cutting planes. For now, we shall also claim (although we will prove it in Section 4.4.2) that V_t is concave with respect to y_t. Therefore, we can approximate the cost-to-go function with respect to y'_t using

the interpolation method described above. This results in an approximation that is a valid lower bound for the cost-to-go function $\mathcal{V}_{t+1}$.

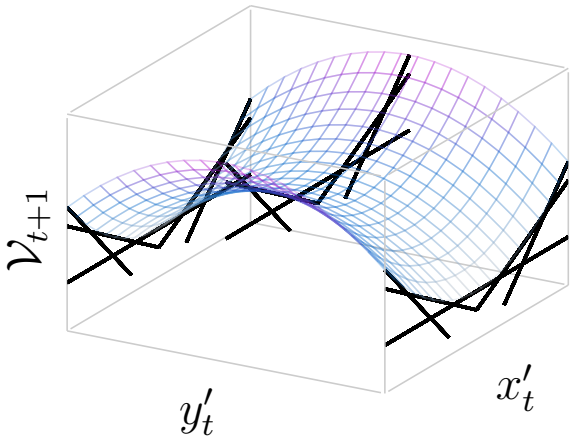


Figure 4.4: Visualisation of the "ribs" over the function $f(x, y) = x^2 - y^2$.

4.3.1 Formulation

In the static interpolation method, we replace the cost-to-go term in the objective of $\mathbf{HD}_t$ with a convex combination of cost-to-go variables. The number of these variables is chosen *a priori*, and each variable $\theta_{t,r}$ is associated with a discretised point $\hat{y}_r$. As in SDDP, each variable $\theta_{t,r}$ is bounded by a set of cuts which are linear functions of the x'_t state variable. We use the term *rib* to describe each cost-to-go variable with its associated cuts, since the method can be visualised by a rib cage, where each rib is composed of a set of cuts, and the ribs are spaced along the price dimension. An example is shown in Figure 4.4. We denote the set of ribs in stage t by $\mathcal{R}_t = \{1, 2, \ldots, N_R\}$. In each subproblem, the price-state y'_t will not necessarily correspond to one of the ribs. Therefore, we interpolate the cost-to-go function between the ribs using the **LIP**. Thus, we can rewrite $\mathbf{HD}_t$ as the approximated problem:

$$
\begin{aligned}
\textbf{AP-Static}_t^K : V_t(x_t, y_t, \omega_t) = \min_{\bar{x}_t, x'_t, u_t, y'_t, \theta} \max_{\gamma} \quad & -e^{y'_t} \times u_t + \sum_{r \in \mathcal{R}_t} \gamma_r \theta_{t,r} \\
\text{s.t.} \quad & x'_t = \bar{x} - u_t + \omega_t^w \\
& \bar{x}_t = x_t, \quad [\lambda_t] \\
& y'_t = y_t + \omega_t^p \\
& u_t \in [0, 100] \\
& x'_t \in [0, 350] \\
& \sum_{r \in \mathcal{R}_t} \gamma_r \hat{y}_r = y'_t \\
& \sum_{r \in \mathcal{R}_t} \gamma_r = 1 \\
& \gamma_r \geq 0, \quad \forall r \in \mathcal{R}_t \\
& \theta_{t,r} \geq \alpha_{t,r}^k + \beta_{t,r}^k x'_t, \ \forall r \in \mathcal{R}_t, \ k \in \{1, \ldots, K\},
\end{aligned}
$$

where the state y_t is the log of the actual price p_t. Note that in the objective, we take the exponential of y_t so that the actual objective is $-p'_t \times u_t$.

Because the new price-state y'_t depends only on the incoming price-state y_t and the observation of the random variable ω_t^p, the price-state can be updated outside the optimisation according to the transition function $y'_t = y_t + \omega_t^p$. Therefore, the value of y'_t in the objective function can be set as a constant, rather than an optimisation variable. Once the value of y'_t is known, the values of γ_r can be calculated. To do this with a one-dimensional price-state, we find the closest rib at a price less than y'_t (denoted $\hat{y}_{r^-}$), and the closest rib at a price greater than y'_t (denoted $\hat{y}_{r^+}$), and set:

$$\gamma_r = \begin{cases} \frac{y'_t - \hat{y}_{r^-}}{\hat{y}_{r^+} - \hat{y}_{r^-}}, & r = r^+ \\ \frac{\hat{y}_{r^+} - y'_t}{\hat{y}_{r^+} - \hat{y}_{r^-}}, & r = r^- \\ 0, & otherwise. \end{cases} \tag{4.1}$$

Note that as a result, the price process must be contained within the bounding box defined by the smallest and largest ribs. If the price-state is outside the ribs, then we cannot find a feasible solution for γ_r. (We expand upon this point in Section 4.3.3.) If the price-state is multi-dimensional, a more complicated interpolation scheme must be chosen.

Combining the ideas we have discussed so far, we can re-write the subproblems as:
SP-Static$_t^K\,(x_t, y_t, \omega_t)$:
Calculate $y'_t = y_t + \omega_t^p$, compute and set γ according to Eq. 4.1, then solve:

$$\begin{aligned} V_t\,(x_t, y_t, \omega_t) = \min_{\bar{x}_t, x'_t, u_t, \theta} \quad & -e^{y'_t} \times u_t + \sum_{r \in \mathcal{R}_t} \gamma_r \theta_{t,r} \\ \text{s.t.} \quad & x'_t = \bar{x}_t - u_t + \omega_t \\ & \bar{x}_t = x_t, \quad [\lambda_t] \\ & u_t \in [0, 100] \\ & x'_t \in [0, 350] \\ & \theta_{t,r} \geq \alpha_{t,r}^k + \beta_{t,r}^k x'_t, \quad \forall r \in \mathcal{R}_t,\ k \in \{1, \ldots, K\}. \end{aligned}$$

Readers should note that the interpolation between the ribs may result in some interpolation error. Therefore, the static interpolation method may not converge to the exact solution to **HD**$_t$. To reduce the interpolation error, more ribs can be added, but this comes at the cost of an increased computational burden.

Modifications to the SDDP algorithm

The forward pass of the static interpolation method is similar to the SDDP algorithm described in Chapter 2. The only difference is the process of calculating y'_t and γ_r outside of the main optimisation problem and updating the objective coefficient before solving each linear program. However, on the backward pass we disregard the prices y'_t that were sampled on the forward pass and add a cut to each rib in every stage *as if we observed the price corresponding to that rib*.

Pseudo-code is given in Algorithm 6.

Algorithm 6: The static interpolation algorithm.

set $K = 0$
while *not converged* **do**
  `/* Forward Pass */`
  set $i = R$, $x = x_R$, $y = y_R$, $\mathcal{S} = [\,]$
  sample i from R^+ according to the transition matrix Φ
  while $i^+ \neq \varnothing$ **do**
    sample ω_i from Ω_i
    solve **SP-Static**$_i^K(x, y, \omega_i)$
    set $x = \hat{x}_i^{K+1} = x_i'$
    set $y = y_i'$
    append $(i, \hat{x}_i^{K+1})$ to the list $\mathcal{S}$
    sample new i from i^+ according to the transition matrix Φ
  end
  `/* Backward Pass */`
  for $(i, \hat{x}_i^{K+1})$ *in* $reverse(\mathcal{S})$ **do**
    for $r \in \mathcal{R}_i$ **do**
      for $j \in i^+$ **do**
        for $\omega_j \in \Omega_j$ **do**
          solve **SP-Static**$_j^K(\hat{x}_i^{K+1}, \hat{y}_r, \omega_j)$
          set $\bar{V}_{j,\omega_j}^K$ to the optimal objective value
          set $\bar{\lambda}_{j,\omega_j}^K$ to the value of λ_j in the optimal solution
        end
      end
      compute the changed distribution ξ for $\mathbb{F}_{j \in i^+;\, \omega_j \in \Omega_j}\left[\bar{V}_{j,\omega_j}^K\right]$
      set $\beta_{i,r}^{K+1} = \mathbb{E}_\xi\left[\bar{\lambda}_{j,\omega_j}^K\right]$
      set $\alpha_{i,r}^{K+1} = \mathbb{E}_\xi\left[\bar{V}_{j,\omega_j}^K\right] - \langle \beta_{i,r}^{K+1}, \hat{x}_i^{K+1} \rangle$
      add the cut $\theta_{i,r} \geq \alpha_{i,r}^{K+1} + \langle \beta_{i,r}^{K+1}, x_i' \rangle$ to **SP-Static**$_i^K$
    end
  end
  increment K
end

Implementation in SDDP.jl

The static interpolation method is implemented in the `SDDP.jl` package that we introduced in the previous chapter. Two changes are made to the user-interface in order to accommodate the new method. First, `value_function`, a new keyword argument to the `SDDPModel` is introduced. This argument must be a `StaticPriceInterpolation` object, which itself takes a number of arguments. They are:

- `dynamics`: a function that takes the price-state y_t and noise ω_t as arguments and returns y'_t.
- `initial price`: the value of the price-state at the root node y_R.
- `rib_locations`: the list of rib locations $\hat{y}_r$.
- `noise`: a `DiscreteDistribution` representing the noise. The first argument is a list of observations, the second argument is a list of the corresponding probabilities.
- `cut_oracle`: the cut selection heuristic to apply to each rib.

Below we give the code to initialise the static interpolation method for the widget producer example. Note that we pass `value_function` a function that takes the stage and Markov state as inputs and returns a `StaticPriceInterpolation` object. Also note that we need to ensure that the ribs cover the price domain so that there exists a feasible value of γ_r for all reachable price-states in stage t. We elaborate upon this requirement in Section 4.4.4.

```
function buildvaluefunction(stage, markovstate)
    R = log.([0.9, 0.95, 0.99, 1.0, 1.01, 1.05, 1.1])
    min_y = log(1.5) + stage * minimum(R)
    max_y = log(1.5) + stage * maximum(R)
    StaticPriceInterpolation(
            dynamics      = (y,noise) -> y + noise,
            initial_price = log(1.50),
            rib_locations = linspace(min_y, max_y, 5),
            noise         = DiscreteDistribution(R),
            cut_oracle    = LevelOneCutOracle()
        )
end
m = SDDPModel(
    ... some arguments omitted ...
    value_function = buildvaluefunction
) do sp, t
    ... some lines omitted ...
end
```

In addition to the `value_function` keyword, a new version of the `@stageobjective` macro is introduced, which takes a function as the second argument. This function takes one input, the price-state y'_t, and returns the stage-objective. For example, given a JuMP variable `u` that represents the number of widgets sold, the stage-objective for the widget example is:

```
@stageobjective(sp, y -> -exp(y)*u)
```

4.3.2 Results

We solve the widget producer example using a varying number of evenly spaced ribs in each stage. In Figure 4.5, we plot the lower bound after 100 iterations against the number of ribs in each stage. As the number of ribs increases, the bound converges towards the value of $-\$487.59$.

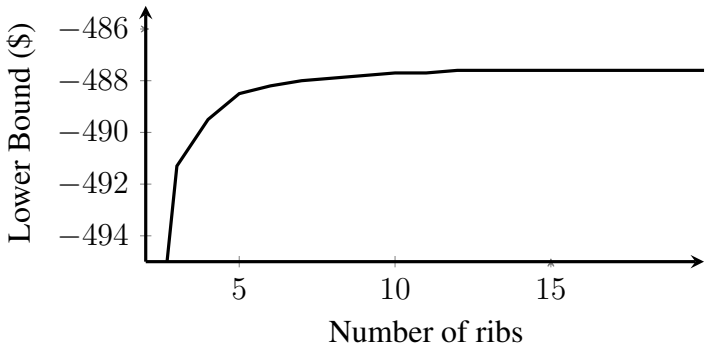


Figure 4.5: Convergence of the lower bound as the number of static ribs increases.

To understand the policy that the widget producer should adopt, we take the optimal policy from the model with 15 ribs and perform a Monte Carlo simulation with 1000 replications. The results are plotted in Figure 4.6. In each of the subplots, we plot, as shaded bands in order of increasing darkness, the 0–100, 10–90, and 25–75 percentiles of the distribution of the plotted variable. The solid line corresponds to the 50$^{\text{th}}$ percentile.

In the first stage, the optimal decision is to sell (Figure 4.6d) the quantity of widgets produced during the first stage (Figure 4.6c). Then, in stages $t = 2, 3, 4, 5$, the optimal decision is to sell 50 widgets, plus the quantity produced during that stage. As a result, the quantity of widgets in storage (Figure 4.6a) decreases linearly. However, in a small number of simulations ($\approx 1\%$), the policy sells the maximum number of widgets (100 widgets) in the first two stages. This is a sub-optimal decision since the price process trends upwards, and occurs because of the interpolation error that results from the fixed number of ribs.

In most simulated realisations, the price remains within a band ranging from \$1.30 to \$1.70 (Figure 4.6b). However, in some cases, a sequence of high prices is observed (e.g. $p_4, p_5 \geq \$2$).

4.3.3 Practical considerations

The chief consideration when solving a problem using the static interpolation method is choosing the number and placement of the ribs. Unfortunately, our experience suggests that this choice is highly problem-dependent. However, since the process evolves exogenously, we can simulate the process *a priori* to help guide this decision.

Another consideration is the behaviour of the backward pass. In Algorithm 6, we add a cut to all the ribs in each subproblem at every iteration. Another approach could be to add a cut to the closest rib, or to a subset of the ribs. However, we did not test these alternatives.

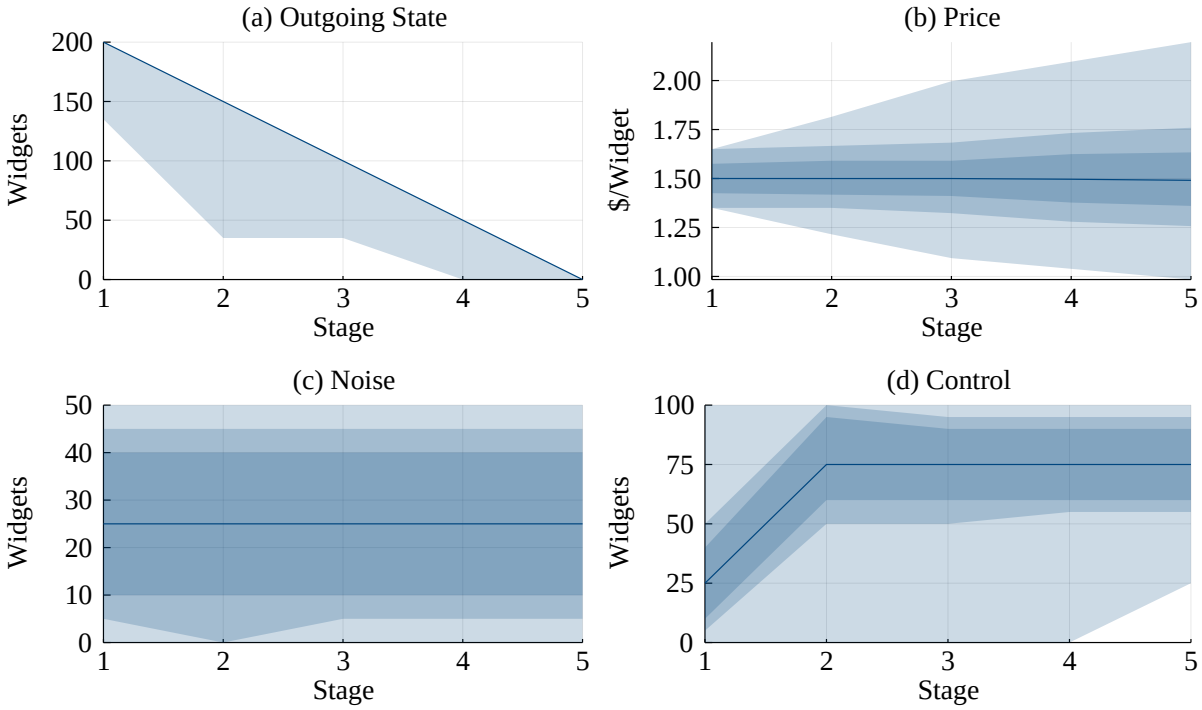


Figure 4.6: Simulated policy for widget producer.

The policy of adding a cut to all the ribs in each subproblem can lead to infeasibilities if the ribs in the next stage are not valid outer bounds for the price process from that point. For example, in the widget producer example above, the price process is bounded by ribs at \$0.75/widget and \$2.25/widget. If we place a rib at \$2.25/widget in the second stage, then on the backward pass the solution process will attempt to solve the third stage problem with a price of \$2.00/widget and \$2.50/widget. If the greatest price rib is only at \$2.25/widget in the third stage, then we cannot find a feasible γ. There are three potential fixes for this problem:

1. choose the position of the ribs in each stage such that the price is always feasible;
2. relax the requirement that $\gamma \geq 0$; or
3. project the price process back onto a fixed domain.

In the widget producer example, we chose option (1); however, it would be interesting to compare the trade-offs in solution quality between the three options.

However, aside from the convergence issues, other techniques that have been developed to improve the performance of traditional SDDP can be applied in this setting with minimal changes. For example, cut selection techniques can be applied to each rib independently and result in similar performance improvements. Nested risk measures can also be used via the same "change-of-probability" approach of Philpott et al. [159].

4.4 The dynamic interpolation method

In the static interpolation method, the ribs must be chosen *a priori*. Thus, without significant domain knowledge, the modeller may place ribs at points in the price dimension that are infrequently (if ever) visited. Moreover, the method does not scale well to high dimensions. This is analogous to the problem of discretising the state- and control-spaces faced by dynamic programming. SDDP avoids the need to discretise the state-space by refining the estimate of the cost-to-go function at a set of sampled points in the state-space. The analogue to this idea is to dynamically add new $\theta_{i,r}$ and $\gamma_{i,r}$ variables with associated cuts at sampled points in the price domain. However, since each new cost-to-go variable will only have one cut associated with it, the method of fixing the values for γ based on the two closest ribs will no longer result in a tight lower bound. This problem of finding a tight interpolation was solved by Baucke et al. [14] through a novel reformulation. Downward et al. [65] use this reformulation to develop the algorithm we refer to as the *dynamic interpolation method*. In this section, we explain their method and extend the class of problems it can be applied to.

4.4.1 Formulation

While explaining the dynamic interpolation method, we shall utilise the notation of Downward et al. [65]. To highlight the differences, consider the following subproblem (using our notation):

$$\begin{aligned} V_i(x_i, y_i, \omega_i) = \min_{u_i} \quad & C_i(\bar{x}_i, y_i', u_i, \omega_i) + \mathbb{F}_{j\in i^+\ \omega_j\in\Omega_j}\left[V_j\left(x_i', y_i', \omega_j\right)\right] \\ \text{s.t.} \quad & x_i' = T_i(\bar{x}_i, u_i, \omega_i) \\ & y_i' = Y_i(y_i, \omega_i) \\ & \bar{x}_i = x_i, \quad [\lambda_i] \\ & u_i \in U_i(x_i, \omega_i). \end{aligned}$$

Downward et al. [65] make a number of simplifications and modifications to this notation. In particular, they:

- denote x_t as the value of the *outgoing* state variable instead of the incoming state variable;
- assume the policy graph is linear, so $x_i = x_{t-1}$ and $x_i' = x_t$;
- combine the state and control variables into a single x_t variable;
- combine U_i and T_i, and assume they can be represented by the set of linear inequalities:

$$A_t^{\omega_t} x_t + a^{\omega_t} \geq x_{t-1};$$

- assume the Y_i function is linear, such that: $y_t = B_t^{\omega_t} y_{y-1} + b^{\omega_t}$; and
- only consider the risk-neutral case where $\mathbb{F} = \mathbb{E}$.

Under these modifications to the notation, each stage $t = 1, 2, \ldots, T-1$ can be expressed

as the subproblem:

$$\begin{aligned}\mathbf{SP}_t:\ V_t\left(x_{t-1}, y_{t-1}, \omega_t\right) = \min_{x_t, y_t}\quad & y_t^\top Q_t x_t + \mathbb{E}_{\omega_{t+1}\in\Omega_{t+1}}\left[V_{t+1}\left(x_t, y_t, \omega_{t+1}\right)\right] \\ \text{s.t.}\quad & A_t^{\omega_t} x_t + a^{\omega_t} \geq x_{t-1} \\ & y_t = B_t^{\omega_t} y_{t-1} + b^{\omega_t},\end{aligned}$$

where $A_t^{\omega_t}$ and $B_t^{\omega_t}$ are square matrices, Q_t is an appropriately dimensioned matrix, and a^{ω_t} and b^{ω_t} are vectors. Note that x_t and y_t are both state vectors for stage t. Moreover, $\mathbf{SP}_t$ is not a convex optimisation problem due to the bilinear term in the objective. However, observe that, given y_{t-1} and ω_t, y_t is uniquely determined.

In the final stage a variant of this problem is solved, where the cost-to-go is set to 0:

$$\begin{aligned}\mathbf{SP}_T:\ V_T\left(x_{T-1}, y_{T-1}, \omega_T\right) = \min_{x_T, y_T}\quad & y_T^\top Q_T x_T \\ \text{s.t.}\quad & A_T^{\omega_T} x_T + a^{\omega_T} \geq x_{T-1} \\ & y_T = B_T^{\omega_T} y_{T-1} + b^{\omega_T}.\end{aligned}$$

We define the expected cost-to-go function at the end of time period t (prior to observing the noise in period $t+1$) as: $\mathcal{V}_{t+1}(x_t, y_t) = \mathbb{E}_{\omega_{t+1}\in\Omega_{t+1}}\left[V_{t+1}\left(x_t, y_t, \omega_{t+1}\right)\right]$.

Downward et al. [65] show that $\mathcal{V}_{t+1}(x_t, y_t)$ is a saddle function. By a saddle function, we mean that $\mathcal{V}_{t+1}(x_t, y_t)$ is convex with respect to x_t, and concave with respect to y_t. Therefore, since SDDP relies on convexity, the traditional cutting plane formula cannot be used. The solution proposed by Downward et al. is to approximate the cost-to-go function $\mathcal{V}_{t+1}(x_t, y_t)$ by introducing a family of cuts with respect to x_t at a set of observed prices $\hat{y}_t^k$, and then interpolate between the cuts to estimate the cost-to-go function at prices that do not have a corresponding cut. Since $\mathcal{V}_{t+1}(x_t, y_t)$ is convex with respect to x_t, the cutting planes will form an outer approximation. Moreover, since $\mathcal{V}_{t+1}(x_t, y_t)$ is concave with respect to y_t, the interpolation will also form an outer approximation. Therefore, after K iterations, the approximated problem **AP-Dynamic**$_t^K$ is a valid lower bound for the true subproblem $\mathbf{SP}_t$. The approximated problem has the form:

AP-Dynamic$_t^K$:

$$\begin{aligned}V_t^K\left(x_{t-1}, y_{t-1}, \omega_t\right) = \min_{x_t, y_t, \theta_{t,k}}\ \max_{\gamma_k}\quad & y_t^\top Q_t x_t + \sum_{k=1}^{K} \gamma_k \theta_{t,k} \\ \text{s.t.}\quad & A_t^{\omega_t} x_t + a^{\omega_t} \geq x_{t-1} \\ & y_t = B_t^{\omega_t} y_{t-1} + b^{\omega_t} \\ & \sum_{k=1}^{K} \gamma_k \hat{y}_t^k = y_t \\ & \sum_{k=1}^{K} \gamma_k = 1 \\ & \gamma_k \geq 0, \quad \forall k \in \{1, 2, \ldots, K\} \\ & \theta_{t,k} \geq \alpha_t^k + {\beta_t^k}^\top x_t, \quad \forall k \in \{1, 2, \ldots, K\}.\end{aligned}$$

Note that this formulation is almost identical to **AP-Static**$_t^K$, the formulation in the static interpolation method. However, instead of having a set of cuts associated with each cost-to-go variable like **AP-Static**$_t^K$, **AP-Dynamic**$_t^K$ has one cut for each cost-to-go variable, and there may be multiple cost-to-go variables associated with the same price $\hat{y}_t^k$. Solving **AP-Dynamic**$_t^K$ is difficult due to the bilinear terms in the objective function. The solution identified by Baucke et al. [14] is to take the inner maximisation problem:

$$\begin{aligned}
\mathbf{P}: \quad \max_{\gamma} \quad & \sum_{k=1}^{K} \gamma_k \theta_{t,k} \\
\text{s.t.} \quad & \sum_{k=1}^{K} \gamma_k \hat{y}^k = y_t \quad [\mu] \\
& \sum_{k=1}^{K} \gamma_k = 1 \quad [\varphi] \\
& \gamma_k \geq 0 \qquad \forall k \in \{1, \ldots, K\},
\end{aligned}$$

and form the dual problem:

$$\begin{aligned}
\mathbf{D}: \quad \min_{\mu,\varphi} \quad & \mu^\top y_t + \varphi \\
\text{s.t.} \quad & \mu^\top \hat{y}^k + \varphi \geq \theta_{t,k}, \; k \in \{1, \ldots, K\},
\end{aligned}$$

This translates the problem of adding new variables to the primal formulation into the problem of adding constraints to the dual formulation. Moreover, Baucke et al. [14] show that, given a finite bound on the magnitude of μ so that $||\mu||_\infty \leq M$, the optimal objective value $\mu^\top y_t + \varphi$ is a valid lower bound for the expected future cost term $\mathcal{V}_{t+1}(x_t, y_t)$. We refer to M as the *Lipschitz Constant*. (See Baucke et al. [14] for details.)

Returning to **AP-Dynamic**$_t^K$, observe that given y_{t-1} and ω_t, y_t is uniquely defined. Therefore, we can calculate the value of y_t outside of the optimisation problem **AP-Dynamic**$_t^K$ and fix its value as a constant. Therefore, by calculating the value of y_t outside the optimisation problem, and by replacing the primal problem with the dual problem, we get:
SP-Dynamic$_t^K$ $(x_{t-1}, y_{t-1}, \omega_t)$:
Calculate $y_t = B_t^{\omega_t} y_{t-1} + b^{\omega_t}$ and set as constant, then solve:

$$\begin{aligned}
V_t^K(x_{t-1}, y_{t-1}, \omega_t) = \min_{x_t, \mu_t, \varphi_t} \quad & y_t^\top Q_t x_t + \mu_t^\top y_t + \varphi_t \\
\text{s.t.} \quad & A_t^{\omega_t} x_t + a^{\omega_t} \geq x_{t-1} \quad [\lambda_t] \\
& \mu_t^\top y_t^k + \varphi_t \geq \alpha_t^k + {\beta_t^k}^\top x_t \quad \forall k \in \{1, 2, \ldots, K\}.
\end{aligned}$$

We refer to the modified cut as a *saddle-cut*.

Modifications to the SDDP algorithm

The forward pass of the dynamic interpolation method is similar to the SDDP algorithm described in Chapter 1. The only difference is the process of calculating y_t outside of the main

optimisation problem, and passing the two states, x_t and y_t, forward. The backward pass is also very similar, except we use the saddle-cut:

$$\mu_t^\top y_t^k + \varphi_t \geq \alpha_t^k + {\beta_t^k}^\top x_t.$$

Pseudo-code is given in Algorithm 7 (using our notation).

Algorithm 7: The dynamic interpolation

set $K = 0$
while *not converged* **do**
  `/* Forward Pass */`
  set $x = x_R$, $y = y_R$, $\mathcal{S} = [\,]$
  sample i from R^+ according to the transition matrix Φ
  while $i^+ \neq \varnothing$ **do**
    sample ω_i from Ω_i
    solve **SP-Dynamic**$_i^K(x, y, \omega_i)$
    set $x = \hat{x}_i^{K+1} = x_i'$
    set $y = \hat{y}_i^{K+1} = y_i'$
    append $(i, \hat{x}_i^{K+1}, \hat{y}_i^{K+1})$ to the list $\mathcal{S}$
    sample new i from i^+ according to the transition matrix Φ
  end
  `/* Backward Pass */`
  for $(i, \hat{x}_i^{K+1}, \hat{y}_i^{K+1})$ *in* $reverse(\mathcal{S})$ **do**
    for $j \in i^+$ **do**
      for $\omega_j \in \Omega_j$ **do**
        solve **SP-Dynamic**$_j^K(\hat{x}_i^{K+1}, \hat{y}_i^{K+1}, \omega_j)$
        set $\bar{V}_{j,\omega_j}^K$ to the optimal objective value
        set $\bar{\lambda}_{j,\omega_j}^K$ to the value of λ_j in the optimal solution
      end
    end
    compute the changed distribution ξ for $\mathop{\mathbb{F}}_{j \in i^+;\, \omega_j \in \Omega_j} \left[\bar{V}_{j,\omega_j}^K\right]$
    set $\beta_i^{K+1} = \mathbb{E}_\xi\left[\bar{\lambda}_{j,\omega_j}^K\right]$
    set $\alpha_i^{K+1} = \mathbb{E}_\xi\left[\bar{V}_{j,\omega_j}^K\right] - \langle \beta_i^{K+1}, \hat{x}_i^{K+1} \rangle$
    add the cut $\langle \hat{y}_i^{K+1}, \mu_i \rangle + \varphi_i \geq \alpha_i^{K+1} + \langle \beta_i^{K+1}, x_i' \rangle$ to **SP-Dynamic**$_i^K$
  end
  increment K
end
algorithm.

Implementation in SDDP.jl

The dynamic interpolation method is implemented in the `SDDP.jl` package that we introduced in the previous chapter. It can be initialised in a similar manner to the `StaticPriceInterpolation` method we described previously. However, there are three notable exceptions.

First, we no longer need to specify a fixed set of `rib_locations`. Instead, it is necessary to give a `min_price` and `max_price`. Second, for the reasons that we outline at the end of this section, the dynamic method does not (yet) support cut selection. Therefore, there is no `cut_oracle` keyword. Third, we need to specify the Lipschitz constant M via the `lipschitz_constant` keyword. Below we give the code to initialise the dynamic interpolation method for the widget producer example.

```
function buildvaluefunction(stage, markovstate)
    R = log.([0.9, 0.95, 0.99, 1.0, 1.01, 1.05, 1.1])
    DynamicPriceInterpolation(
        dynamics           = (y, noise) -> y + noise,
        initial_price      = log(1.50),
        min_price          = log(1.50) + stage * minimum(R),
        max_price          = log(1.50) + stage * maximum(R)
        noise              = DiscreteDistribution(R),
        lipschitz_constant = 10
    end
end
m = SDDPModel(
    ... some arguments omitted ...
    value_function = buildvaluefunction
        )
) do sp, t
    ... some lines omitted ...
end
```

The stage-objective is identical to the one described in the static interpolation section. However, note that the stage-objective of the widget example, $-e^{y_t}u_t$, is not of the form $y_t^\top Q_t x_t$. Therefore, the result of Downward et al. [65] does not ensure that the dynamic interpolation method will converge to the optimal solution. To overcome this limitation, we extend the convergence result of Downward et al. [65] to show that the widget producer example can be solved using the dynamic interpolation method.

4.4.2 Extended convergence proof for the dynamic interpolation method

We consider an extension to the convergence result of Theorem 2 in Downward et al. [65] to the case where the stage-objective is $f_t^{\omega_t}(y_t)^\top x_t$ instead of $y_t^\top Q_t x_t$, where $y_t \in \mathbb{R}^M$, $x_t \in \mathbb{R}^N$, and $f_t^{\omega_t} : \mathbb{R}^M \to \mathbb{R}^N$. In addition, we require the following assumptions in each stage $t = 1, 2, \ldots, T$.

(A1) Given a fixed ω_t, $f_t^{\omega_t}(y_t)$ is elementwise concave with respect to y_t. That is, for $y_t^{(1)}$, $y_t^{(2)} \in \mathbb{R}^M$, $\lambda \in [0, 1]$: $f_t^{\omega_t}(\lambda y_t^{(1)} + (1-\lambda) y_t^{(2)}) \geq \lambda f_t^{\omega_t}(y_t^{(1)}) + (1-\lambda) f_t^{\omega_t}(y_t^{(2)})$.

(A2) $x_t \geq 0$.

Under these assumptions, subproblems with the modified objective function have the form: $\mathbf{SP}_t(x_{t-1}, y_{t-1}, \omega_t)$:

$$\begin{aligned} V_t(x_{t-1}, y_{t-1}, \omega_t) = \min_{x_t} \quad & f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}))^\top x_t + \mathbb{E}_{\omega_{t+1}}\left[V_{t+1}\left(x_t, g_t^{\omega_t}(y_{t-1}), \omega_{t+1}\right)\right] \\ \text{s.t.} \quad & A_t^{\omega_t} x_t + a^{\omega_t} \geq x_{t-1} \\ & x_t \geq 0, \end{aligned}$$

where the transition function $y_t = g^{\omega_t}(y_{t-1}) = B_t^{\omega_t} y_{t-1} + b^{\omega_t}$.

To show that the algorithm still converges under this modification, we need to show that the expected cost-to-go function $\mathcal{V}_{t+1}(x_t, y_t) = \mathbb{E}_{\omega_{t+1} \in \Omega_{t+1}}\left[V_{t+1}\left(x_t, y_t, \omega_{t+1}\right)\right]$ remains a saddle function. Before we begin, we prove a simple lemma.

Lemma 1. *If $f : \mathbb{R}^M \to \mathbb{R}$ is a concave function, A is an $M \times N$ matrix, and b is an $M \times 1$ vector, then $f(Ax + b)$ is a concave function with respect to x.*

Proof. Consider $x_1, x_2 \in \mathbb{R}^N$, $\lambda \in [0, 1]$:

$$\begin{aligned} f(A(\lambda x_1 + (1-\lambda)x_2) + b) &= f(\lambda(Ax_1 + b) + (1-\lambda)(Ax_2 + b)) \\ &\geq \lambda f(Ax_1 + b) + (1-\lambda) f(Ax_2 + b). \end{aligned}$$

□

We now prove the concavity of $V_{t+1}(x_t, y_t, \omega_{t+1})$ with respect to y_t using induction.

Lemma 2. *Under assumptions A1 and A2, $V_{t+1}(x_t, y_t, \omega_{t+1})$ is concave with respect to y_t, for all $t \in \{1, \ldots, T-1\}$.*

Proof. We will prove this by induction for $V_t(x_{t-1}, y_{t-1}, \omega_t)$. First, we assume that the cost-to-go function in stage $t+1$, $V_{t+1}(x_t, y_t, \omega_{t+1})$, is concave with respect to y_t. Then, by Lemma 1, we have that $V_{t+1}(x_t, g^{\omega_t}(y_{t-1}), \omega_{t+1})$ is concave with respect to y_{t-1}.

Recall that the stage problem is:

$$\begin{aligned} V_t(x_{t-1}, y_{t-1}, \omega_t) = \min_{x_t} \quad & f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}))^\top x_t + \mathbb{E}_{\omega_{t+1}}\left[V_{t+1}\left(x_t, g_t^{\omega_t}(y_{t-1}), \omega_{t+1}\right)\right] \\ \text{s.t.} \quad & A_t^{\omega_t} x_t + a^{\omega_t} \geq x_{t-1} \\ & x_t \geq 0. \end{aligned}$$

Given fixed values of x_{t-1} and ω_t, consider two values of y_{t-1}: $y_{t-1}^{(1)}$ and $y_{t-1}^{(2)}$, with optimal solutions $x_t^{(1)}$ and $x_t^{(2)}$, respectively.

Now suppose we had a third value $y_{t-1}^{(3)} = (1-\lambda)y_{t-1}^{(1)} + \lambda y_{t-1}^{(2)}$, with a corresponding optimal solution $x_t^{(3)}$. For concavity, we require:

$$V_t(x_{t-1}, y_{t-1}^{(3)}, \omega_t) \geq (1-\lambda) V_t(x_{t-1}, y_{t-1}^{(1)}, \omega_t) + \lambda V_t(x_{t-1}, y_{t-1}^{(2)}, \omega_t).$$

Therefore, this would imply:

$$f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(3)}))^\top x_t^{(3)} + \mathbb{E}_{\omega_{t+1}}\left[V_{t+1}\left(x_t^{(3)}, g_t^{\omega_t}(y_{t-1}^{(3)}), \omega_{t+1}\right)\right] \geq$$

$$(1-\lambda)\left(f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(1)}))^\top x_t^{(1)} + \mathbb{E}_{\omega_{t+1}}\left[V_{t+1}\left(x_t^{(1)}, g_t^{\omega_t}(y_{t-1}^{(1)}), \omega_{t+1}\right)\right]\right) +$$

$$\lambda\left(f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(2)}))^\top x_t^{(2)} + \mathbb{E}_{\omega_{t+1}}\left[V_{t+1}\left(x_t^{(2)}, g_t^{\omega_t}(y_{t-1}^{(2)}), \omega_{t+1}\right)\right]\right).$$

With a view to contradiction, suppose that:

$$f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(3)}))^\top x_t^{(3)} + V_{t+1}\left(x_t^{(3)}, g_t^{\omega_t}(y_{t-1}^{(3)}), \omega_{t+1}\right) <$$

$$(1-\lambda)\left(f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(1)}))^\top x_t^{(1)} + V_{t+1}\left(x_t^{(1)}, g_t^{\omega_t}(y_{t-1}^{(1)}), \omega_{t+1}\right)\right) + \tag{4.2}$$

$$\lambda\left(f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(2)}))^\top x_t^{(2)} + V_{t+1}\left(x_t^{(2)}, g_t^{\omega_t}(y_{t-1}^{(2)}), \omega_{t+1}\right)\right),$$

for some ω_{t+1}.

From the concavity of $f_t^{\omega_t}(y_t)$ (by assumption A1), and since $x_t^{(3)} \geq 0$ (by assumption A2), we have that:

$$(1-\lambda)\left(f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(1)}))^\top x_t^{(3)}\right) + \lambda\left(f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(2)}))^\top x_t^{(3)}\right) \leq f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(3)}))^\top x_t^{(3)}.$$

In addition, from the concavity of $V_{t+1}(x_t, g^{\omega_t}(y_{t-1}), \omega_{t+1})$ with respect to y_{t-1}, we have:

$$(1-\lambda)\left(V_{t+1}\left(x_t^{(3)}, g_t^{\omega_t}(y_{t-1}^{(1)}), \omega_{t+1}\right)\right) + \lambda\left(V_{t+1}\left(x_t^{(3)}, g_t^{\omega_t}(y_{t-1}^{(2)}), \omega_{t+1}\right)\right) \leq$$

$$V_{t+1}\left(x_t^{(3)}, g_t^{\omega_t}(y_{t-1}^{(3)}), \omega_{t+1}\right).$$

Therefore, from Eq. 4.2 and the above inequalities we have:

$$(1-\lambda)\left(f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(1)}))^\top x_t^{(3)} + V_{t+1}\left(x_t^{(3)}, g_t^{\omega_t}(y_{t-1}^{(1)}), \omega_{t+1}\right)\right) +$$

$$\lambda\left(f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(2)}))^\top x_t^{(3)} + V_{t+1}\left(x_t^{(3)}, g_t^{\omega_t}(y_{t-1}^{(2)}), \omega_{t+1}\right)\right) <$$

$$(1-\lambda)\left(f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(1)}))^\top x_t^{(1)} + V_{t+1}\left(x_t^{(1)}, g_t^{\omega_t}(y_{t-1}^{(1)}), \omega_{t+1}\right)\right) +$$

$$\lambda\left(f_t^{\omega_t}(g_t^{\omega_t}(y_{t-1}^{(2)}))^\top x_t^{(2)} + V_{t+1}\left(x_t^{(2)}, g_t^{\omega_t}(y_{t-1}^{(2)}), \omega_{t+1}\right)\right).$$

This is obviously a contradiction, since $x_t^{(3)}$ cannot give a lower cost than the optimal solutions $x_t^{(1)}$ and $x_t^{(2)}$; therefore, $V_t(x_{t-1}, y_{t-1}, \omega_t)$ is concave with respect to y_{t-1}.

Finally, in the terminal stage T, $V_{T+1}(x_T, y_T, \omega_{T+1}) = 0$ and is therefore concave. Thus, applying the induction, $V_{t+1}(x_t, y_t, \omega_{t+1})$ is concave with respect to y_t for all $t \in \{1, \ldots, T-1\}$. □

Lemma 3. *Under assumptions A1 and A2, $\mathcal{V}_{t+1}(x_t, y_t)$ is concave with respect to y_t, for all $t \in \{1, 2, \ldots, T-1\}$.*

Proof. By Lemma 2, we have that $V_{t+1}(x_t, y_t, \omega_{t+1})$ is concave with respect to y_t given fixed values of x_t and ω_{t+1}, for all $t \in \{1, 2, \ldots, T-1\}$. It follows that:

$$\mathcal{V}_{t+1}(x_t, y_t) = \mathbb{E}_{\omega_{t+1} \in \Omega_{t+1}} \left[V_{t+1}\left(x_t, y_t, \omega_{t+1}\right) \right]$$

is concave with respect to y_t, for all $t \in \{1, 2, \ldots, T-1\}$. □

Lemma 4. *$\mathcal{V}_{t+1}(x_t, y_t)$ is convex with respect to x_t, for all $t \in \{1, 2, \ldots, T-1\}$.*

Proof. We can directly apply Lemma 1 of Downward et al. [65] since y_t is still uniquely determined for a fixed y_{t-1} and ω_t. □

We now prove the main result that the dynamic interpolation algorithm will still converge almost surely under our new assumptions.

Theorem 1. *Under assumptions A1 and A2, Algorithm 7 converges with probability 1 to an optimal solution of our modified problem in a finite number of iterations.*

Proof. By Lemma 3 and Lemma 4, $\mathcal{V}_{t+1}(x_t, y_t)$ is a saddle function for all $t \in \{1, 2, \ldots, T-1\}$. Since the remainder of the proof of Theorem 2 in Downward et al. [65] does not utilise the transition function (i.e. $y_t = B_t^{\omega_t} y_{t-1} + b_t^{\omega_t}$) or the stage-objective (i.e. $f_t^{\omega_t}(y_t)^\top x_t$), the algorithm will still converge almost surely in a finite number of iterations. □

4.4.3 Results

Observe that the cost-to-go function for the widget producer example is a saddle function since $u_t \geq 0$, the transition function $y_t = y_{t-1} + \omega_t$ is linear, and $f_t(y_t) = -e^{y_t}$ is concave. This implies that the first-stage objective function value of the static interpolation formulation for the widget producer is a lower bound. In addition, since the cost-to-go function is a saddle function, we can solve the widget producer example using the dynamic interpolation algorithm. The subproblems are:

SP-Dynamic$_t^K$ (x_t, y_t, ω_t):

Calculate $y_t' = y_t + \omega_t^p$ and set as a constant in the objective, then solve:

$$\begin{aligned} V_t\left(x_t, y_t, \omega_t\right) = \min_{u_t, x_t', \mu, \varphi} \quad & -e^{y_t'} \times u_t + y_t' \mu + \varphi \\ \text{s.t.} \quad & x_t' = \bar{x}_t - u_t + \omega_t \\ & \bar{x}_t = x_t, \quad [\lambda_t] \\ & u_t \in [0, 100] \\ & x_t' \in [0, 350] \\ & \hat{y}^k \mu + \varphi \geq \alpha_t^k + \beta_t^k x_t' \quad k \in \{1, \ldots, K\}, \\ & || \mu ||_\infty \leq M. \end{aligned}$$

Note that in this formulation, we do not define a set of ribs *a priori*. Instead, cuts are placed in the price dimension at sampled points (i.e. $\hat{y}^k$) and the interpolation happens via the embedded dual problem. We also bound the magnitude of the μ variables to prevent the subproblems from being unbounded. We call M the Lipschitz constant. (See Baucke et al. [14] for details.)

In Figure 4.7, we plot the convergence of the lower bound against the number of iterations. As the number of iterations increases, the dynamic interpolation bound converges towards the lower bound that we obtained using the static interpolation method with 20 ribs ($-\$487.59$). This confirms that static and dynamic interpolation methods converge to the same optimal solution. Note that we do not compare the computational performance (in terms of the number of iterations or solution time) here; that is analysed in Section 4.5 below. The policy is identical to that observed in Figure 4.6 so we omit its description.

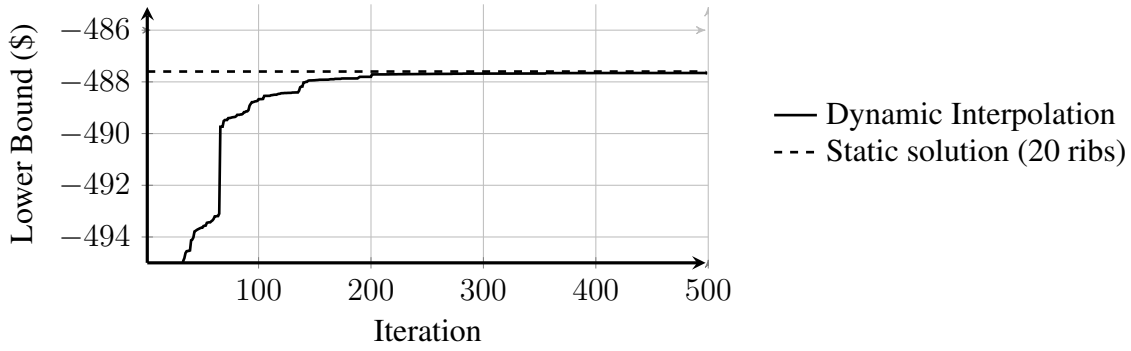


Figure 4.7: Convergence of the lower bound against the number of iterations using the dynamic interpolation method.

4.4.4 Practical considerations

Convergence As claimed by Pereira and Pinto [153], SDDP is able to avoid the "curse of dimensionality" by only refining the approximation of the cost-to-go function at portions of the state-space that are likely to be visited. If a point in the state-space has not been visited a large number of times in the iteration phase of the algorithm, the policy may be sub-optimal. In most cases, the agent can use their controls to regulate the trajectory of the system through the state-space. Thus, the system may visit parts of the state-space a large number of times regardless of the sequence of random variables observed. In contrast, the price-states in the dynamic interpolation algorithm evolve independently of the controls. Since the probability of sampling a sequence of very high, or very low prices is small, the cost-to-go function approximation at the edges of the price domain for a given stage may be poor. When the policy is simulated, this can result in a sub-optimal sequence of controls. This issue can be resolved by carrying out more iterations, but this becomes computationally expensive.

Readers should note that this issue is also present in the SDDP algorithm, but it is exacerbated in the dynamic interpolation algorithm by the uncontrollable price-states.

Cut selection During the course of the SDDP algorithm, thousands of valid cutting planes are discovered. However, many of those cuts may be dominated by cuts discovered in later iterations. Therefore, the performance of the algorithm can be improved by only including a subset of the cuts in each stage problem (we referred to this in Chapter 2 as *cut selection*). In Section 2.4, we discussed a number of heuristics that have been proposed for selecting cuts in the SDDP algorithm. However, due to the different stage problem in the dynamic interpolation algorithm compared with the traditional SDDP formulation, heuristics such as Level One cut selection [55] are no longer directly applicable. However, there may be close analogues that extend to the new formulation. One class of heuristics that is still directly applicable is based around scoring the number of times that a cut was active in the solution (i.e. the dual of the constraint was non-zero), and removing those cuts that are used infrequently, or have not been used for a given number of iterations [142]. However, since the price-state evolves independently, there may be areas which the solution visits infrequently. Therefore, discarding cuts in these areas may exacerbate the convergence issue described above.

The choice of an efficient cut selection heuristic for the dynamic interpolation method remains an open question.

Initialisation In the initial iterations of the SDDP algorithm, it is necessary to bound the cost-to-go variable by some large value. In contrast, the dynamic interpolation algorithm requires a bound on the cost-to-go, as well as a bound on the subgradient of the cost-to-go function with respect to the price-state (the Lipschitz constant α in Baucke et al. [14]). The choice of these bounds can have a large effect on the algorithm's performance. Moreover, whilst the practitioner may have an intuitive understanding of an appropriate bound on the cost-to-go variable, the bound on the subgradient is less intuitive.

To improve performance, a different Lipschitz constant M should be used for each dimension of the vector μ. In addition, it is useful to provide an estimate for the domain of the price-state y_t. This enables initial constraints based on the bound of the cost-to-go to be added at each of the vertices of the bounding box for price. However, since this leads to 2^N constraints being added, this is only practical for low-dimensional price processes (i.e. $N < 5$).

Risk Although not mentioned by Downward et al. [65], the algorithm can naturally be extended to solve the risk-averse case via the same "change-of-probability" approach of Philpott et al. [159], which we outlined in Section 1.3.

4.5 Hybrid methods

Observe that the static interpolation method is equivalent to the dynamic interpolation that samples a fixed set of price-states on the backward pass instead of the one observed on the forward

pass. Therefore, a hybrid method could be implemented that consists of the dynamic interpolation method, augmented by a set of prices to visit on each backward pass in addition to the observed price. This has the benefit of guaranteeing convergence (since we still eventually sample every price-state), whilst providing better sampling properties at the edge of the price domain. It also allows the model to extrapolate the cost-to-go function outside the fixed set of ribs and can scale to multiple dimensions. However, this method is hampered by the requirement to appropriately choose the location and number of the fixed set of ribs. In addition, the issue of an appropriate cut selection heuristic is still unsolved. As such, any performance improvement would be problem-specific.

Another option is to run the static interpolation method for a truncated number of iterations, and then switch to the dynamic interpolation method. The cuts discovered during the static interpolation phase can be used as an initial approximation of the cost-to-go function for the dynamic interpolation phase. We call this method the *hot-start* method. There are two key considerations with the hot-start method:

1. the number of ribs in the initial phase; and
2. the number of iterations in the initial phase.

If more ribs are used in the initial phase, then the interpolation error is minimised. However, each iteration is more computationally intensive. If more iterations are conducted, then the bound will converge closer to the true bound. However, if the bound has stalled, or is only making slow progress, it may be beneficial to change to the dynamic interpolation phase.

In Figure 4.8, we plot the lower bound of the widget producer example against total solution time for three methods: (1) the dynamic interpolation method; (2) the static interpolation method with five ribs; and (3) the "hot-start" method using five ribs in the initial phase. We terminate the static phase of the hybrid method using the bound stalling stopping rule discussed in Section 2.3. For this example, we terminated the static interpolation phase when the bound failed to improve after five iterations.

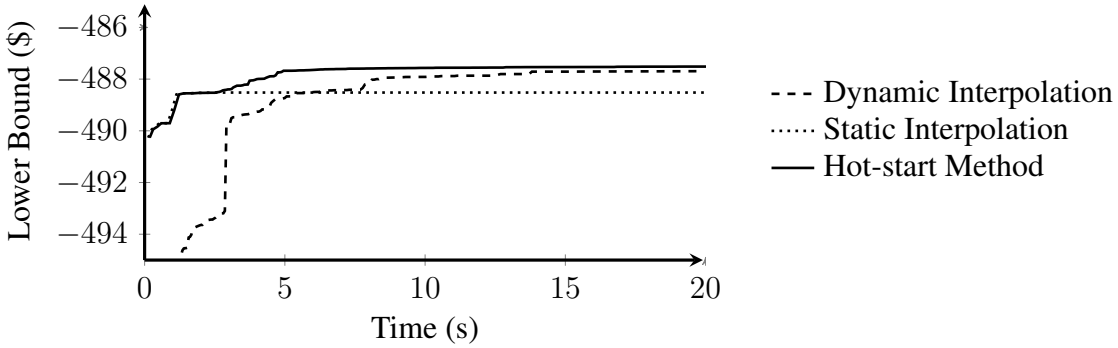


Figure 4.8: Convergence of the lower bound over solution time for the dynamic (dashed), static (dotted), and hot-start (solid) interpolation methods.

Because the static interpolation method samples a disperse range of price-states, the lower bound is better than the dynamic interpolation method in the first five seconds. However, be-

cause of the discretisation error, the static interpolation method does not converge, and the lower bound stalls at approximately $-\$488.5$. The hot-start method is equivalent to the static method in the initial iterations by design. However, after the static interpolation phase has stalled for five iterations, the hot-start method switches to the dynamic interpolation phase. It then quickly converges to the bound we obtained previously of $-\$487.59$. Note that after 20 seconds, the dynamic interpolation method has not yet converged.

4.6 Summary

This chapter examined extensions to SDDP to enable the solution of multistage stochastic optimisation problems with stagewise-dependent noise in the objective. We compared two methods. The *static interpolation method* discretises *a priori* the stagewise-dependent process into a number of *ribs* and constructs an approximation of the cost-to-go function at each rib. Due to the discretisation, the method may not converge to the optimal solution. The second method, which we refer to as the *dynamic interpolation method*, adds saddle-cuts at points in the stagewise-dependent process sampled on the forward pass. Because the saddle-cuts are placed at sampled points, Downward et al. [65] show that the method will converge an optimal solution almost surely in a finite number of iterations.

We extended the convergence result of Downward et al. [65] to show that a broader class of problems can be solved using the dynamic interpolation method. This extension allows some multiplicative auto-regressive price processes to be included in the model.

We also applied both the static and dynamic interpolation methods to a simple example and provided some advice for practitioners implementing and using the methods. Finally, we explored a hybrid method that demonstrated improved computational performance compared with the static and dynamic interpolation methods.

In Chapter 9 of this thesis, we apply the dynamic interpolation method to a dairy farming model with milk price uncertainty.

Chapter 5

Conclusion

5.1 Main results and contributions

In Part I of this thesis, we discussed the theory and computation of multistage stochastic optimisation. In particular, we focused on stochastic dual dynamic programming (SDDP) as a solution technique. The main contribution of Part I is the two software libraries we have developed for multistage stochastic optimisation:

- `SDDP.jl`, a Julia library for stochastic dual dynamic programming; and
- `DynamicProgramming.jl`, a Julia library for stochastic dynamic programming.

In addition to those libraries, we made a number of theoretical contributions to the field.

- We proposed a new framework for communicating a class of multistage stochastic optimisation problems: the *policy graph*.

  In particular, the policy graph can model discrete-time, multistage stochastic optimisation problems with nested risk measures and finite discrete stagewise-independent random variables.

- We conducted a literature review of the current state-of-the-art for stochastic dual dynamic programming.

  In general, the computational improvements can be summarised into two categories: improvements that reduce the required number of iterations before convergence, and improvements that reduce the computation time of each iteration. We provided a summary of the improvements in Figure 2.2.

- We found that SDDP is susceptible to numerical instability.

  In one common case, solvers allow the primal solution of a state variable to be outside the exact domain by some tolerance ϵ (e.g. $x'_t \in [l - \epsilon, u + \epsilon]$). When x'_t is set as the value of

x_{t+1} in the forward pass, this can result in infeasibilities. This can be fixed by projecting the primal solution value back onto the exact domain in the forward pass.

- We found that dual degeneracy can lead to sub-optimal simulations of the optimal policy.

  This has the troubling implication that simulating a policy that has converged to an optimal solution can result in a sub-optimal sequence of controls.

- We described an algorithm for stochastic dual dynamic programming on an acyclic policy graph.

  This is a novel contribution as previous literature has focused on broad sub-classes such as linear policy graphs [153], Markovian policy graphs [85, 160, 173], and special cases such as models with a random numbers of stages [95].

- We described the practical implementation of the static and dynamic interpolation methods.

  In particular, we raised a number of problems associated with the convergence of both methods.

- We extended the convergence result of Downward et al. [65] to a broader class of models.

  In particular, we showed that some multiplicative auto-regressive processes can be incorporated into the model. Such models are common in financial applications [70].

Summarised into a single finding, this half of thesis suggests that SDDP is a powerful tool for solving multistage stochastic optimisation problems. However, it has a number of limitations that are often overlooked in the literature. When analysing the results of a policy produced by SDDP, careful consideration is needed to avoid misinterpreting the solution.

5.2 Future work

We began Part I by describing how the stochastic optimisation community is fragmented. Different communities use different terminology, and they do not routinely publish or share their optimisation codes. This standard of "publication only" research has slowed progress in the field as it limits the ability to rapidly build upon the findings of others [152].

For example, we saw in Chapter 2 how there are many different computational improvements that have been suggested to SDDP. Many of the reported computational improvements are likely to be problem-specific, and it can be difficult to isolate techniques that deliver large benefits across a broad range of problems (e.g. cut selection), against those that may, at best, offer a small improvement (e.g. scenario incrementation). Moreover, since the overhead of developing and maintaining a state-of-the-art implementation of SDDP is large, practitioners do not compare new methods against the existing state-of-the-art or on a wide variety of problems.

We also described a number of numerical issues and other bugs that we encountered when developing `SDDP.jl`. Through experience, we have learned that developing an efficient, generic, and correct implementation of multistage stochastic optimisation software such as SDDP is difficult. Having each researcher independently re-invent this wheel is unnecessary. Compounding the problem, since each researcher's implementation is different, models are not transferable between implementations. Therefore, it is difficult to replicate the findings in published papers.

To address these concerns, we propose the following research directions.

- Develop an open-source suite of tools for solving multistage stochastic optimisation problems.
- Construct a large benchmark set of multistage stochastic optimisation test problems.
- Perform a computational analysis of the different solution methods and computational improvements suggested in the literature.

In this thesis, we have attempted to remedy some of these issues by publishing `SDDP.jl` and `DynamicProgramming.jl`, as well as the source-code for MOO, POWDER-I, and POWDER-II (the models we construct in Part II of this thesis). Any ambiguity in the textual descriptions of these models can be resolved by examining the source-code.

In addition to the three broad topics outlined above, we suggest the following as more specific directions one could take.

- Extend the SDDP algorithm to cyclic policy graphs.

  We noted in Chapter 2, that the SDDP algorithm requires an acyclic policy graph. Indeed, we neglected to even introduce cyclic policy graphs in Chapter 1. However, it should be apparent to readers that a policy graph can include cycles in its definition. Some work has been done on infinite horizon SDDP (e.g. Baucke [12], Nannicini et al. [142]); however, we suggest extending the SDDP algorithm to acts on a general policy graph with an arbitrary number of cycles.

- Explore the deterministic convergence methods of Baucke et al. [14] and Leclère et al. [124].

  We noted in Chapter 2 that the SDDP algorithm has no clear method of termination. If they can be made computationally efficient, deterministic upper bounds, such as those proposed by Baucke et al. [14] and Leclère et al. [124], resolve this termination question. In addition, they may even reduce solution times by directing the solution search in ways that are not provided by the current Monte Carlo simulation approach.

- Explore techniques from Benders decomposition for generating strong cuts when the primal subproblem is degenerate.

A review paper by Rahmaniani et al. [172] discusses a number of computational improvements that have been made to the Benders decomposition algorithm since its inception by Benders [25]. However, aside from cut selection, few of these improvements have flowed into the SDDP literature. It would be interesting to adapt many of the techniques presented by Rahmaniani et al. [172] to the SDDP setting and investigate their computational benefit.

- Explore cut selection heuristics for the price interpolation methods.

  We noted in Chapter 4 that the question of an appropriate cut selection heuristic for the dynamic interpolation method is unresolved. Given that it is common to see an order of magnitude improvement in solution times as a result of cut selection in the traditional SDDP algorithm, this feature should be given a high priority to investigate.

- Explore hybrid methods for price interpolation.

  Finally, we ended Chapter 4 with the description of a hybrid interpolation method that demonstrated better computational performance than either the static or dynamic interpolation method. It would be interesting to study these hybrid methods in more detail in order to further improve the computational performance of the interpolation methods.

Part II

Applications in the Dairy Industry

Chapter 6

Introduction

Uncertainty in the agricultural supply sector has received increasing attention from researchers and practitioners in recent years [32]. This is unsurprising given the inherent uncertainty in all steps of the agricultural supply chain. In this half of the thesis, we focus on the first step in the supply chain for dairy products – namely, the dairy farmer. Such farmers face two main types of uncertainty: production uncertainty as a result of weather conditions, and price uncertainty from international markets [71]. Ignoring these uncertain effects can lead to poor farming decisions. In addition, dairy farmers are likely to be risk-averse due to the serious downside risks they face. These risks range from financial (they may accrue too much debt and go into bankruptcy) to biological (a lack of feed can lead to malnourished cows and animal-welfare issues). Furthermore, farms may be located in disparate geographic locations, and even farms that are located in the same local area may have differences in topology and soil type between their paddocks. Therefore, it is necessary to model conditions on an individual farm basis in order to produce meaningful insight for the farmer. Although our models are generally applicable to a wide-class of outdoor, pastoral farms, for this thesis we calibrate and test them in a New Zealand setting.

In New Zealand, dairy farming forms a large part of the economy, and in the 2016 season it was responsible for 29% of New Zealand's export earnings [10]. In addition, despite producing only 3% of the total global production volume, New Zealand exports 96% of its domestic production [186]. As a consequence, it is the largest player in the international dairy commodity market with around 28% of international exports [186]. These factors make the operation of a New Zealand dairy farm a critical component in the global dairy supply chain.

Established farmers in New Zealand largely rely on experience, low debt, efficiency, and a financial buffer to help them survive economic downturns [190]. However, as dairy markets liberalise and new environmentally focused domestic regulations (such as a proposed charge on the commercial use of water [39]) are introduced, relying on previous experience may result in sub-optimal decision-making. This decision-making is made up of two main components: short-term operational planning, such as choosing how much feed to give the cows each day; and long-term strategic planning, such as choosing the number of cows to farm. These decisions are sequential and subject to uncertainty.

In the first part of this thesis, we addressed the theory and computation of problems involving sequential decision-making under uncertainty, which we called *multistage stochastic optimisation* problems. Now, in this half of the thesis, we apply the tools and techniques developed in Part I to the multistage stochastic optimisation problem faced by pastoral dairy farmers. We call this problem *the pastoral dairy farmer problem*, and we describe it in the next section. (Our description assumes no prior knowledge of pastoral dairy farming. Readers familiar with the industry may wish to skip the next section.)

6.1 The pastoral dairy farmer problem

Dairy farms raise cows for the purpose of milk production. These are different to farms which raise cows[1] for the purpose of meat production. In this half of the thesis, we examine *pastoral* dairy farms, which rely on grass as their primary source of feed for the cows and are predominant in places like New Zealand, Ireland, and Argentina [119]. We do not consider the indoor farming systems prevalent in the United States and Europe.

A group of cows on a farm is referred to as a *herd*. Within a herd, there will be cows of different ages, breeds, and sizes. However, in this thesis, we overlook this complication and assume that all of the cows within a herd are identical. In addition, some farms may have multiple herds. In this thesis, we only consider a single herd. On a farm, the number of cows per hectare is termed the *stocking rate*. The stocking rate is an important strategic level decision for a farmer as it has consequences for future decision-making.

The predominant crop grown on pastoral farms is ryegrass (*Lolium* L.). However, other crops such as clover (*Trifolium repens*), lucerne (*Medicago sativa*), and chicory (*Cichorium intybus*) may also be grown. Since these crops are not grasses, the literature refers to the collective crops grown on pastoral dairy farms as *herbage*. However, to avoid confusing readers who may be unfamiliar with the dairy industry, we use the term *grass*. The land that is used to grow grass is referred to as *pasture*.

Pasture is subdivided into fenced areas called paddocks. Paddocks are typically grazed in sequence by the herd over a multi-week rotation. In most circumstances, the herd will graze a paddock during the day and a different paddock during the night. After grazing, the paddock is rested for a period of time (typically 20–30 days, although it varies during the year; referred to as the *rotation length*) before it is grazed again. This creates a distribution of grass height over the farm at any point in time. We refer to the mass of grass per unit area as the *pasture cover* (measured in kg/ha). In this thesis, we measure the pasture cover in terms of kilograms of dry matter[2] above ground level.[3] Furthermore, we make the simplifying assumption that all paddocks are identical at all times. Therefore, we represent the quantity of grass on the farm

[1] Since *cow* refers to the female gender, we really mean cattle (i.e. male and female animals) here.

[2] The mass of grass excluding the water content.

[3] Since cows will not eat all of the grass to ground level, other conventions exist, such as measuring the quantity of grass above a set height.

by the average pasture cover. As a consequence, we also exclude the ability of the farmer to manage the rotation length. The grass on the farm grows at an uncertain rate due to the weather, as well as the current level of pasture cover (since if there is no grass, none can grow).

Since pastoral farms rely on grass growth, which is driven by sunlight and rainfall, their operations are strongly tied to the seasons. These operations can be broken down into yearly cycles, each of which we term a *season*. In this thesis we model the season using a *spring calving* pattern that is common in New Zealand.[4] The season begins around August (in New Zealand) with each cow giving birth. This triggers each cow to begin producing milk. We call cows that are producing milk *lactating* cows and the process of producing milk *lactation*. In a real herd, there will be some distribution of calving dates; however, since we assume that all the cows are identical, we assume that all of the cows calve on the same day. After giving birth, most farmers will milk the cows twice-a-day. (Other systems exist, such as once-a-day milking, and even one milking every 18 hours. Our models could easily be modified to handle such cases.)

One important indicator of animal health is the Body Condition Score (BCS), a subjective assessment of the proportion of body fat in a cow [177]. In this thesis, we use the United States system of a five-point scale, where 1 represents an emaciated cow, and 5 represents an obese cow. A typical range is between 2.5 and 3.5. At the start of the season (i.e. when the cow gives birth), a Body Condition Score at the top of this range is associated with improved profitability and health during the season [177].

Each day, the farmer must make a set of decisions regarding the operation of their farm. The decisions they make today impact the decisions they can make tomorrow. There are three main operational decisions that a farmer must make each day.

First, they must decide the quantity of grass to feed the herd. However, farmers cannot force cows to graze pasture. Instead, farmers place the herd in a paddock with a given pasture cover. In response, the herd consumes some proportion of the offered quantity based on the time of year, quality of grass, and other factors. The farmer can control the quantity of grass to offer by paddock selection (for size and pasture cover) or by subdividing a paddock using temporary fencing. In this thesis we ignore these complications and assume that the farmer can offer any quantity of grass.

Second, each day the farmer must decide the quantity of *supplement* to feed the herd. Supplement is feed that is not grown on the farm but is imported. Common supplementary feeds in New Zealand include palm kernel, maize, and oats [47]. The two main uses of supplement are to boost milk production and manage temporary feed shortages [59]. We refer to the percentage of total feed that is imported over the season as the *supplementation intensity*. Choosing this intensity level is an important strategic level decision for the farmer.

Third, one of the most important decisions a farmer must make is when to *dry-off* their

[4] Some farmers adopt different calving patterns such as autumn calving in order to produce milk during the New Zealand winter, thereby capturing a premium for the milk they produce.

cows. A cow starts the season after giving birth to a calf and therefore begins producing milk. If the farmer keeps milking the cow on a regular basis and the cow remains in good health, the cow will continue to produce milk. However, due to low feed quality or low feed availability towards the end of the season, the energy spent producing milk may come at the expense of animal health (as measured by the Body Condition Score). Therefore, to reduce the energy expenditure of the cow so that more energy can be spent increasing fat reserves, the farmer may choose to *dry-off* the cow. This action causes the cow to cease milk production. After this point, the mammary glands stop secreting milk until another calf is born. Therefore, the farmer faces a trade-off between extending lactation in order to increase milk production, and halting lactation in order to improve animal condition for the next season. We refer to the length of time that the cow is milking as the *lactation length*. In New Zealand, the average lactation length is around 276 days [52].

A final complication is the mechanism by which farmers are paid for milk. The majority (84%) of dairy farmers in New Zealand supply milk to Fonterra, a large milk processing co-operative [195]. However, they are not paid for their milk on delivery. Instead, at the end of each season (one year), they are back-paid an *end-of-season* milk price for each kilogram of milk solids supplied during the preceding season.[5] This end-of-season milk price is calculated from a series of international auctions for dairy commodity products that occur during the year. Therefore, at the start of the season there is some uncertainty surrounding the end-of-season milk price since the auctions have not yet occurred. To help guide the farmers in their decision-making, Fonterra releases a sequence of *forecast* milk prices during the season. As the season progresses, this sequence of forecast milk prices trends toward the end-of-season milk price.

Therefore, during each season, the farmer faces a multistage stochastic optimisation problem that we term the *Pastoral Dairy Farmer Problem*.

The Pastoral Dairy Farmer Problem

maximise:	revenue from milk production less operating costs
by deciding:	the number of cows to farm
	the quantity of supplement to feed
	the quantity of grass to feed
	when to dry-off the herd
subject to:	obtaining a high Body Condition Score at the end of the season
	uncertainty in the rate of future grass growth
	uncertainty in the price of milk.

In this half of the thesis we investigate various aspects of the Pastoral Dairy Farmer Prob-

[5] In reality, this process is more complicated, since it involves part-payments during the season and a series of wash-up payments after the season has finished. However, the net effect of these payments is the same as we have described.

lem. Although we focus on the New Zealand dairy industry, "the reader, by means of a simple transformation of the situations, will be able to conceive many questions of similar nature."[6]

6.2 Literature review

In the previous section we described two important variables in the Pastoral Dairy Farmer Problem: the stocking rate (number of cows per hectare) and the supplementation intensity (quantity of feed imported to the farm).

New Zealand has a five-point scale for describing the supplementation intensity ranging from System One (no feed imported) to System Five (more than 25% of feed is imported) [48]. From the 2007/08 to the 2014/15 season, the percentage of System One and System Two farms decreased from 44% to 32%, signalling a shift in farming methodologies towards increased supplementation [45, 49]. An analysis by Dairy NZ [49] found that the operating profit per hectare increased as the proportion of feed imported increased. However, the same analysis found that System One and System Two farms had a lower debt-to-asset ratio than System Four and System Five farms. Therefore, low-intensity farmers may be better placed to cope with milk price volatility [190].

Farmers must also decide how many cows to farm. From the 2007/08 to the 2014/15 season, the average stocking rate in New Zealand increased from 2.79 cows/ha to 2.87 cows/ha [45, 49]. Moreover, in the 2014/15 season, System One and System Two farms had an average stocking rate of 2.5 cows/ha compared with 3 cows/ha for System Four and System Five farms [49]. Combined with the trend towards higher input farming systems (i.e. System Four and Five), there is evidence of a positive correlation between the supplementation intensity and the stocking rate on New Zealand dairy farms.

Considerable research has been conducted to understand the impact of the stocking rate and supplementation intensity on farm profitability, including controlled field trials [132–134], "what-if" case studies [59, 209], and the construction of mathematical models of single animals [15, 35, 88, 122] and whole farm systems [19, 63, 113, 123, 182]. Although the exact results of these studies vary depending upon environmental (e.g. rainfall, soil type), economic (e.g. milk price, supplement cost), and biological (e.g. breed of cows) factors, the literature can be summarised into a single trend: on average, high intensity farming systems (i.e. System Four or Five on the five-point scale) are the most profitable; however, low-input, low-intensive farming systems (i.e. System One or Two) are more profitable when the milk price is low.

Regarding the mathematical modelling approach, we direct readers to review papers by Bryant and Snow [36], Feola et al. [73], and Snow et al. [191]. Few of the models explored in these reviews incorporated optimisation. Authors that did incorporate optimisation usually implemented some form of an evolutionary search algorithm (e.g. [7, 90, 101, 143]).

[6] To borrow a phrase from Bellman and Dreyfus [24, p. 27].

One notable exception to this is the Integrated Dairy Enterprise Analysis (IDEA) [64], a nonlinear optimisation model of a pastoral dairy farm, which we consider to be the current state-of-the-art. IDEA divides the dairy farming season into fortnightly blocks and maximises operating profit. Decision variables in IDEA include a number of operational decisions (such as the type and quantity of supplement to buy and feed each fortnight), as well as some strategic decisions such as stocking rate. A detailed description of the model is given in Doole et al. [63]. IDEA has been used to investigate various aspects of New Zealand dairy farms, including supplementary feeding [59], reducing greenhouse gas emissions [1, 60], stocking rates [62, 180, 181], and general profitability [61]. Although IDEA is a useful tool to investigate locally optimal management strategies, due to the use of nonlinear optimisation it is difficult to claim that the solutions it produces are global optima.

IDEA shows how we can model the farming season as a deterministic sequential decision-making process. When uncertainty is incorporated, the problem becomes a multistage stochastic optimisation problem of the type we discussed in the first half of this thesis. In a recent review paper, Borodin et al. [32] lay out the state-of-the-art in handling uncertainty in agricultural supply chain management. They conducted a wide survey of papers, and a number of stochastic programming approaches were identified as being used in the literature. Most were two-stage models, and only a few were concerned with dairy [74, 92, 102, 174]. The paper of Guan and Philpott [92] is a notable exception in that it formulates, and solves, a large multistage stochastic optimisation problem. However, it concerns the operations of a large dairy processing company, rather than an individual farmer.

In their review, Borodin et al. [32] note that "multistage stochastic programs, in general, are intractable," and that "even if the use of [multistage stochastic programming] is showing increasing promise, there are still very few real-world implementations ... in the agricultural area." However, multistage stochastic programming has been widely applied to real-world problems in the electricity sector with considerable success (for example, in Brazil [135]). A key contributor to this has been the development [153, 161] and improvement [55, 160] of the *stochastic dual dynamic programming* (SDDP) algorithm, which we described in the first half of this thesis.

For readers who have not read the first half of this thesis, SDDP is a dynamic programming-inspired algorithm for solving multistage stochastic optimisation problems. It decomposes the multistage stochastic optimisation problem in time into a series of sequential subproblems. Each subproblem (typically associated with a discrete interval in time) is an optimisation problem that chooses an action in order to minimise the cost associated with the current decision, plus the *cost-to-go* of the remaining stages given the action is taken. Traditional dynamic programming [23] estimates the cost-to-go function (also called the Bellman function) at a set of discretised points. However, because of this discretisation, the method is limited to low-dimensional problems (dynamic programming's "curse of dimensionality"). Instead of evaluating the function at a set of discretised points, SDDP approximates the Bellman function with a set of piecewise linear functions called *cuts*. When the problem instance has a specific form (stagewise

independence of the random variable, convexity of the Bellman function, and continuous state variables), the SDDP algorithm can efficiently find an optimal policy (to within some bound).

6.3 Objective and outline

There are two main objectives for this half of the thesis: (1) to investigate the trade-off between stocking rate and supplementation intensity; and (2) to investigate the impact of weather and price uncertainty on the management actions of a dairy farmer.

This half of the thesis is structured into three chapters, each of which analyses a different component of pastoral dairy farming through the lens of multistage optimisation. The first, Chapter 7, stands apart from Chapters 8 and 9 in that it uses dynamic programming as a solution method instead of stochastic dual dynamic programming and does not include weather uncertainty. The remaining two chapters are related and should be read in sequence.

The first of the three chapters, Chapter 7, develops MOO – the Milk Output Optimiser. MOO is a dynamic program based on *E-Cow*, a nonlinear model of a single dairy cow [18]. We present a case study in which we use MOO to analyse the trade-off that a selected farmer in the Bay of Plenty region of New Zealand makes between their stocking rate and supplementation intensity. We find that the current farm set-up is sub-optimal; operating profit could be increased by decreasing the stocking rate, or by increasing the supplementation intensity. However, since MOO is a dynamic program, we run into the "curse of dimensionality." This prevents further extensions to MOO and limits our ability to add uncertainty.

The second of the three chapters, Chapter 8, develops POWDER – the milk Production Optimiser incorporating Weather Dynamics and Economic Risk. Compared to MOO, POWDER makes a number of simplifications in order to escape the curse of dimensionality. However, it includes weather uncertainty and a simple stochastic model for the price of milk. We apply POWDER to the same farm discussed in Chapter 7 and show that the farmer's actions depend on the price of milk. If the milk price is high, the farmer purchases more supplement and delays the decision to dry-off their cows. If the milk price is low, the opposite is true. We also confirm the finding from Chapter 7 that the farm is over-stocked, and that operating profit could be increased by decreasing the stocking rate.

Finally, Chapter 9 extends the POWDER model developed in Chapter 8 to accommodate two alternative auto-regressive models for the end-of-season milk price and the ability of the farmer to forward contract their milk. Again, we apply POWDER to the same farm discussed in the previous two chapters. We find that without contracting, the farmer has little recourse to manage their downside risk since most of their risk is due to the price of milk, rather than environmental risks such as drought. In contrast, forward contracts for milk allow the farmer to mitigate most of their downside risk in exchange for a small decrease in their expected operating profit. We also investigate a number of other hypothetical risk management tools such as weather derivatives, but find that these would offer limited benefit to our farmer.

6.4 Units

Before we begin, we advise the reader of two important points regarding the units that we use in this half of the thesis.

First, when we discuss the milk, we do so in terms of kilograms of milk solids. Milk solids are the fat and protein components of liquid milk (i.e. milk, excluding water and lactose). Typically, kilograms of milk solids are denoted kgMS. However, to avoid confusing readers unfamiliar with the dairy industry, we shall use units of kg. At no point will we refer to liquid milk in terms of litres.

Second, when we discuss feed (e.g. palm kernel, grass, etc.), we refer to it on the basis of kilograms of dry matter. Similar to milk solids, dry matter refers to the mass of a feed after the water has been removed. Typically, kilograms of dry matter are denoted kgDM. However, following a similar argument to the one we used for milk solids, we shall use kg. In addition to this, when discussing supplement (e.g. palm kernel), we refer to the quantity of dry matter consumed by the cow. At no point shall we refer to supplement via its weight as fed by the farmer. In addition, we will not explicitly account for losses like wastage and spoilage; these are accounted for in the cost per kilogram of dry matter consumed.

Chapter 7

Optimising stocking rates and supplementary feeding

In this chapter we embed the *E-Cow* [18] model of a dairy cow into a dynamic program. We call this dynamic program MOO – the Milk Output Optimiser. MOO breaks the dairy farming season into weekly intervals and calculates the optimal actions the farmer should make each week. We use MOO to investigate the trade-off a selected farmer faces between their stocking rate and supplementation intensity. Our findings reinforce empirical evidence that high-input, high-stocking rate farms are the most profitable farms in expectation, but, if the farmer is risk-averse, they should reduce their intensification level by purchasing less supplement and by lowering their stocking rate.

For brevity, we do not provide a complete description of the MOO model or a list of all the parameter values used. Instead, we provide the Julia code to implement MOO, as well as the data needed to run the case study and replicate the results in this chapter, at `https://github.com/odow/MOO`.

The chapter is laid out as follows. First, in Section 7.1 we analyse a simple example in order to gain an intuitive understanding of the problem. Then, in Section 7.2, we formulate MOO as a multistage deterministic optimisation problem. In Section 7.3, we apply MOO in a case study to a farm in the Bay of Plenty region of New Zealand. In Section 7.4 we extend MOO to model the trade-off between stocking rate and supplementation intensity as a two-stage stochastic program. We finish the chapter with a discussion in Section 7.5 and some summarising remarks in Section 7.6.

7.1 A simple model

Before we introduce MOO, it is useful to analyse a simple example in order to gain some intuition behind the problems faced by dairy farmers and the kind of results we can expect. In particular, we analyse the trade-off between the stocking rate (the number of cows per hectare)

and the total quantity of feed available.

Consider a farm that has a stocking rate of ρ cows/ha. Each day, the farmer feeds each cow q kg of feed. The cow consumes $\bar{q}$ kg/day of the feed for maintenance (i.e. the quantity needed each day to stay alive) and apportions the rest ($q - \bar{q}$ kg/day) to milk production. The maintenance is a minimum requirement, so $q \geq \bar{q}$. In response to consuming $q - \bar{q}$ kg/day of feed for milk production, the cow produces $f(q - \bar{q})$ kg/day of milk. Since there is some biological maximum quantity of milk that can be produced by the cow, the law of diminishing returns applies. Thus, we assume f is a concave, non-decreasing function of the quantity of feed allocated to milk production. Finally, we assume that the farmer has a total quantity Q of feed available each day (kg/ha/day). (We normalise this value to a per hectare value, and we do not distinguish between grass and supplementary feed such as palm kernel.)

Therefore, given a total quantity Q of feed available (kg/ha/day), the farmer's job is to choose a stocking rate ρ (cows/ha) and a per cow quantity of feed q (kg/cow/day) in order to maximise total milk production:

$$\begin{aligned} \text{Total Milk Production}(Q) = \max_{\rho,q} \quad & \rho \times f(q - \bar{q}) \\ \text{s.t.} \quad & \rho \times q \leq Q \\ & q \geq \bar{q}. \end{aligned}$$

We now consider a simplistic animal model in which the cow requires $\bar{q}$ =2.75 kg/day of feed for maintenance. For every extra 7.5 kilograms of feed the cow consumes, they produce 1 kg of milk, up to a limit of 2 kg of milk per day (i.e. a maximum intake of extra feed for milk production of 15 kg/day). These values, and the assumption that milk production increases linearly with feed intake, were based on rough averages calculated by Dairy NZ [47].

Therefore, the optimisation problem faced by the farmer is:

$$\begin{aligned} \text{Total Milk Production}(Q) = \max_{q,\rho} \quad & \rho \times (q - 2.75)/7.5 \\ \text{s.t.} \quad & \rho \times q \leq Q \\ & 2.75 \leq q \leq 17.75, \end{aligned}$$

which has the optimal solution of $q = 17.75$ and $\rho = \frac{Q}{17.75}$ for any given Q. In Figure 7.1 we illustrate the trade-offs faced by a farmer.

If we choose a fixed quantity of feed Q and then increase the stocking rate ρ from zero, there is sufficient feed available for each additional cow to produce the maximum quantity of milk of 2 kg/cow/day. Therefore, milk production increases linearly as the stocking rate ρ increases. When the stocking rate reaches $\rho = Q/17.75$ cows/ha, each cow is producing 2 kg/cow/day of milk, and the total quantity of feed available is consumed. As we increase the stocking rate further, the total available feed Q is distributed among more cows. Therefore, the quantity q that each cow receives decreases, and the corresponding decrease in milk production per cow is more than the increase in milk production from the additional cow. Therefore, total milk

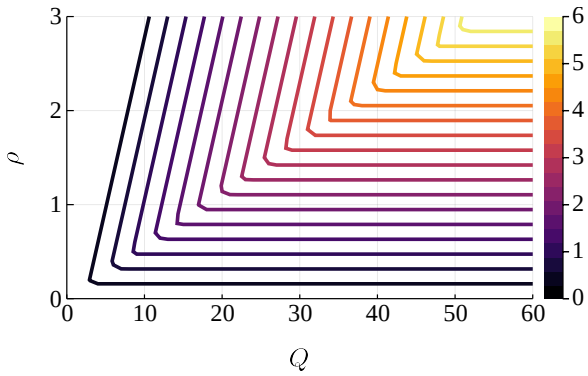


Figure 7.1: Contour plot showing the total quantity of milk produced (kg/ha/day) as the total quantity of feed available Q and the stocking rate ρ are varied.

production decreases. Once the stocking rate ρ exceeds $Q/2.75$ cows/ha, the feed required for maintenance, $\rho\bar{q}$, exceeds the total available quantity, and the solution is infeasible.

Farmers make this trade-off by choosing a stocking rate ρ at the start of the season when the total quantity of available feed Q is unknown (due to uncertain grass growth, or uncertain price of supplement). If they choose a stocking rate that is too low, they may under-utilise their pasture and forego revenue. If they choose a stocking rate that is too high, they may under-utilise their cows by spending too much feed on maintenance.

In the remainder of this chapter, we investigate this trade-off using *E-Cow*, a detailed animal model of a single cow [18]. In particular, the maintenance requirement $\bar{q}$ and the shape of the milk production function $f(q - \bar{q})$ are not fixed in *E-Cow*; instead, they vary depending on the time of year, health of the cow, and many other factors. As we shall see, the results we obtain using *E-Cow* (e.g. Figure 7.6) are different from those above, but they obey similar principles.

7.2 Formulation

In this section we formulate MOO as a multistage deterministic optimisation problem that models, using *E-Cow* [18], a cow over a single season (i.e. one year). The objective of the farmer is to maximise revenue (from selling milk), while meeting various animal welfare constraints. However, before we detail the specifics of the multistage deterministic optimisation model, we begin with a brief description of *E-Cow*, the animal model that forms the basis for MOO.

7.2.1 The *E-Cow* model

E-Cow [18] is a nonlinear system of equations that approximates the biological workings of a dairy cow. The model first predicts the energy intake of a cow given factors such as feed availability, feed quality, and the number of days since the cow's last calving. Then, energy is set aside for the maximum rate of milk production, genetically driven fat deposition, pregnancy, and maintenance (the energy required to maintain basic life functions such as basal metabolism,

digestion, and exercise). Any negative energy imbalance is resolved by reducing the quantity of energy apportioned to milk production and fat deposition (i.e. by reducing the rate of accumulation, or increasing the rate of burning, of fat reserves). Since we assume that the cow is producing the maximum rate of milk production, any positive energy balance is resolved by allocating all of the excess energy to building fat reserves.

E-Cow can predict a range of factors including grass intake, milk production, and Body Condition Score (BCS). The BCS is a subjective assessment of the proportion of body fat in a cow and is used by farmers as an important indicator of animal health [177]. *E-Cow* uses the United States system of a five-point scale, where 1 represents an emaciated cow, and 5 represents an obese cow. The *E-Cow* model has been previously validated against experimental data for New Zealand Holstein-Friesian cows that were grazed on predominantly pasture diets (i.e. less than 50% supplementation). *E-Cow* integrates three existing published models: grass intake [17], milk production [201], and body condition change [78]. It is a deterministic simulation with a daily time-step and has been implemented in a web interface [16]. For brevity we have chosen to omit the full formulation here; however, readers are directed to Baudracco [15] for the original formulation or to the GitHub repository[1] for our implementation in Julia [28].

There are three main decisions that the farmer must make in each time step: the quantity of grass to offer, the quantity of supplement to offer, and whether to dry-off the cow. We now describe each of these three decisions.

First, when 'feeding' the *E-Cow*, the farmer cannot force the cow to consume grass; instead, the farmer *offers* a quantity of grass by placing the cow (or the herd) in a paddock with a given pasture cover (the quantity of grass per hectare). The farmer can control the quantity of grass to offer by paddock selection (for size and pasture cover) or by subdividing a paddock using temporary fencing. In this thesis, we ignore these complications and assume that the farmer can offer any quantity of grass. In response to being offered some grass, the cow will consume some proportion of the offered quantity based on the time of year, net energy deficit, quality of grass, and other factors.

Second, the farmer chooses a quantity of supplement to feed. If the farmer feeds supplement, some *substitution* will occur as the cow reduces its grass intake to compensate for the supplement intake. *E-Cow* assumes that the cow preferentially consumes supplement over grass from pasture.

Finally, one of the most important decisions a farmer must make is when to *dry-off* their cows. This action causes the cow's mammary glands to stop secreting milk until another calf is born. However, this reduces the energy requirements of the cow, allowing it to increase its fat reserves. Therefore, the farmer faces a trade-off between extending lactation in order to increase milk production and halting lactation in order to improve animal condition for the next season.

We chose *E-Cow* as an animal model for two reasons. First, the model is simpler than other

[1] `https://github.com/odow/MOO`

animal models such as MINDY [88], IDEA [63], and Dairy-Mod [113]. This is important as we shall see that despite coding the model in the fast, numerical computing language Julia [28] and exploiting parallelism, solve times are on the order of hours. Moreover, our solution method (*backward recursion* – see Appendix A) scales poorly. Therefore, increasing the complexity of the model could create solve times on the order of days. Second, the *E-Cow* model has been validated against experimental data for New Zealand Holstein-Friesian cows in New Zealand conditions [18].

Example simulation

To give the reader a better understanding of the inputs and outputs that are modelled in *E-Cow*, we coded a simulation model of *E-Cow* in the Julia programming language [28].

Four inputs are required by the *E-Cow* simulation: a calibrated set of parameters that define the cow; the quantity of grass to offer each day; the quantity of supplement to feed each day; and the number of weeks to milk the cow.

The *E-Cow* model can be calibrated to different animal breeds. In this example, and the rest of this chapter, we use the constants given in Baudracco et al. [16] for a New Zealand Holstein-Friesian. For simplicity in this example, the farmer offers 35 kg/day of grass (containing 10.3 MJ/kg)[2], the cow consumes 2 kg/day of supplement (containing 11.5 MJ/kg), and the cow is milked for 40 weeks.

Using these four inputs, our implementation simulates the evolution of five key output variables each day over one year (365 days). The five output variables are the Body Condition Score, the energy required for maintenance, the quantity of milk solids produced, the quantity of grass consumed, and the quantity of supplement consumed. In Figure 7.2, we plot these five output variables against the day of the season.

The cow begins the season with a Body Condition Score of 3.2 (Figure 7.2a). During the first 100 days, the cow loses body condition. The reason for this is that the cow produces milk at the expense of its fat reserves in order to provide for the calf it has just given birth to. At day 112, the cow conceives a new calf. After this point, the cow stops losing body condition and focuses on regaining its fat reserves. However, the rate at which it can gain fat is limited by the quantity of food available. At day 280 (i.e. 40 weeks), the farmer stops milking the cow, allowing the animal to focus its energy on regaining body condition in preparation for the next season. This explains the rapid accumulation of body condition in the last part of the season.

E-Cow defines the concept of energy required for maintenance as a delayed response to actions taken in the previous time step. This depends on factors such as the quantity of grass grazed and the quantity of milk produced. All else being equal, a cow with a greater Body Condition Score, or that consumes more food, or that produces more milk, will require a greater amount of energy the following day for maintenance than an otherwise identical cow. In Figure

[2]We really mean MJ ME/kgDM – the megajoules of metabolisable energy per kilogram of dry-matter.

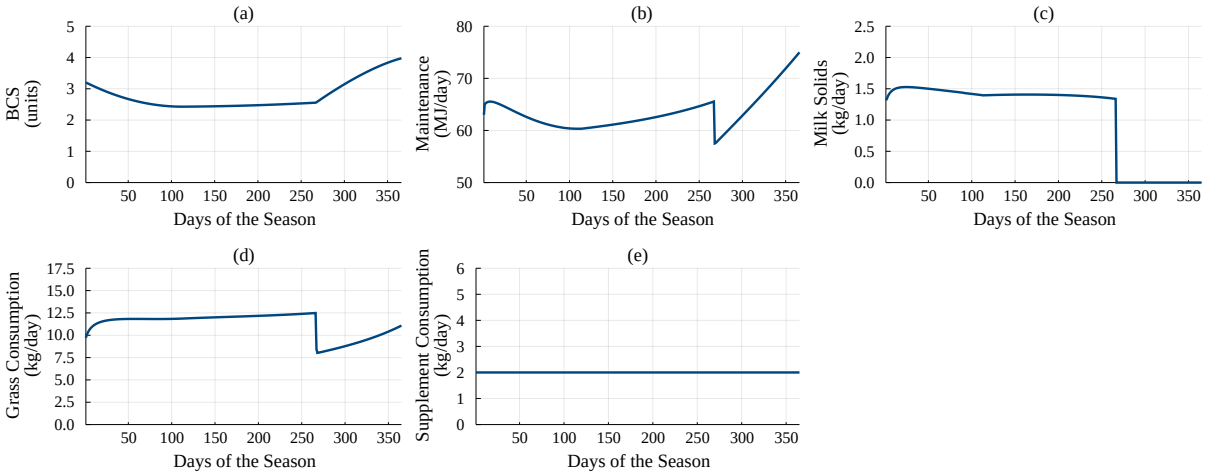


Figure 7.2: Output variables from a simulation of a single cow using the *E-Cow* model: (a) the Body Condition Score; (b) the energy required for maintenance (MJ/day); (c) the quantity of milk solids produced (kg/day); (d) the quantity of grass consumed (kg/day); and (e) the quantity of supplement consumed (kg/day).

7.2b, we see that the energy required for maintenance rises during the first 10 days, corresponding to the rise in milk solids production. However, after this point, the energy required decreases as the body condition and milk solids production decrease. After conception on day 112, the energy required for maintenance increases due to the increase in body condition. After day 280, the cow is dried-off, causing a sudden drop in the energy required for maintenance. However, after this point, the energy required for maintenance increases again due to the increase in body condition.

When consuming grass, the *E-Cow* model considers factors such as grass quality, grass availability, and the energy requirements of the cow. In this example, we offered the cow 35 kg/day of grass. The *E-Cow* model predicts (see Figure 7.2d) that, in addition to the 2 kg/day of supplement (Figure 7.2e), the cow will consume only 12.5 kg/day of the offered quantity of grass. After day 280, the energy requirement of the cow decreases due to the suspension of milk production. Therefore, the cow is less driven to consume grass, explaining the sudden drop in feed intake (Figure 7.2d).

Figure 7.2c plots the quantity of milk solids produced by the cow per day across the season. During the first 20 days of the season, there is a rise in milk solids production due to the biological workings of the cow. After this early peak, and until conception (at day 112), the cow sustains a higher rate of milk production (compared with days 112-280) due to the conversion of body condition (i.e. fat) into energy for milk production. After day 280, the cow is dried-off and milk production ceases.

7.2.2 The MOO model

MOO is a multistage deterministic optimisation problem that we solve using *dynamic programming* (see Appendix A). In this section, we describe MOO using the notation outlined in Chapter 1. However, note that in this model there is no uncertainty. Therefore, there are no noise terms.

To aid the reader's understanding, we provide a diagram of the model in Figure 7.3 using the diagramming conventions of System Dynamics [192]. States are in square boxes. Controls that can be chosen by the farmer are named in bold underlined text.

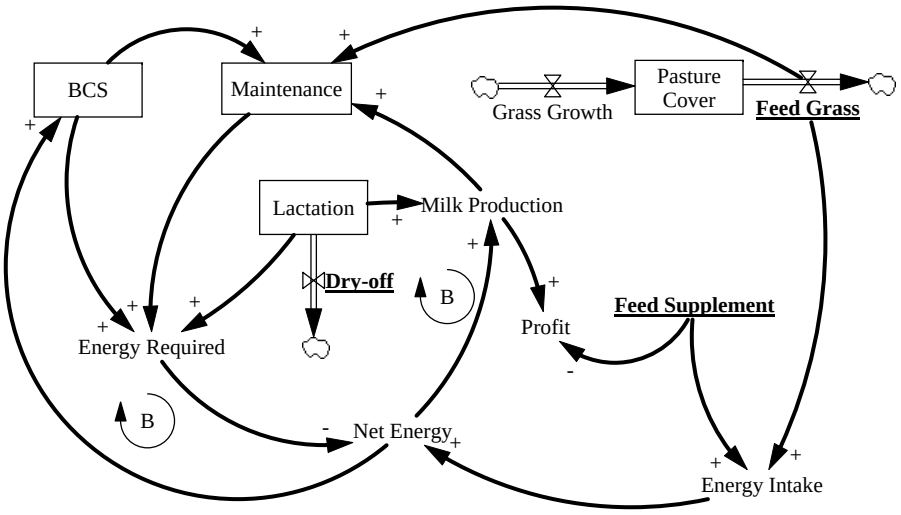


Figure 7.3: System Dynamics diagram of the MOO model.

We now describe the stages, states, controls, transition function, constraints, and stage-objective that comprise our multistage deterministic optimisation problem.

Stages MOO divides the dairy farming season (i.e. one year) into 52 weekly stages $t = 1, 2, \ldots, 52$. In each week, we choose a control (e.g. the quantity of grass to offer the cow each day), but we simulate the control through *E-Cow* on a daily basis. This reduces the computational effort compared with solving an optimisation problem with 365 stages (one for each day).

State variables There are four state variables in the model. Each variable is indexed by week t and corresponds to the value at the beginning of week t. The state variables are:

(1) the Body Condition Score x_t^b;
(2) the energy required for maintenance x_t^e (MJ/day);
(3) the lactation status x_t^l; and
(4) the average pasture cover x_t^g (kg/ha).

We use the value $x_t^l = 1$ to code when the cow is lactating at time t, and the value $x_t^l = 0$ to code when the cow is dried-off and thus not lactating. In addition, we use $x_t = (x_t^b, x_t^e, x_t^l, x_t^g)$ to denote the state of the system at the start of week t.

Control variables The farmer has three different actions they can use to change the state of the system during week t. They are:

(1) the quantity u_t^s of supplement to feed each cow, each day, during week t (kg/cow/day);
(2) the quantity u_t^g of grass to offer each cow, each day, during week t (kg/cow/day); and
(3) a dry-off decision u_t^l made at the end of week t.

We use the value $u_t^l = 1$ to denote the decision to stop milking the cow at the end of week t and the value $u_t^l = 0$ to denote the decision to maintain the current state. In addition, we use $u_t = (u_t^s, u_t^g, u_t^l)$ to represent the control chosen by the farmer in week t. Finally, note that u_t^g is not the quantity *consumed* by the cow. (As we explained earlier, the quantity of grass consumed by the cow is determined by the *E-Cow* system of equations.)

Transition function We define the dynamics of the system as a nonlinear function that maps the current state x_t to a new state x_{t+1}, given the control u_t such that $x_{t+1} = T_t(x_t, u_t)$. For brevity we have chosen to omit the full formulation of $T_t(\cdot)$. Instead, readers are directed to Baudracco [15] for a more detailed explanation or to the GitHub repository[3] for our implementation in Julia [28].

It is important to reiterate that although the transition function acts on a weekly time step, we have not aggregated the *E-Cow* model into a weekly time step. Instead, the transition function acts over a weekly time step by simulating *E-Cow* on a daily basis, holding the control u_t constant for each day in week t. This results in some loss of solution quality (since the optimal decision may be to dry-off halfway through a week) but reduces the computational complexity of the optimisation model.

Constraints When choosing the control u_t we must respect a set of constraints. For example, once the cow is dried-off we cannot resume milking it. Therefore, we have the constraint that:

$$x_{t+1}^l \leq x_t^l.$$

In addition, even though that dairy farming is a multi-year enterprise, the natural structure of the dairy farming season allows us to model the farm system as a finite horizon problem. However, in order to ensure that the farm is in an adequate state for the following season, we force the Body Condition Score at the start of the 53^{rd} week x_{53}^b and pasture cover x_{53}^g to be greater than, or equal to, their initial values ($x_1^b = 3.2$ and $x_1^g = 2500$). This creates a steady state solution but ignores the fact that cows age over time and have a finite lifespan.

We denote the set of constraints by $u_t \in U_t(x_t)$.

[3] `https://github.com/odow/MOO`

Stage-objective The objective of the farmer is to minimise cost. This cost comes from purchasing supplement, less the profit from selling milk. The quantity of milk solids produced by the cow is a nonlinear function $M_t(x_t, u_t)$ of the state of the cow x_t and feeding actions u_t taken by the farmer. (Again, this function is defined by the *E-Cow* model, so for brevity we do not provide the full formulation.) Therefore, the stage-objective for stage t is:

$$C_t(x_t, u_t) = \rho \times \left(c^s \times u_t^s - p^m \times M_t(x_t, u_t)\right), \quad t = 1, 2, \ldots, 52, \tag{7.1}$$

where ρ is the stocking rate (cows/ha), p^m is the price of milk solids (\$/kg), and c^s is the cost of supplement (\$/kg). These values are set as constants in the model.

Policy graph We model this problem using a linear policy graph with 52 stages, one for each week $t = 1, 2, \ldots 52$. Since there is no uncertainty, each subproblem is deterministic. The formulation of the subproblem in week t is:

$$\begin{aligned} \mathbf{D}_t(x_t): \quad V_t(x_t) = \min_{u_t} \quad & C_t(x_t, u_t) + V_{t+1}(x_{t+1}) \\ \text{s.t.} \quad & x_{t+1} = T_t(x_t, u_t) \\ & u_t \in U_t(x_t), \end{aligned} \tag{7.2}$$

where $V_{53}(\cdot) = 0$. The objective of the farmer is to minimise $V_1(x_1)$.

7.3 Case study

In this section, we investigate the application of MOO to a working dairy farm in the Bay of Plenty region of New Zealand. For simplicity, we shall refer to it as *the farm*. The results we obtain will be specific to our chosen farm; however, the study can easily be repeated for different farms.

Supplementation Until this point, we have discussed "supplement" in generic terms. This is because the *E-Cow* model can be parameterised to model different types of supplement. In the remainder of this chapter we assume that the farmer chooses *palm kernel* as their supplement. Palm kernel is a by-product of the production of palm oil. It is commonly used as a supplementary feed on New Zealand farms as it has a high energy content, is easily obtained (typically arriving on-farm within a day or two of ordering), and is relatively inexpensive [44]. We ignore other supplementary feeds such as maizebut note that they can easily be added into our model.

7.3.1 Calibration

The majority of the parameters in MOO are part of the *E-Cow* model. For these parameters, we used the values given in Baudracco et al. [18] for a New Zealand Holstein-Friesian. In

addition to the *E-Cow* parameters, MOO contains parameters for the price of milk p^m, the cost of supplement (i.e. palm kernel) c^s, the stocking rate ρ, and the grass growth rate g_t (in kg/ha/week for each week $t = 1, 2, \ldots, 52$). We now detail how we calibrated these parameters for the case study.

DairyBase is a voluntary database of financial and physical key performance indicators for New Zealand dairy farmers [50]. It provides a standardised reporting mechanism that can be used to benchmark a farm's performance. The owner of the farm investigated in this case study provided us with access to their DairyBase data for the 2013/14 and 2014/15 production seasons. The data included information such as the total milk solids production per hectare, the quantity of pasture and palm kernel eaten, and the lactation length. It also included a detailed break-down of operating expenditure. We provide a summary of the data in Table 7.1. We draw the reader's attention to two items. First, the palm kernel cost is not the price per tonne from the supplier (typically \$275/t-\$350/t for palm kernel [49]) but includes the additional costs of storage, as well as wastage and spoilage. Second, the line item *Fixed Expenses* accounts for all costs excluding the cost of palm kernel. This allows us to standardise the operating profit per hectare between seasons to account for changes in milk price and palm kernel costs.

	2013/14	2014/15	Average
Stocking Rate (cows/ha)	3	3	3
Milk Price (\$/kg)	8.40	4.40	6.40
Milk Production (kg/ha)	1,240	1,146	1193
Lactation Length (days)	275	256	266
Pasture Consumption (t/ha)	12.6	11.7	12.15
Palm Kernel Consumption (t/ha)	2.8	2.9	2.85
Palm Kernel Cost (\$/t)	473	419	446
Fixed Expenses (\$/ha)	3512	3560	3536

Table 7.1: Summary data for the farm over two seasons, and the average of these.

Based on the DairyBase data, we set the stocking rate ρ to 3 cows/ha, the price of milk p^m to \$6/kg, and the cost of palm kernel c^s to \$0.5/kg. However, the farmer has no long-term records of grass growth for their farm. Therefore, we take the monthly average grass growth of a nearby farm (chosen by the farmer from Dairy NZ [47]) and interpolate these values to obtain a weekly estimate for the average grass growth rate. Then, the average values are scaled so that the cumulative total matches the average pasture consumption in Table 7.1. This results in a deterministic sequence of grass growth rates $g_1, g_2, \ldots, g_{52}$.

7.3.2 Solution method

MOO was solved using the `DynamicProgramming.jl` package (see Appendix A) in the Julia language [28]. As part of the solution process, it is necessary to discretise the state and control variables. A coarse discretisation reduces the computational effort required to solve the

problem, but it introduces interpolation error and so may result in a poorer solution. We chose the discretisation listed in Table 7.2 through experimentation as a trade-off between speed and accuracy. In each stage, there are 55,022 discrete points in the state-space and 286 possible controls. Since we need to evaluate the cost of each control at every discrete point in the state-space for each stage $t = 1, 2, \ldots, 51$, this results in $\approx 8 \times 10^8$ evaluations of the transition and cost functions. (Note that we do not need to evaluate the cost-to-go of the 52nd stage since $V_{53}(\cdot) = 0$.) The discretised model was solved using eight cores of an Intel i7-4770 CPU with 16GB of memory. The total solution time is approximately 25 minutes.

Variable	Discretisation
x^b	2.500, 2.525, 2.550, ..., 4.000
x^e	40, 45, 50, ..., 90
x^l	0, 1
x^g	1500, 1550, 1600, ..., 3500
u^s	0, 0.5, 1, ..., 6
u^g	0, 5, 10, ..., 50
u^l	0, 1

Table 7.2: Discretisation of the state and control variables for the case study.

7.3.3 Results

The solution of the model is a policy based on an estimate for the cost-to-go function $V_t(x_t)$ at each of the discretised points in the state-space. We can interpolate between the discretised points to create a four-dimensional surface. The dimensionality makes the cost-to-go surface difficult to visualise, but one can fix the values of some state dimensions to make this easier. In Figure 7.4, we set the energy required for maintenance $x^e = 70$ and the lactation status $x^l = 1$, and then plot the cost-to-go surface $V_t(x_t)$ with respect to the Body Condition Score x_t^b and the pasture cover x_t^g for weeks $t = 1$ (Figure 7.4a), $t = 25$ (Figure 7.4b), and $t = 50$ (Figure 7.4c). In all weeks, the cost-to-go decreases as the Body Condition Score x_t^b increases. However, for a fixed pasture cover x_t^g, the marginal value of each extra unit of body condition decreases. In addition, in all weeks the cost-to-go decreases as the pasture cover increases. In Figure 7.4c, the cost-to-go reflects the two end-of-horizon penalties in the model incurred when the pasture cover x_{53}^p is below 2500 kg/ha and when the Body Condition Score x_{53}^b is less than 3.2.

Given the initial state $x_1 = (3.2, 70.0, 1, 2500)$, we use the cost-to-go function to simulate the policy. The result of the simulation is a sequence of optimal controls $u_1, u_2, \ldots, u_{52}$ and a corresponding sequence of optimal states $x_1, x_2, \ldots, x_{53}$. These are plotted in Figure 7.5. (We have not plotted the lactation state x_t^l or the dry-off decision u_t^l since they can be inferred from the other outputs.)

The results are broadly similar to those observed in Figure 7.2, although now the policy is optimised. During the first 100 days, the cow loses body condition (Figure 7.5a). Then,

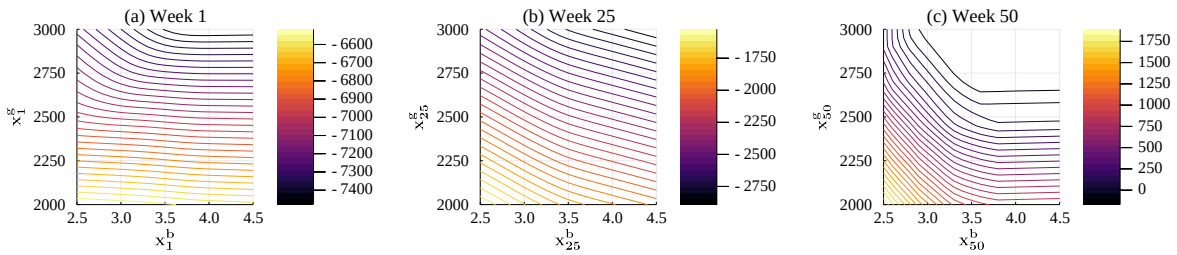


Figure 7.4: Contour plots showing the cost-to-go surface $V_t(x_t)$ ($/ha) for three weeks: (a) $t = 1$, (b) $t = 25$, and (c) $t = 50$. We vary the Body Condition Score x_t^b and pasture cover x_t^g, assuming the energy required for maintenance is fixed at $x_t^e = 70$ MJ/cow/day and the cow is current lactating (i.e. $x^l = 1$). Numbers that are more negative are better.

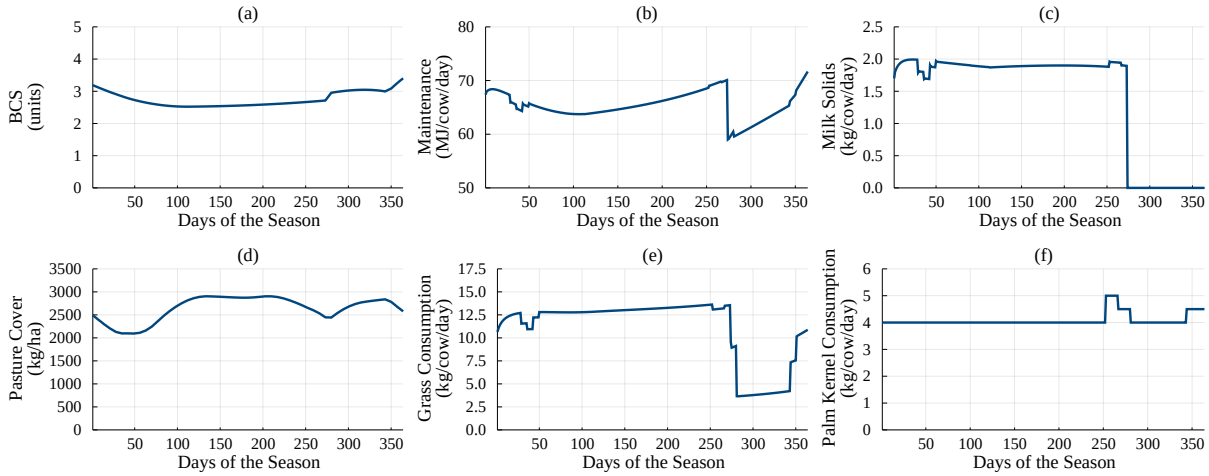


Figure 7.5: Simulation of MOO using the optimal policy: (a) the Body Condition Score x_t^b; (b) the energy required for maintenance x_t^e; (c) the quantity of milk solids produced; (d) the pasture cover x_t^g; (e) the quantity of grass consumed; and (f) the quantity of palm kernel consumed.

between days 100 and 280, the body condition increases. At day 280 the cow is dried-off (Figure 7.5c). This results in a reduction in the energy required for maintenance (Figure 7.5b), as well as the cessation of milk production (Figure 7.5c). As a consequence, feed intake drops (Figure 7.5e). The pasture cover (Figure 7.5d) follows a clear seasonal trend. During the first 50 days, the grass grows at a slower rate than it is eaten (Figure 7.5e). As a result, the pasture cover declines. Then, as the grass growth rate increases in spring, the pasture cover increases. During the late summer and early autumn (days 225–275), the pasture cover begins to decrease. Once the cows are dried-off at day 280, the reduction in feed intake allows the pasture cover to increase. However, in the last 20 days of the season the extra energy requirements caused by pregnancy, and the need to regain body condition in order to meet the end-of-season target (i.e. $x_{53}^b \geq 3.2$), lead to an increase in grass intake and the corresponding reduction in pasture cover. Over the season palm kernel is fed at approximately 4 kg/cow/day (Figure 7.5f). In addition, milk production, while milking, remains essentially constant at 1.8 kg/cow/day (Figure 7.5c).

In Table 7.3, we present a summary of the simulation and compare it with the DairyBase data from the farm (Table 7.1). Compared with the DairyBase data from the farm, MOO consumes

less grass (12.07 t/ha compared with 12.15 t/ha) but feeds more palm kernel (4.46 t/ha compared with 2.85 t/ha). This increases the quantity of milk produced from 398 kg/cow on the farm to 518 kg/cow in MOO. Therefore, the operating profit increases from $2197/ha to $3560/ha. This suggests that the farm can increase its profitability by feeding more palm kernel.

	MOO	Historical
Lactation Length (weeks)	39	38
Milk Production (kg)		
per Hectare	1555	1193
per Cow	518	398
Milk Revenue ($/ha)	9328	7158
Feed Consumed (t/ha)		
Pasture	12.07	12.15
Palm Kernel	4.46	2.85
% Feed Imported	27.0	19.0
Palm Kernel Expense ($/ha)	2231	1425
Fixed Costs ($/ha)	3536	3536
Operating Profit ($/ha)	3560	2197

Table 7.3: Summary data of the simulation using the optimal policy.

7.4 Two-stage model

In the previous section we solved an instance of MOO given a fixed stocking rate and the ability to purchase unlimited palm kernel. However, instead of using the spot market, some farmers purchase a fixed quantity of palm kernel on contract at the start of the season. Moreover, recall from our intuitive exploration of the problem (Section 7.1) that the two key strategic level decision variables are the stocking rate and total quantity of feed. In effect, farmers face a two-stage stochastic optimisation problem. In the first stage, farmers choose a stocking rate ρ and a quantity of palm kernel to purchase Q. Then, in the second stage, they operate according to the optimal policy like the one described in the previous section, but with a cap on the total quantity of palm kernel that they can feed over the season. In addition, the milk price p^m in the second stage is uncertain. In this section, we form a distribution for the milk price p^m from the last eight Fonterra end-of-season milk prices [76], which we list in Table 7.4. Each of the historical milk prices is assumed to occur with equal probability (i.e. 1/8).

	2009/10	2010/11	2011/12	2012/13	2013/14	2014/15	2015/16	2016/17
p^m	6.10	7.60	6.08	5.84	8.40	4.40	3.90	6.12

Table 7.4: Historical Fonterra end-of-season milk prices ($/kg).

As a result of choosing the stocking rate ρ and the quantity of palm kernel to purchase Q, and then operating according to an optimal policy, the farmer earns an operating profit ($/ha),

which we denote $\Pi(p^m, \rho, Q)$. As we discussed in Chapter 1, there are two types of stochastic optimisation subproblems that we could use to model this problem: Hazard-Decision and Decision-Hazard.

In a Hazard-Decision model, the farmer first observes the price (i.e. the hazard), and then chooses the stocking rate and quantity of palm kernel to purchase (i.e. the decision) that maximises total profit. This is also called a "Wait-and-See" model [30]. The Hazard-Decision objective is the expectation of taking the best action in each outcome of price:

$$\text{Hazard-Decision} = \mathbb{E}_{p^m}\left[\max_{\rho,Q}\left\{\Pi(p^m, \rho, Q)\right\}\right]. \tag{7.3}$$

In a Decision-Hazard model, the farmer is forced to choose a single stocking rate and quantity of palm kernel to purchase *before* they observe the price. This is also called a "Here-and-Now" model [30]. The Decision-Hazard objective is:

$$\text{Decision-Hazard} = \max_{\rho,Q}\left\{\mathbb{E}_{p^m}\left[\Pi(p^m, \rho, Q)\right]\right\}. \tag{7.4}$$

In a Decision-Hazard model, the farmer faces the *risk* of a poor outcome as a result of their decision. The optimal solution maximising expected profit may be to purchase a large quantity of palm kernel to support a high stocking rate. However, in years with a low milk price, the farmer may lose money as the cost of the palm kernel outweighs the value of the additional milk produced.

In the next section, we discuss how we solve these two-stage stochastic optimisation problems.

7.4.1 Solution method

In order to solve these two-stage stochastic programming problems (Eq. 7.3 and Eq. 7.4), we follow a four-step process. First, we extend and then solve a formulation of MOO with two new state variables: one for the quantity of palm kernel in storage, and one for the current stocking rate. Next, we simulate the optimal policy of the extended formulation for discretised combinations of stocking rate ρ and quantity of palm kernel Q. This results in a function $\Gamma(\rho, Q)$, that represents the maximum quantity of milk that can be produced given the stocking rate ρ and quantity of palm kernel Q. Third, we calculate the operating profit function $\Pi(p^m, \rho, Q)$ using the milk production function $\Gamma(\rho, Q)$ that we compute in the second step. Finally, we solve both the Hazard-Decision and Decision-Hazard models (i.e. Eq. 7.3 and 7.4) by enumeration. We now describe each of the four steps in more detail.

Step One: extended formulation

We extend the formulation of MOO by adding two new state variables. They are:

(5) x_t^q, the quantity of palm kernel in storage at the start of week t; and
(6) x_t^ρ, the stocking rate at the start of week t.

The quantity of palm kernel in storage at the start of week t has the corresponding dynamic:

$$x_{t+1}^q = x_t^q - 7 \times x_t^\rho \times u_t^s,$$

where u_t^s is the quantity of palm kernel fed per cow per day.

The dynamic of the stocking rate is worth explaining before we introduce it. For this two-stage model, we are only concerned with different stocking rates that are fixed during each season. One way to solve this problem would be to solve a separate instance of MOO (containing x_t^q) for discretised set of stocking rates in parallel. However, our solution library, `DynamicProgramming.jl`, already includes an efficient parallelisation scheme. Therefore, we introduce x_1^ρ as an additional state variable and we set up the dynamics so that it does not change over the season:

$$x_{t+1}^\rho = x_t^\rho.$$

The first stage decisions x_1^ρ and x_1^q correspond to the initial decisions ρ and Q described above. Furthermore, since we assume that the farmer purchases a fixed quantity x_1^q of palm kernel at the start of the year, we remove the cost of purchasing palm kernel from the stage-objective.

Like the case study in the previous section, the extended MOO model was solved using the `DynamicProgramming.jl` package (see Appendix A) in the Julia language [28]. In initial experiments, we used the same discretisation as that listed in Table 7.2 (in addition to discretising the two new state variables). This discretisation resulted in 14,030,610 discrete points in the state-space in each stage (compared with 55,022 previously). Thus, the solution time would have been on the order of days instead of 30 minutes. This is an example of the well-known "curse-of-dimensionality" for dynamic programming. In order to solve the model in a reasonable amount of time, we chose the coarser discretisation listed in Table 7.5. In each stage, there are 3,652,110 discrete points in the state-space and 154 possible controls. Since we need to evaluate the cost of each control at every discrete point in the state-space for each stage $t = 1, 2, \ldots, 51$, this results in $\approx 2.9 \times 10^{10}$ evaluations of the transition and cost functions. (Note that we do not need to evaluate the cost-to-go of the 52nd stage since $V_{53}(\cdot) = 0$.) The discretised model was solved using a virtual machine with 20 Intel E5-2698 CPUs with 64GB of memory. The total solution time was approximately 8 hours.

Step Two: construct milk surface

Once the model is solved, we simulate the optimal policy given the initial conditions $x_1^b = 3.2$, $x_1^e = 70$, $x_1^l = 1$, and $x_1^g = 2500$, for all $x_1^\rho \in \{2.5, 2.05, 2.1, \ldots, 4.5\}$ and $x_1^q \in$

Variable	Discretisation
x^b	2.50, 2.55, 2.60, . . . , 4.00
x^e	40, 45, 50, . . . , 90
x^l	0, 1
x^g	1500, 1600, 1700, . . . , 3500
x^ρ	2.5, 3.0, 3.5, 4.0, 4.5
x^q	0, 200, 400, . . . , 10000
u^s	0, 1, . . . , 6
u^g	0, 5, 10, . . . , 50
u^l	0, 1

Table 7.5: Discretisation of the state and control variables for the two-stage model.

$\{0, 100, 200, \ldots, 10000\}$. For each simulation of the optimal policy, we calculate the maximum quantity of milk solids that can be produced given the stocking rate x_1^ρ and quantity of palm kernel x_1^q:

$$\Gamma(x_1^\rho, x_1^q) = x_1^\rho \sum_{t=1}^{52} M_t(\hat{x}_t, \hat{u}_t), \tag{7.5}$$

where $\hat{x}_t$ and $\hat{u}_t$ are the optimal controls as a result of simulating the optimal policy with the stocking rate x_1^ρ and quantity of palm kernel x_1^q. This surface is plotted in Figure 7.6. The surface is similar to our intuitive example (Figure 7.1).

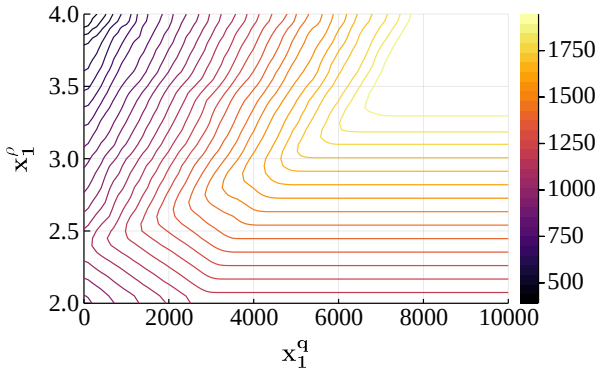


Figure 7.6: Contour plot showing the total quantity of milk produced (kg/ha/year) as the total quantity of palm kernel x_1^q (kg/ha) and the stocking rate x_1^ρ (cows/ha) are varied.

If we choose a fixed stocking rate of $x_1^\rho = 2$ cows/ha and increase the quantity of palm kernel x_1^q from 0, each additional kilogram of palm kernel can be converted by the cows into milk. Therefore, milk production increases. However, once we exceed ≈ 3000 kg of palm kernel, each cow is producing the maximum biological quantity of milk, so each additional kilogram of palm kernel does not increase the quantity of milk produced.

If we choose to feed zero palm kernel (i.e. $x_1^q = 0$), and then increase the stocking rate x_1^ρ from 2 cows/ha, milk production initially increases. This is because there is sufficient grass growth over the season to support the additional cows that produce the additional milk. However, once the stocking rate exceeds $x_1^\rho = 2.4$ cows/ha, the herd fully consumes the grass grown

over the season, and each additional cow requires feed for maintenance, which decreases the quantity of feed available for milk production and therefore total milk production.

If we choose a quantity of $x_1^q = 4000$ kg of palm kernel and then increase the stocking rate x_1^ρ from 2 cows/ha, the quantity of milk produced increases linearly at a steep rate. This is because every cow is producing the biological maximum quantity of milk. Once we increase the stocking rate above $x_1^\rho = 2.6$ cows/ha, the quantity of milk produced continues to increase, but at a slower rate. In this region, all of the feed is consumed (i.e. $x_{53}^g = 2500$ kg/ha), but the marginal contribution to the total milk production by each additional cow is positive. However, once we increase the stocking rate above $x_1^\rho = 2.9$ cows/ha, the marginal contribution to the total milk production by each additional cow is negative, and the total quantity of milk produced decreases.

Finally, in the top right-hand corner of the plot (i.e. when x_1^q and x_1^ρ are large), each cow is fed palm kernel at a maximum quantity of 6 kg/day (because of our discretisation of u_t^s). If we increased this maximum limit, the plateau would be lifted higher.

Step Three: construct operating profit surface

The next step is to calculate the maximum operating profit that can be earned given the stocking rate x_1^ρ and quantity of palm kernel x_1^q.

There are three components to operating profit: milk revenue, palm kernel cost, and the stocking rate cost. We have already discussed the first two components: milk revenue is the quantity of milk produced multiplied by the milk price, and palm kernel cost is the total quantity of palm kernel purchased (x_1^q) multiplied by the palm kernel cost (c^s). However, the stocking rate cost is more complicated. As the stocking rate increases, farm operating expenses increase. Macdonald et al. [134] give a breakdown of different farm expenses per hectare by stocking rate. We fitted a linear regression line through the data to obtain an estimate for the fixed cost (the intercept) per hectare ($1467/ha) and the variable cost (the slope) of the number of cows per hectare ($589/cow/ha). Therefore, the operating profit $\Pi(p^m, x_1^\rho, x_1^q)$ given the milk price p^m, stocking rate x_1^ρ, and quantity of palm kernel x_1^q is:

$$\Pi(p^m, x_1^\rho, x_1^q) = \underbrace{p^m \times \Gamma(x_1^\rho, x_1^q)}_{\text{milk revenue}} - \underbrace{(589 \times x_1^\rho + 1467)}_{\text{cost of stocking rate}} - \underbrace{c^s \times x_1^q}_{\text{cost of palm kernel}}, \tag{7.6}$$

where c^s is the cost of palm kernel ($0.5/kg as before).

Step Four: solve via enumeration

Finally, we solve, via enumeration, Eq. 7.3 and Eq. 7.4 using the eight prices listed in Table 7.4 and the discretisations for x_1^ρ and x_1^q used in Step Two. For example, the Hazard-Decision

problem is:

$$\text{Hazard-Decision} = \mathbb{E}_{p^m}\left[\max_{x_1^\rho \in P,\, x_1^q \in Q} \{\Pi(p^m, x_1^\rho, x_1^q)\}\right], \tag{7.7}$$

where $P = \{2.5, 2.05, 2.1, \ldots, 4.5\}$ and $Q = \{0, 100, 200, \ldots, 10000\}$ from Step Two.

7.4.2 Results

In Table 7.6, we give the solution of the two-stage models under the Hazard-Decision (HD) and Decision-Hazard (DH) assumptions. For each of the eight seasons (2009/10 to 2016/17 inclusive) we give the Fonterra end-of-season milk price p^m, the optimal stocking rate x_1^ρ, the optimal quantity of palm kernel x_1^q, and the operating profit Π.

	Season	09/10	10/11	11/12	12/13	13/14	14/15	15/16	16/17
	p^m (\$/kg)	6.10	7.60	6.08	5.84	8.40	4.40	3.90	6.12
HD	x_1^ρ (cows/ha)	3.25	3.3	3.25	3.2	3.45	2.75	2.4	3.25
	x_1^q (t/ha)	6.2	6.4	6.2	6.0	7.0	3.0	0.0	6.2
	Π (\$/ha)	4846	7650	4809	4368	9183	1935	1273	4883
DH	x_1^ρ (cows/ha)	3.2	3.2	3.2	3.2	3.2	3.2	3.2	3.2
	x_1^q (t/ha)	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
	Π (\$/ha)	4845	7599	4809	4368	9067	1725	807	4882

Table 7.6: Solution of the two-stage models under the Hazard-Decision (HD) and Decision-Hazard (DH) assumptions.

By definition, the Decision-Hazard model makes a single decision for the stocking rate x_1^ρ and the quantity of palm kernel to purchase x_1^q before learning the milk price p^m. Therefore, the stocking rate ($x_1^\rho = 3.2$ cows/ha) and initial quantity of palm kernel to purchase ($x_1^q = 6$ t/ha) are constant across all eight seasons. In contrast, the Hazard-Decision model can choose a different stocking rate and quantity of palm kernel in each season. When the milk price is high (e.g. \$8.40/kg in 2013/14), the optimal decision is to have a stocking rate of 3.45 cows/ha and to purchase 7 t/ha of palm kernel. When the milk price is low (e.g. \$3.90/kg in 2015/16), the optimal decision is to have a stocking rate of 2.4 cows/ha and to purchase 0 t/ha of palm kernel.

In three of the seasons (2009/10, 2011/12, and 2016/17), the Hazard-Decision solution ($x_1^\rho = 3.25$ cows/ha, $x_1^q = 6.2$ t/ha) is very similar to the Decision-Hazard solution ($x_1^\rho = 3.2$ cows/ha, $x_1^q = 6$ t/ha). However, despite the higher intensity, the operating profits differ by at most \$1/ha (e.g. \$4846/ha compared with \$4845/ha in 2009/10). This suggests that the two policies are nearly identical. In the 2012/13 season, the Hazard-Decision solution is identical to the Decision-Hazard solution. We call the policy of $x_1^\rho = 3.25$ cows/ha and $x_1^q = 6.2$ t/ha the *medium intensity* policy. We provide a plot of the simulated *medium intensity* policy in Appendix D.

In two of the remaining seasons (2010/11 and 2013/14), the optimal Hazard-Decision solution is more intensive than the Decision-Hazard solution. In 2010/11, the optimal solution is

$x_1^\rho = 3.3$ cows/ha and $x_1^q = 6.4$ t/ha. Compared with the Decision-Hazard solution, the operating profit increases by \$51/ha from \$7599/ha to \$7650/ha. In 2013/14, the optimal solution is even more intensive: $x_1^\rho = 3.45$ cows/ha and $x_1^q = 7$ t/ha. Compared with the Decision-Hazard solution, the operating profit increases by \$116/ha from \$9067/ha to \$9183/ha. We call the policy of $x_1^\rho = 3.45$ cows/ha and $x_1^q = 7$ t/ha the *high intensity* policy. We provide a plot of the simulated *high intensity* policy in Appendix D.

In the remaining two seasons (2014/15 and 2015/16), the Hazard-Decision solution is less intensive than the Decision-Hazard solution. In these two seasons, the milk price is lower than average (\$4.40/kg in 2014/15 and \$3.90/kg in 2015/16). Therefore, compared with the stocking rate of the Decision-Hazard solution (3.2 cows/ha), the Hazard-Decision model chooses a low stocking rate of 2.75 cows/ha in 2014/15 and 2.4 cows/ha in 2015/16. The Hazard-Decision model also purchases less palm kernel than the Decision-Hazard model (3 t/ha in 2014/15 and 0 t/ha in 2015/16, compared with 6.0 t/ha). We refer to the solution for the 2015/16 season as the *low intensity* policy. We provide a plot of the simulated *low intensity* policy in Appendix D.

In Table 7.7, we summarise the three policies of varying intensity and show the average of the historical DairyBase data from the farm (i.e. from Table 7.1). As the policy intensifies, the stocking rate, the quantity of palm kernel consumed, and the quantity of milk production per hectare all increase. However, the quantity of grass from pasture consumed per cow decreases. As a result, the policy that maximises the total energy intake (grass + palm kernel) per cow, and therefore per cow milk production, is the medium intensity policy. This highlights the trade-off between the stocking rate and the quantity of feed available per cow that we explored in Section 7.1.

	Policy Intensity			
	Low	Medium	High	Historical
Stocking Rate	2.4	3.2	3.45	3
Lactation Length (weeks)	39	39	39	38
Milk Production (kg)				
per Hectare	1061	1865	1918	1193
per Cow	442	574	556	398
Pasture Consumed (t)				
per Hectare	11.94	12.09	12.05	12.15
per Cow	4.98	3.78	3.49	4.05
Palm Kernel Consumed (t)				
per Hectare	0.0	6.00	7.00	2.85
per Cow	0.0	1.88	2.03	0.95
% Feed Imported	0.0	33.2	36.7	19.0

Table 7.7: Summary of the low, medium, and high intensity policies.

The value of price uncertainty

In Table 7.8 we give the expected and worst-case operating profit for each model, as well as the standard deviation of the operating profit. Because the Hazard-Decision model can choose a different (x_1^ρ, x_1^q) for each price scenario, it has the largest expected profit ($4868/ha compared with $4762/ha). In addition, it also performs better in the worst-case ($1273/ha compared with $807/ha).

Type	Expectation	Std. Deviation	Worst-Case
Hazard-Decision	4868	2622	1273
Decision-Hazard	4762	2716	807

Table 7.8: Operating Profit ($/ha) under the Hazard-Decision and Decision-Hazard assumptions.

The expected value of perfect information (EVPI) is the maximum cost the decision maker should be willing to pay to learn about the random outcome before it is realised [30]. For the farmer, the EVPI is the amount they should be willing to pay at the start of the season to learn the end-of-season milk price. The EVPI can be calculated as the difference between the expected operating profit of Hazard-Decision (HD) solution and the expected operating profit of Decision-Hazard (DH) solution. Therefore, the EVPI is:

$$\text{EVPI} = \text{HD} - \text{DH} = \$4868/\text{ha} - \$4763/\text{ha} = \$105/\text{ha}.$$

This suggests that a farmer should be willing to pay $105/ha in order to find out the milk price at the start of the season. In other words, on average, knowing the milk price at the start of the season allows the farmer to increase their expected profit by 2.2%. However, the increase in profit between the Hazard-Decision and Decision-Hazard solutions is not distributed evenly with respect to the end-of-season milk price. In Figure 7.7 we plot the difference between the operating profit of the Hazard-Decision and Decision-Hazard policies against the end-of-season milk price. The benefit of using the Hazard-Decision policy over the Decision-Hazard policy is largest (+$466/ha) when the end-of-season milk price is low (e.g. $3.90/kg). Therefore, risk-averse farmers, who are more concerned about their downside risks, will value perfect information more than a risk-neutral farmer. In the next section we build upon this finding and analyse the decision-making of risk-averse farmers.

7.4.3 Risk

In the previous analyses, we assume that the farmer maximises their profit in expectation. However, this can lead to policies that perform badly in the extreme cases. For example, in the Decision-Hazard solution, the optimal decision was to purchase a large quantity of palm kernel (x_1^q =6 t/ha) and run a high stocking rate (x_1^ρ =3.2 cows/ha), regardless of the milk price. This

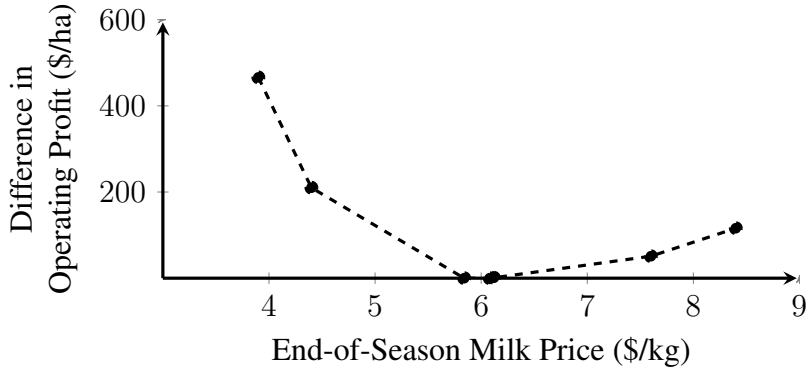


Figure 7.7: Difference in operating profit between the Hazard-Decision and Decision-Hazard policies against the end-of-season milk price. Positive differences are in favour of the Hazard-Decision policy.

policy performs well in expectation, but in years with a low milk price (e.g. $3.90/kg in the 2015/16 season) it can perform poorly. In these scenarios, farmers are forced to either borrow more money from the bank or reduce spending on items such as animal health. Consequently, some farmers may be averse to such downside outcomes.

To illustrate the effects of risk aversion, we present an example using the exponential utility function [82]:

$$U_\alpha(x) = 1 - e^{-\alpha x/1000}, \tag{7.8}$$

which maps the operating profit x to the risk-adjusted operating profit $U_\alpha(x)$. As the risk aversion level α increases, the utility function becomes more risk-averse.

We construct the risk-adjusted Decision-Hazard problem by mapping the profit achieved by the farmer through the exponential utility function:

$$\text{risk-adjusted Decision-Hazard} = \max_{x_1^\rho, x_1^q} \left\{ \mathbb{E}_{p^m} \left[U_\alpha \left(\Pi(p^m, x_1^\rho, x_1^q) \right) \right] \right\}. \tag{7.9}$$

The risk-adjusted Decision-Hazard problem was solved three times for three different levels of the risk aversion parameter α: 0.1, 1, and 10. In Figure 7.8 we plot the optimal (x_1^ρ, x_1^q) pairs for each of these problems. On the x-axis, we plot the total quantity of palm kernel to purchase x_1^q. On the y-axis, we plot the stocking rate x_1^ρ. Also plotted are contours of the expected profit (Figure 7.8a) and standard deviation in profit (Figure 7.8b).

For low levels of risk aversion ($\alpha \leq 0.1$), the optimal solution is to purchase 6 t/ha of palm kernel and run a stocking rate of 3.2 cows/ha. This is the same solution as the risk-neutral Decision-Hazard solution. As the risk aversion parameter α is increased, the optimal decision is to purchase less palm kernel and decrease the stocking rate. This decreases the expected operating profit, but it also decreases the standard deviation in operating profit. This reinforces the findings from Dairy NZ [49] that showed that lower intensity farms had lower average profits, but that they also a lower variance in operating profit between seasons.

For high levels of risk aversion ($\alpha \geq 10$), the optimal solution is to purchase 0 t/ha of palm kernel and run a stocking rate of 2.4 cows/ha. This is because the uncertainty was derived from the eight historical observations of the Fonterra end-of-season milk price (which were all assumed to be equally likely). This policy is optimal for the worst-case scenario in that set of eight observations ($3.90/kg in the 2015/16 season). Therefore, increasing the level of risk-aversion above $\alpha = 10$ does not change the policy.

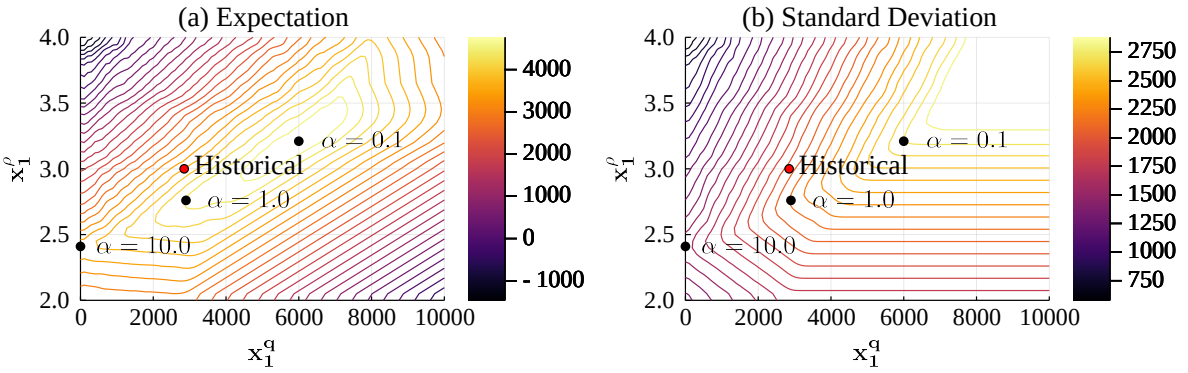


Figure 7.8: Contour plots showing (a) the expected operating profit, and (b) the standard deviation in operating profit, as the stocking rate x_1^ρ and quantity of supplement x_1^q are varied. Also plotted are the optimal solutions for varying levels of α in the exponential utility function, as well as the historical set-up of the farm.

In Table 7.9 we give, for each of the risk-adjusted Decision-Hazard problems, the expected operating profit from each model, the standard deviation of the operating profit, and the operating profit obtained in the worst-case. When the risk aversion parameter $\alpha = 0.1$, the solution is identical to the risk-neutral Decision-Hazard solution. As the level of risk aversion increases, the expected profit decreases, the standard deviation decreases, and the profit obtained in the worst-case increases. In expectation, the most risk-averse solution ($\alpha = 10$) is $3568 - 4762 = -\$1194$/ha worse off compared with the least risk-averse solution ($\alpha = 0.1$). However, in the worst-case (i.e. when the milk price was $3.90/kg), the most risk-averse solution ($\alpha = 10$) is better off than the least risk-averse solution by $1273 - 807 = \$466$/ha. Therefore, when choosing the stocking rate and quantity of palm kernel to purchase, the farmer needs to trade-off doing well in expectation against doing well in the bad outcomes. This trade-off will depend on their aversion for risk, as measured by α.

Parameter	Risk-Aversion	Expectation	Std. Deviation	Worst-Case
$\alpha = 0.1$	Low	4762	2716	807
$\alpha = 1$	Medium	4369	2176	1199
$\alpha = 10$	High	3568	1576	1273

Table 7.9: Operating profit ($/ha) when optimised using different utility functions.

7.5 Discussion

In this chapter we applied MOO to the farm described in Section 7.3. From the results we obtain, it is apparent that the current farm setup is sub-optimal. If the current stocking rate (3 cows/ha) is maintained, the expected operating profit could be increased by increasing the supplementation intensity from 2.85 t/ha to 5 t/ha. Alternatively, if the same level of supplementation was maintained (i.e. 2.85 t/ha), the expected operating profit could be increased by decreasing the stocking rate from 3 cows/ha to 2.75 cows/ha. Before we performed this analysis, the farmer decided to reduce their stocking rate to 2.7 cows/ha in preparation for the 2015/16 season. This decision is close to that recommended by MOO. However, we do not have the data (e.g. Table 7.1) needed to analyse the impact of this decision.

This type of analysis, aimed at improving the profitability of a single dairy farm, has been performed elsewhere in the literature (e.g. in the Waikato region of New Zealand by Doole [61], and on three farms in Australia in response to the threat of climate change by Harrison et al. [100]). However, like our analysis in Section 7.3, but unlike our analysis in Section 7.4, these authors used a fixed milk price. This is a large limitation since we showed through the Hazard-Decision model that the optimal intensification decision is dependent upon the milk price. This observation is not novel and has been observed multiple times in the literature (e.g. in New Zealand [19, 59], Spain[4], and the United Kingdom [204]). Moreover, optimising intensification decisions given a fixed milk price (e.g. the expected milk price) ignores the volatility in operating profit associated with an uncertain milk price.

Our analysis of the last 18 years of data from Dairy NZ [49] showed that the yearly operating profit of the average owner-operator (i.e. a farmer who owns the land) had a coefficient of variation[4] of 60%. In comparison, the coefficient of variation of the operating profit for the farm over the last eight years using the Decision-Hazard solution (i.e. Table 7.6) was 57%. However, the coefficient of variation of the last eight years of milk prices (Table 7.4) was only 24%. The additional volatility comes from the decision to purchase a fixed quantity of palm kernel on the first day of the year. When milk prices are high, the additional feed increases the operating profit by increasing the total quantity of milk that can be produced. However, when milk prices are low, the cost of the palm kernel is greater than the benefit of the extra milk, reducing the operating profit. This observation, that purchasing palm kernel increases volatility in operating profit, explains why risk-averse farmers decrease the intensity of their farming operations. It has also been observed by Brazendale [33], who noted that despite their use for combating adverse weather events, supplements can expose a farm business to greater financial risk if not used appropriately.

Finally, despite the large variation in operating profit, the expected value of perfect information (EVPI) for the farm is only $105/ha. This represents a gain in expected operating profit of 2.2% if the farmer knew the end-of-season milk price at the start of the season. However,

[4]The standard deviation divided by the mean.

this value might be greater if more recourse decisions were available for the farmer, such as the ability to change their stocking rate during the season, or the ability to purchase palm kernel via the spot market instead of making a single bulk purchase at the start of the season.

7.6 Summary

In this chapter we have shown how the *E-Cow* system of nonlinear equations can be embedded within a dynamic program and solved to find optimal feeding strategies. We applied the resulting dynamic program, which we call MOO, to a farm in the Bay of Plenty region of New Zealand. Our analysis suggests that the operation of this farm is currently sub-optimal, and that profitability could be improved, either by increasing the supplementation intensity or by reducing the stocking rate. It is interesting to note that independently of this work, the farmer elected to reduce their stocking rate from 3 cows/ha to 2.7 cows/ha (instead of the 2.75 cows/ha recommended by MOO) in preparation for the 2015/16 season.

Moreover, the chapter reinforces the finding that high-intensity farms are only the most profitable when milk prices are high. Notably, this chapter differs from the existing literature by modelling the intensification decision as a risk-averse two-stage stochastic program. We demonstrate the intuitive solution that, as the farmer becomes more risk-averse, they decrease the intensity of their farming operation.

Although we have demonstrated that MOO is a useful tool for practical farm management, the biggest limitation of the model is solution time. The model developed in Section 7.4 took eight hours to solve on a virtual machine with 20 CPUs. Thus, due to the "curse of dimensionality," we are limited in our ability to add new features to MOO. For example, if we were to add stochastic grass growth with 10 possible realisations per stage, the solution time would be on the order of 3.5 days. Therefore, in the next two chapters we formulate a novel multistage stochastic optimisation model of a New Zealand dairy farm that uses a simplified piecewise linear approximation of a dairy cow instead of the detailed nonlinear *E-Cow* model. This allows us to solve the problem using the stochastic dual dynamic programming algorithm, which we discuss in the first half of this thesis.

Supplementary materials The Julia code to implement MOO, as well as the data needed to run the case study and replicate the results in this chapter, is available at `https://github.com/odow/MOO`.

Chapter 8

A multistage stochastic optimisation model of a pastoral dairy farm

In this chapter we present POWDER: the Milk Production Optimiser incorporating Weather Dynamics and Economic Risk. POWDER is a novel multistage stochastic programming model of a pastoral dairy farm that incorporates environmental and economic uncertainty. Such a model represents a significant step forward in the ability to gain insight into the interrelationships that weather and price uncertainty have on the decision-making of dairy farmers. Readers should note that despite the reference to economic risk in the name of our model, we defer our analysis of risk to the next chapter.

This chapter is laid out as follows. In Section 8.1 we formulate POWDER as a multistage stochastic optimisation problem. Then, in Section 8.2 we apply POWDER to the farm from Chapter 7. Finally, we finish the chapter with a discussion in Section 8.3 and some summarising remarks in Section 8.4.

8.1 Formulation

In this section we formulate POWDER as a multistage stochastic optimisation problem over 52 weeks, $t = 1, 2, \ldots, 52$. We call the sequence of 52 weeks a *season*. POWDER is a combination of four separate models: a weather model, a grass growth model, an animal model, and a milk price model. Before we detail the specifics of POWDER, we provide an overview in Figure 8.1 using the diagramming conventions of System Dynamics [192]. States are in square boxes. The three random inputs are shown as arcs across the model boundary and are named in bold italicised text. Controls that can be chosen by the farmer are named in bold underlined text, and control variables that cannot be chosen by the farmer are named in a standard font. The weather model is coloured blue, the grass growth model is coloured green, the animal model is coloured brown, and the price model is coloured orange.

We now describe the model in detail, beginning with the weather and grass growth models.

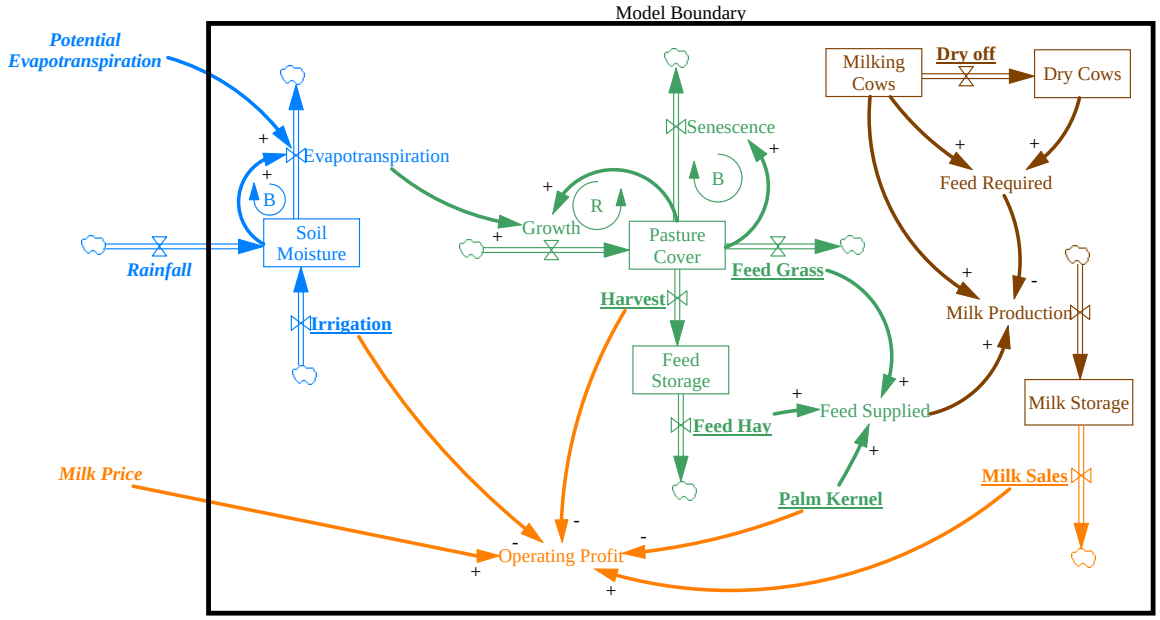


Figure 8.1: System Dynamics diagram of the POWDER model.

The equations we present form the transition function and set of feasible controls that we defined in Chapter 1. A complete formulation is given as an appendix item in Appendix E.

8.1.1 Grass growth and weather models

The basis for any whole-farm model in a pastoral farming context is a model for grass growth. For New Zealand conditions, there is a large literature on different modelling approaches [36, 72, 114, 138, 175]. However, many of these models are highly detailed and nonlinear (and non-convex). Thus, the technical requirements of SDDP prevent us from using these models. In this section, we introduce two previously published models of grass growth. The first relates grass growth to the total quantity of grass on the farm (since if there is no grass, none can grow), but excludes any weather effects such as drought. The second relates grass growth to the weather (via evapotranspiration[1]) but excludes the total quantity of grass on the farm. Therefore, we propose a new model that assumes that the two models act independently to limit grass growth.

The first model assumes that grass growth is related to the current pasture cover and has been used before in a New Zealand setting [19, 80]. The model is a logistic growth function so that:

$$P_{t+1} - P_t = 7 \times \frac{4\Delta P_{\max}}{P_{\max}} \times P_t \left(1 - \frac{P_t}{P_{\max}}\right), \tag{8.1}$$

where P_t is the pasture cover (kg/ha) at the start of week t, the constant $P_{\max}$ is the maximum

[1]Evapotranspiration is the sum of both direct evaporation (from soil to atmosphere) and plant transpiration (where the water moves from the soil through the plant and evaporates from the parts exposed to the atmosphere such as the leaves).

possible pasture cover (the point at which the senescence[2] rate approaches the growth rate [19]), and the constant $\Delta P_{\max}$ is the maximum possible rate of pasture growth (kg/day). The multiplication by 7 converts from a daily growth rate to a weekly growth rate. The daily growth rate attains the maximum of $\Delta P_{\max}$ when $P_t = P_{\max}/2$.

In contrast to the logistic growth function, Moir et al. [138] propose that grass growth is proportional to the evapotranspiration rate such that:

$$P_{t+1} - P_t = \kappa_t \times e_t, \tag{8.2}$$

where P_t is the pasture cover (kg/ha) at the start of week t, e_t is the evapotranspiration rate (mm/week) during week t, and κ_t is a constant (kg/ha/mm) that can be interpreted as an index of soil fertility. One weakness of this model is that it ignores the impact that the current pasture cover has on the growth rate. Evapotranspiration depends on a number of factors. First, there must be sufficient water in the soil for evapotranspiration to occur. Second, more water will evaporate and transpire on a hot, sunny day than a cold, cloudy day. Therefore, the evapotranspiration rate, e_t, during week t is:

$$e_t = \min\left\{e_t^p,\ W_t + r_t\right\}, \tag{8.3}$$

where e_t^p is the *potential* evapotranspiration rate [167] during week t (mm/week) as determined by the weather during week t, W_t is the plant-available water stored in the root zone of the soil at the start of week t (mm), and r_t is the incident rainfall during week t (mm/week). Both the evapotranspiration rate e_t^p and the incident rainfall r_t are random variables in week t.

Farmers sub-divide their farms into a number of individual fenced units called paddocks. The herd is rotated around the paddocks at a rate of one to two paddocks per day, depending on growth rate of the grass. This leaves each paddock with a different pasture cover. In this chapter, we make the simplifying assumption that all paddocks are identical. Therefore, P_t represents the average pasture cover across the entire farm at the start of week t.

One extra feature of pasture is that it can be harvested to form hay or silage.[3] This can be stored until later in the season when the pasture cover is low. We denote the quantity of feed that is in storage at the start of week as Q_t (kg). Therefore:

$$Q_{t+1} = Q_t + \beta h_t - f_t^q, \tag{8.4}$$

where h_t is the quantity of pasture harvested in week t (kg), and f_t^q is the quantity (kg) of the stored pasture fed to the herd in week t (discussed in Section 8.1.2). β is a constant conversion factor between pasture and feed in storage and accounts for spoilage and wastage.

To incorporate the two models of pasture growth (Eq. 8.1 and Eq. 8.2), we assume that both

[2]The gradual deterioration of a cell due to ageing.

[3]A fermented, high-moisture alternative to hay.

of these models act independently to limit grass growth. Our final dynamical model for grass growth is:

$$P_{t+1} = P_t + \min\left\{\kappa \times e_t,\ 4\frac{\Delta P_{\max}}{P_{\max}} \times P_t\left(1 - \frac{P_t}{P_{max}}\right)\right\} - h_t - f_t^p, \tag{8.5}$$

where f_t^p is the quantity of grass from pasture fed to the herd in week t (we discuss this further in Section 8.1.2).

In addition to proposing that grass growth is proportional to evapotranspiration, Moir et al. [138] give some simple water balance equations to describe how the water in the soil changes over time. First, there is some maximum quantity of water that can be stored in the soil (and is available to the plant):

$$W_t \leq \bar{W}, \tag{8.6}$$

where $\bar{W}$ is a constant equal to the maximum available water content of the soil. Any excess water is assumed to drain away. Second, the change in soil moisture W_t from the start of week t to the start of week $t+1$ is limited by the quantity of rainfall r_t, plus irrigation i_t, less evapotranspiration e_t:

$$W_{t+1} \leq W_t + r_t - e_t + i_t. \tag{8.7}$$

There are also some domain constraints on the state and control variables, such that:

$$P_t, Q_t, W_t, f_t^p, f_t^q, h_t, i_t \geq 0,\ \forall t. \tag{8.8}$$

Stochastic process

For simplicity, we assume that the rainfall r_t, and potential evapotranspiration e_t^p, can be modelled by a stagewise-independent joint stochastic process that empirically samples with replacement from historical readings in week t. We call a single sequence of 52 observations (one for each week) of the random weather variables, r_t and e_t^p, a *weather scenario*. The stochastic process can include some stagewise dependence with an increase in complexity of the solution process. (See Morton and Infanger [139] and Shapiro [187] for different approaches to this.) However, we choose to model the process using stagewise independence as it is a commonly used assumption in the SDDP literature [153, 157].

8.1.2 Animal model

Animal models are another critical component of any whole-farm model and are well studied in the literature [18, 88, 113]. However, like the grass growth models mentioned earlier, these are complicated nonlinear models that account for a large number of variables. Due to a desire to maintain tractability of the model with the inclusion of stochasticity, we seek a simplified version of these models.

The first decision a farmer faces is their stocking rate (number of cows per hectare). In this model, we assume the stocking rate is fixed at the start of the season, and we denote it by $\bar{C}$. The season begins with the cow giving birth to a calf and starting lactation (we assume that the cost of raising the calves is a fixed cost, so it is not modelled directly in POWDER). At some point during the season, the farmer can choose to dry-off a cow (i.e. stop its lactation). This reduces the required energy intake for the cow (since it is no longer producing milk). However, once the cow is dried-off, it cannot restart its lactation. Therefore, the farmer faces a stopping problem; they have to trade-off the decision to keep milking the cow (and earning money) against managing feed reserves for the next season. We denote the number of lactating cows in week t as C_t. Each week t, the farmer can choose to dry-off u_t cows, so that:

$$C_{t+1} = C_t - u_t. \tag{8.9}$$

In addition, there are some domain constraints, such that: $C_t \geq 0$, $u_t \geq 0$, and $C_1 = \bar{C}$.

Now that we have the concept of a herd (as measured by the stocking rate) and the idea that cows must be in one of two states (lactating or dried-off), we create a simple model of a single cow. To do so, we consider an energy balance: energy consumed by the cow in the form of grass from pasture, or supplements such as hay and silage, must equal the energy spent on maintenance,[4] changes in body condition, pregnancy, and milk production. To simplify the model, we ignore other factors like nutrition.

Energy requirements

To calculate the energy required by the cow for maintenance, changes in body condition, and pregnancy, we draw from multiple published models in the literature. It is sufficient for general readers to understand that, as a result of the model, we can calculate the net energy required during week t by a cow (excluding milk production) that is lactating (which we denote by $\varepsilon_t^{\text{lac}}$) or dried-off (which we denote by $\varepsilon_t^{\text{dry}}$). These are set as constants in POWDER. In the following, we provide more detail about how we calculated the constants $\varepsilon_t^{\text{lac}}$ and $\varepsilon_t^{\text{dry}}$. Some readers may want to skip forward to the subsection titled *Milk Production*.

Maintenance Maintenance is the energy required to run the cow's core bodily functions. This can be considered a fixed cost of keeping the cow alive. In order to meet this requirement when the cow is underfed, energy is mobilised from body fats (decreasing the BCS) and milk production is reduced. In this model we consider the energy for maintenance to be constant. The farmer must provide sufficient energy input to meet this requirement. For a 450 kg cow, Dairy NZ [47] assume the maintenance requirement to be 54 MJ/day.

[4] Maintenance is the energy required to run the cow's core bodily functions. This can be considered a fixed cost of keeping the cow alive.

Body condition The Body Condition Score (BCS) is an assessment of the proportion of body fat in a cow and is used by farmers as an important indicator of animal health [177]. A five-point scale is used, where 1 represents a severely emaciated cow, and 5 represents a severely obese cow. Over a season, cows typically follow a genetically driven cycle in their Body Condition Score (although under- or over-feeding can cause the cow to deviate from the typical trajectory). The cycle begins after the cow calves and starts with the mobilisation of body fat into energy to contribute to the energetic cost of early lactation. This causes the BCS to decrease. A few weeks after conception for the next season (100 – 120 days into the season), the cow reverses this trend and begins to deposit body fat in preparation for the next calving. This causes the BCS to increase [78].

In this model, we assume that the BCS trajectory is known *a priori* and that the cow modifies its milk production to satisfy the energy balance, rather than a combination of both milk production and BCS change. These are reasonable assumptions, since deviating from the trajectory typically represents a poor health outcome for the cow. However, it also reduces the number of actions the farmer can take, since underfeeding the cow is no longer feasible. This is notably different to the *E-Cow* model in the previous chapter, where the BCS was a state variable that we could control over time.

For the POWDER model, we assume that the cow follows the body condition trajectory as calculated by the Friggens model[5] [78], given an initial body mass of 450 kg and an initial BCS of 3.2.

Next, we need to relate the change in BCS to the net energy requirements of the cow during a week. To do this, we draw from Dairy NZ [47], who assume that a one unit change in BCS corresponds to a change of 14.8% of the cow's body mass. However, the energy required to increase body mass by 1 kg is greater than the energy released by decreasing body mass by 1 kg, and this amount of energy depends upon whether the cow is lactating or dry (see Table 8.1). With this information we can calculate the contribution to the cow's energy balance due to changes in BCS for each week of the season.

	Lactating Cows	Dry Cows
1 kg gain in body mass	50	72
1 kg loss of body mass	-37	-30

Table 8.1: Energy required per change in body mass (MJ/kg) for cows that are lactating and dried-off [47].

Pregnancy Dairy NZ [47] give approximate energy requirements for pregnancy at 2, 4, 8, and 12 weeks before calving, as well as an annual total. To obtain a better estimate, we use the energy required for pregnancy model of SCA [185] (which is also used in *E-Cow*). However, it

[5]The "adjusted" parameter set (as it is called by Friggens et al. [78]) was used as it demonstrated a slightly better fit for their historical data.

is our belief that this equation is over-fitted.[6] If we assume $W = 47$ kg, we can fit, using simple linear regression, the exponential function:

$$\text{energy for pregnancy} = 0.2278e^{0.01989d}, \tag{8.10}$$

where d is the number of days since conception. The greatest absolute error between the fitted curve and the original equation is 0.46 MJ/day and, over a 284-day gestation, the simplified model predicts that the cow will require 6.5 MJ in additional energy for pregnancy. That is less than 1 kg of pasture over a season. In our view this small discrepancy does not justify the additional terms in the equation of SCA [185]. This model also gives similar results to the data given in Dairy NZ [47], which suggests that a cow needs 3240 MJ/year for pregnancy (compared to 3272 MJ/year from Eq. 8.10), and 21 MJ/day eight weeks before calving (compared to 21.2 MJ/day from Eq. 8.10). Using Eq. 8.10, we can calculate the energy (MJ/day) required by the cow for a given week t.

Total energy requirements In Figure 8.2, we plot the energy required by the cow over time when it is lactating ($\varepsilon_t^{\text{lac}}$) and dried-off ($\varepsilon_t^{\text{dry}}$). Of the total required energy, the fixed cost of maintenance (54 MJ/day) represents a large proportion. Towards the end of the season, the energy required for pregnancy also increases. The solid lines represent the net energy requirements for changes in body condition (BCS). At the start of the season, the cow loses body condition, so the required energy is negative. After reaching nadir around conception, the cow begins gaining body condition and the requirement is positive. Note that dry cows require more energy for gaining body condition than lactating cows due to the values in Table 8.1.

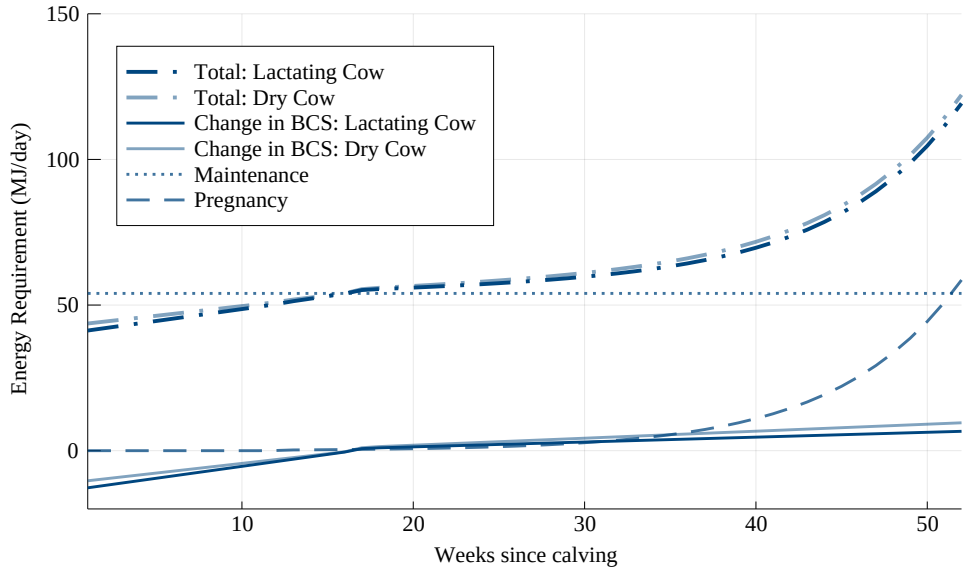


Figure 8.2: Energy required per day for lactating and dry cows over the season.

[6] The full equation is given (in MJ/day) as $9.663 \times 10^{-5} \times W^2 \times e^{-5.76\times10^{-5}d} \times 10^{151.665-151.64\times e^{-5.75\times10^{-5}d}}$, where d is the number of days since conception, and W is the birth weight of the calf (kg).

Milk production

If a lactating cow consumes more energy than the total required (i.e. $> \varepsilon_t^{\text{lac}}$), milk will be produced. Dairy NZ [47] assume that there is a linear relationship between this extra energy intake, which we denote m_t, and the quantity of milk produced. However, cows have a biological maximum milk production capacity that varies over the season. Therefore, following the approach taken by Baudracco et al. [18], we use the alveolar model of Vetharaniam et al. [201] to approximate the maximum production $\overline{\nu}_t$ (in MJ/week), as well as the net energy content of milk η_t^m as a function of the fat and protein composition based on the work of Freer et al. [77]. However, for simplicity, we retain the linear relationship between energy input and milk production. The maximum quantity of energy for milk production is scaled by the number of cows lactating so that the quantity of milk produced by the herd (kg/week) from the m_t MJ of extra energy intake is m_t/η_t^m, where the constant η_t^m is the net energy content of milk (MJ/kg). In addition, we impose bounds on the total energy m_t that can be used by the herd for milk production so that:

$$\underline{\nu}\, C_t \leq m_t \leq \overline{\nu}_t\, C_t, \tag{8.11}$$

where the constant $\underline{\nu}$ is the minimum biological rate of milk production (MJ/week) for a single lactating cow. A lower rate of milk production than $\underline{\nu}$ can lead to lower milk quality and poor animal health [46].

Supplementation

Palm kernel is a by-product of the production of palm oil [44]. It is commonly used as a supplementary feed (food that is fed in addition to grass) on New Zealand farms as it has a high energy content, is easily obtained (typically arriving on-farm within a day or two of ordering), and is relatively inexpensive. In this model we assume that the farmer orders a quantity s_t (kg) of palm kernel on the spot-market and feeds it to their cows during week t. For simplicity, we ignore the ability of the farmer to store palm kernel and engage in forward contracting programs. We also ignore other supplementary feeds such as maize.

Energy intake

Recall that in week t, the farmer feeds their herd f_t^p kg of grass from pasture[7] and f_t^q kg of grass from storage. In addition, let η^p be the metabolisable energy content of grass (both consumed from the paddock directly, and stored), and let η^s be the metabolisable energy content of palm kernel. The total energy consumed by the herd in week t is therefore $\eta^p \times (f_t^p + f_t^q) + \eta^s \times s_t$.

[7] Note that this is consumed directly, rather than the offered quantity as in Chapter 7.

Finally, we enforce an energy balance constraint for the entire herd so that:

$$\underbrace{\eta^p \times (f_t^p + f_t^q) + \eta^s \times s_t}_{\text{energy intake}} = \underbrace{C_t \times \varepsilon_t^{\text{lac}}}_{\text{lactating requirements}} + \underbrace{(\bar{C} - C_t) \times \varepsilon_t^{\text{dry}}}_{\text{dry requirements}} + \underbrace{m_t.}_{\text{milk production}} \tag{8.12}$$

In the energy balance equation, the actions of the farmer are the quantities of grass from pasture f_t^p, grass from storage f_t^q, and palm kernel s_t to feed. In response, the cows apportion m_t MJ/week to milk production. The reader should note that as a combination of Eq. 8.11 and Eq. 8.12, the actions of the farmer are constrained so that they must provide sufficient energy input to meet the minimum demand of the cow ($\varepsilon_t^{\text{dry}}$ if the cow is dry, $\varepsilon_t^{\text{lac}} + \underline{\nu}$ if the cow is lactating), but no more than the maximum energy demand of the cow ($\varepsilon_t^{\text{dry}}$ if the cow is dry, $\varepsilon_t^{\text{lac}} + \overline{\nu}_t$ if the cow is lactating).

In the next section we describe the price model within POWDER.

8.1.3 Price model

There are three components to the price model in POWDER: a milk price model, costs on the various controls chosen by the farmer, and penalty terms.

Milk price model

First, we describe a model for the price of milk. The contemporary New Zealand milk processing sector is dominated by Fonterra, a large processing co-operative that collects 84% of the milk produced in New Zealand [195]. Members of the co-operative are dairy farmers who share in the co-operative profits based on the number of kilogram of milk supplied during the season. The milk price is determined *ex-post* at the end of each year-long season. Therefore, during the season, farmers are uncertain about the price they will receive for milk they produce. Fonterra publish a forecast of the milk price at least every quarter, but these forecasts have large uncertainties of up to $\pm 50\%$ near the beginning of the season, with these forecasts becoming more accurate as the season progresses [207, 208]. We shall refer to the final price as the *end-of-season* milk price and the forecasts issued by Fonterra during the season as the *forecast* milk price.

At the start of each week $t = 1, 2, \ldots, 52$, farmers observe a forecast milk price p_t, provided by Fonterra, of the end-of-season milk price p_{53}. We assume that the first week's forecast is $p_1 = \$6/\text{kg}$ (the long-run average end-of-season milk price [76]). In the week after the end of the season (i.e. week 53), each farmer is paid the end-of-season milk price p_{53} for each kg of milk that they produced during the season. For simplicity, we shall model the sequence of forecast milk prices using nine equally likely scenarios. In each scenario, the forecast does not change until week 26, when we assume new information emerges leading to an update. No further updates are made until the final week when the end-of-season milk price p_{53} is realised. This process can be represented by a scenario tree which branches at two periods in time:

between weeks 25 and 26, and 52 and 53, as shown in Figure 8.3. We call a single sequence of 53 observations of p_t (one forecast for each week of the season, plus the end-of-season milk price) a *price scenario*.

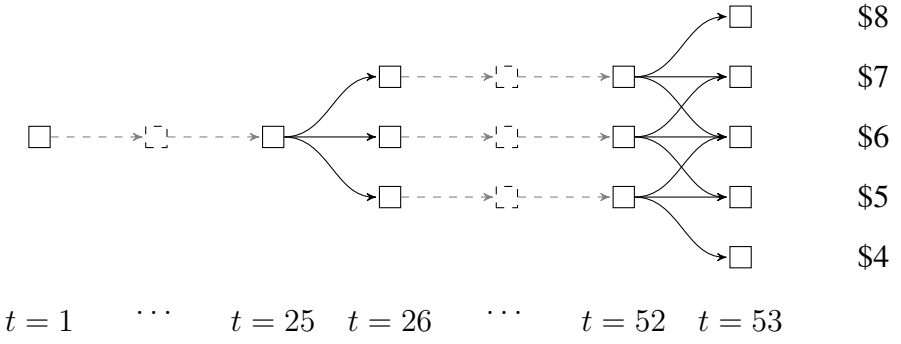


Figure 8.3: Scenario tree of milk price process.

As discussed above, we assume that the farmer sells all their milk in the final stage at the end-of-season milk price associated with the leaf node in the scenario tree. This necessitates the introduction of a new state variable to keep track of the quantity of milk produced to date:

$$M_{t+1} = M_t + m_t/\eta_t^m, \tag{8.13}$$

where M_t is the quantity of milk solids produced to date at the start of week t, and η_t^m is the net energy content of milk. In the final week $t = 53$, the farmer sells the total quantity of milk produced over the season M_{53} at the end-of-season milk price p_{53}.

Cost of controls

The second component of the price model in POWDER is the cost that the farmer incurs as a result of their controls. In each stage t, the farmer incurs a cost of:

$$c^s s_t + c^h h_t + c^i i_t, \tag{8.14}$$

where c^s is the cost of palm kernel (\$/kg), c^h is the cost of harvesting (\$/kg), and c^i is the cost of irrigation (\$/mm/ha).

Penalties

The third component of the price model is the penalty that the farmer incurs as a result of their controls. There are two types of penalty costs: a penalty on the final pasture cover, and a penalty on the quantity of supplement that is fed.

Pasture cover penalty Although POWDER is a single season model, dairy farming is a multi-year endeavour. Therefore, we need to ensure that the farm is in a better (or equal) condition at

the end of the season than it was at the start to ensure that the next season's performance is not affected. Therefore, we add an artificial variable $\Delta^p \geq 0$ to measure the amount by which the final pasture cover is less than the initial pasture cover:

$$P_{53} + \Delta^p \geq P_1. \tag{8.15}$$

Then, we penalise Δ^p by a large coefficient (arbitrarily chosen to be 1000) in the objective function of the final stage.

Fat Evaluation Index penalty Feeding palm kernel as a supplement to lactating cows causes a change in the ratio of the fatty acids in milkfat. When fed palm kernel at high levels, the milkfat produced by a cow becomes difficult to process to customer requirements [51]. To quantify this, a *Fat Evaluation Index* has been developed by a collaboration of industry groups. Beginning June 1st, 2018, New Zealand dairy farmers will have their milk graded into four Fat Evaluation Index grades: *A*, *B*, *C*, and *D*. An *A* grade is given to milk that is acceptable for processing. A *B* grade is given to milk that is approaching the acceptable limit for processing. A *C* grade is given to milk that exceeds the acceptable limit by a small amount. A *D* grade is given to milk that exceeds the limit by a large amount. Milk that is graded as a *C* or *D* will incur (currently unspecified) financial penalties. Current information released by the industry suggests that as a rule of thumb, three kilograms per cow per day is approximately the upper limit for maintaining an *A* or *B* grade [51]. To model the penalties associated with the excess intake of palm kernel, we introduce a penalty based on the quantity of palm kernel fed per cow per day (Figure 8.4). We denote the penalty associated with a high Fat Evaluation Index in stage t by Δ_t^q.

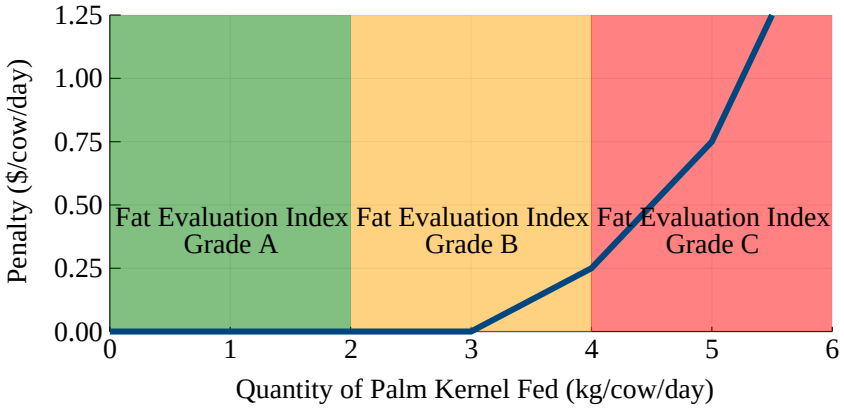


Figure 8.4: Fat Evaluation Index Penalty for differing levels of palm kernel intake. Shaded vertical bands represent different Fat Evaluation Index grades: A in green, B in orange, and C in red (D not shown).

Penalties for exceeding the Fat Evaluation Index threshold have not yet been decided and are likely to be based on the total milk produced, rather than palm kernel fed. As a convex cost

on palm kernel fed, our approximation preserves the convexity necessary for optimisation using SDDP.

8.1.4 Optimisation model

In Chapter 1, we described the components of a multistage stochastic optimisation problem: the policy graph, the states, the controls, the noise terms, the transition function, and the stage-objective. We now describe each of these in turn for POWDER. The full formulation of $\mathbf{SP}_{t,p_t}$ is given as an appendix item in Appendix E.

Policy graph The scenario tree in Figure 8.3 forms a *Markovian* policy graph with 53 stages: one Markov state in stages 1 to 25, three Markov states in stages 26 to 52, and five Markov states in stage 53. The probabilities on all the arcs are uniform in each stage. We denote each subproblem in the graph as $\mathbf{SP}_{t,p_t}$, where $t = 1, 2, \ldots, 53$, and p_t is the Markov state corresponding to the forecast milk prices \$4, \$5, ..., \$8.

Readers should note that for notational simplicity in the remainder of this chapter, we drop the subscript of the Markov state. Therefore, when we write x_t, we really mean x_{t,p_t}.

Noise terms At the beginning of each stage, the farmer observes the potential evapotranspiration e_t^p and the quantity of rainfall r_t. This makes the model a *Hazard-Decision* or *Wait-and-See* model. We assume that the farmer is able to observe these random variables at the start of the week in order to simplify the model. One justification for this simplification is that short-term weather forecasts are sufficiently accurate to negate the need to complicate the model.

States There are five state variables in the model: the soil moisture W_t, the pasture cover P_t, the quantity of feed in storage Q_t, the number of cows milking C_t, and the quantity of milk solids produced to date M_t.

Controls After the farmer has observed the state of the farm and the noise, the farmer needs to decide the quantity of irrigation to apply i_t, the quantity of pasture to harvest h_t, the number of cows per hectare to dry-off u_t, and the quantities of grass from pasture f_t^p, grass from storage f_t^q, and palm kernel s_t to feed the cows.

Transition function The transition function (and feasible set of controls) is implicitly defined by equations: 8.3–8.13.

Stage-objective In each stage, the farmer incurs the cost of irrigation, harvesting, and palm kernel purchases. They will also incur a Fat Evaluation Penalty if they feed too much palm

kernel. Therefore, the stage-objective in stages $t = 1, 2, \ldots, 52$ is:

$$-\left(c^s s_t + c^h h_t + c^i i_t + \Delta_t^q\right).$$

In the final stage problem $\mathbf{SP}_{53,p_{53}}$, the farmer sells the total quantity of milk produced during the season M_{53} at the end-of-season milk price p_{53} and pays any penalty for low pasture cover. Therefore, the stage-objective is:

$$p_{53} M_{53} - 1000\Delta^p.$$

Subproblems Putting all of this together, we can describe the subproblem $\mathbf{SP}_{t,p_t}$ for stages $t = 1, 2, \ldots, 52$ as the optimisation problem:

$$\mathbf{SP}_{t,p_t}:\ V_{t,p_t}(x_t, e_t^p, r_t) = \max\ \underbrace{\mathbb{E}_{p_{t+1}, e_{t+1}^p, r_{t+1}}\left[V_{t+1,p_{t+1}}(x_{t+1}, e_{t+1}^p, r_{t+1})\right]}_{\text{future profit}} - \quad \ldots$$
$$\ldots \quad \underbrace{\left(c^s s_t + c^h h_t + c^i i_t + \Delta_t^q\right)}_{\text{week } t \text{ cost}}$$

$$\text{s.t.} \quad \text{the transition function,}$$

where x_t denotes the five state variables (W_t, P_t, Q_t, C_t, M_t) at the start of the stage t. Note that the expectation is with respect to the two stagewise-independent noise terms, e^p and r, as well as the probability of transitioning from Markov state p_t to p_{t+1}, which is defined by the policy graph. The full formulation of $\mathbf{SP}_{t,p_t}$ is given as an appendix item in Appendix E.

In the final stage $t = 53$, the subproblems are different as the season has ended and all that occurs is the final payment of milk:

$$\mathbf{SP}_{53,p_{53}}:\ V_{53,p_{53}}(x_{53}) = \max\ \underbrace{p_{53} M_{53}}_{\text{milk profit}} - 1000\Delta^p$$

$$\text{s.t.} \quad P_{53} + \Delta^p \geq P_1$$

$$\Delta^p \geq 0.$$

The objective of the farmer is to maximise the expectation of the first-stage objective:

$$\max \mathbb{E}_{e_1^p, r_1}[V_{1,p_1}(x_1, r_1, e_1^p)], \tag{8.16}$$

where x_1 defines the initial state of the farm, and $p_1 = \$6/\text{kg}$ as defined earlier.

Summary POWDER models the decision process as follows. At the beginning of week t, the farmer measures the five state variables in the model: the soil moisture W_t, the pasture

cover P_t, the quantity of feed in storage Q_t, the number of cows milking C_t, and the quantity of milk solids produced to date M_t. Before choosing an action, the farmer observes the realisation of the three random variables: the potential evapotranspiration e_t^p, the quantity of rainfall r_t, and the forecast milk price p_t. Taking into account the current state and the observation of the random variables, the farmer decides the quantity of irrigation to apply i_t, the quantity of pasture to harvest h_t, the number of cows per hectare to dry-off u_t, and the quantities of grass from pasture f_t^p, grass from storage f_t^q, and palm kernel s_t to feed the herd. As a result of these actions, the system transitions to a new state that will serve as the incoming state in the next week, and the farmer will incur the cost of purchasing palm kernel, harvesting pasture, and applying irrigation. In the 53rd week, the farmer sells the milk produced during the season M_{53} at the end-of-season milk price p_{53}.

8.2 Case study

In this section we investigate the application of POWDER to the farm of the case study in Section 7.3. The results we obtain will be specific to our chosen farm; however, the case study can easily be repeated for different farms.

8.2.1 Calibration

We begin by describing how POWDER was calibrated to the farm. The farm is not irrigated, so i_t was set to 0 for all stages. In addition, the farmer estimated the cost of harvesting c^h at \$275/t, and the metabolisable energy content of grass η^p and palm kernel η^s to be 11 MJ/kg. Furthermore, a maximum lactation length of 44 weeks was set (so that $C_t = 0$ for $t = 45, \ldots, 52$) as the farmer considered lactation lengths longer than this to be infeasible. The minimum quantity of energy for milk production $\underline{\nu}$ was set to 500 MJ/cow/week (0.9 kg/cow/day) based on the value reported by Dairy NZ [46]. We scaled the net energy content of milk η_t^m predicted by the model of Freer et al. [77] by 1.2 so that it matched the value (≈80 MJ/kg) reported by Dairy NZ [46]. The stocking rate $\bar{C}$ was set at 3 cows/ha to match the actual stocking rate on the farm. The cost of palm kernel c^s was set at \$0.5/kg. We validated these parameters by showing the results to the farmer. They confirmed that, given the quantity of feed consumed by the herd, the simulated milk production in the results below was representative of the farm.

Weather data Historical data were obtained from the NIWA CliFlo database [146] for the incident rainfall r_t and evapotranspiration potential e_t^p from 1 January 1997 to 31 December 2016 for the three closest stations to the farm (Wharawhara Water Stn, Tauranga Aero Stn, and Whakamaramara). Then, for each week in the dataset, we averaged the data from all three stations. This resulted in 20 historical readings for each week of the year for both the evapotranspiration potential and incident rainfall. Ribbon plots showing the distributions of rainfall

and evapotranspiration potential by week are given in Figure 8.5.

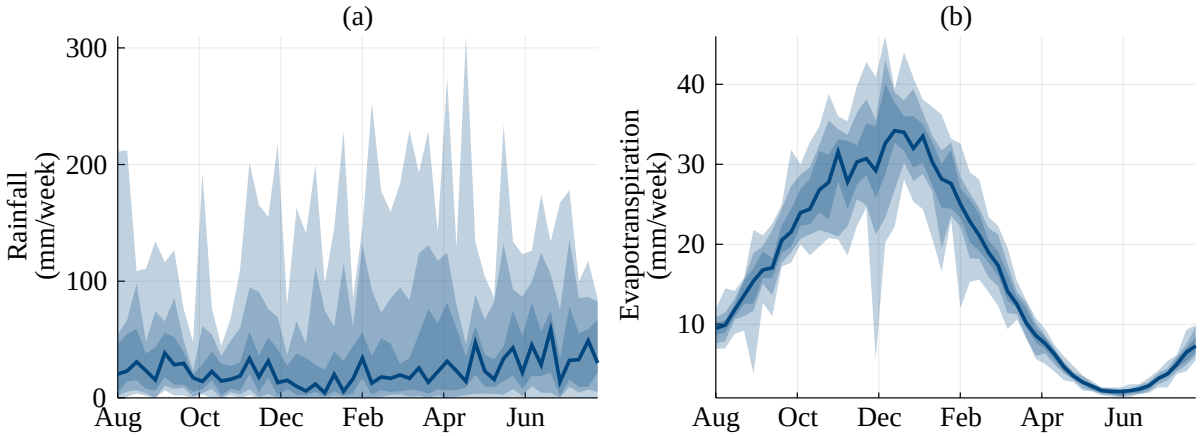


Figure 8.5: Empirical distributions for: (a) incident rainfall r_t; and (b) evapotranspiration potential e_t^p. In order of increasing darkness, shaded bands correspond to the 0-100, 10-90, and 25-75 percentiles. The dark single line represents the 50$^{\text{th}}$ percentile for each week.

We construct a stagewise-independent stochastic process for the weather by empirically sampling with replacement from historical readings that occurred during each week. There are 20 readings in each week. This gives a total of 20^{52} possible different weather scenarios for the stochastic weather process. When coupled with the nine possible price scenarios from the milk price model, there are a total of 9×20^{52} possible scenarios.

Grass model parameters The maximum rate of pasture growth $\Delta P_{\max}$ was assumed to be 65 kg/ha/day, and the maximum pasture cover $P_{\max}$ was assumed to be 3500 kg/ha. These values are regional averages obtained from Dairy NZ [47] and from an interview with the farmer. In addition, the farmer was only able to supply cumulative annual grass growth data. Therefore, it was difficult to calibrate the soil-fertility index κ_t. Instead, we calibrated κ_t by dividing the monthly average pasture growth of a nearby farm (chosen by the farmer from Dairy NZ [47]) with the average weekly potential evapotranspiration readings (e_t^p) for the farm to obtain a weekly estimate for κ_t. These weekly estimates were then scaled so that when κ_t was multiplied by the historical potential evapotranspiration data for the 2013/14 and 2014/15 seasons, the annual pasture growth matched the total grown on the farm during the 2013/14 and 2014/15 seasons. The farmer confirmed that the predicted growth rates were representative of the farm.

8.2.2 Solution method

The POWDER model was implemented in the Julia [28] language using the `SDDP.jl` and JuMP [131] packages discussed in Chapter 3. Gurobi [98] was used to solve the linear subproblems. The model was solved for 1000 SDDP iterations to form an approximately optimal policy using the initial conditions $(W_1, P_1, Q_1, C_1, M_1) = (150, 2500, 0, \bar{C}, 0)$. A plot of the convergence is given in Figure 8.6. The solid line is the upper bound over the iterations. The

semi-transparent circles correspond to the objective of the forward pass of each iteration. The model converged to a first-stage objective value (Eq. 8.16) of $2246/ha. After the model had converged, 1000 Monte Carlo simulations were conducted using the optimal policy and summary statistics were collected for each of these 1000 scenario realisations.

Using a single thread of an Intel i7-4770 CPU with 16GB of memory, the solution time is approximately ten minutes. In total, the solution process involved solving over 2.3 million linear programs. A plot of the solution time of POWDER against the number of iterations is given in Figure 8.7.

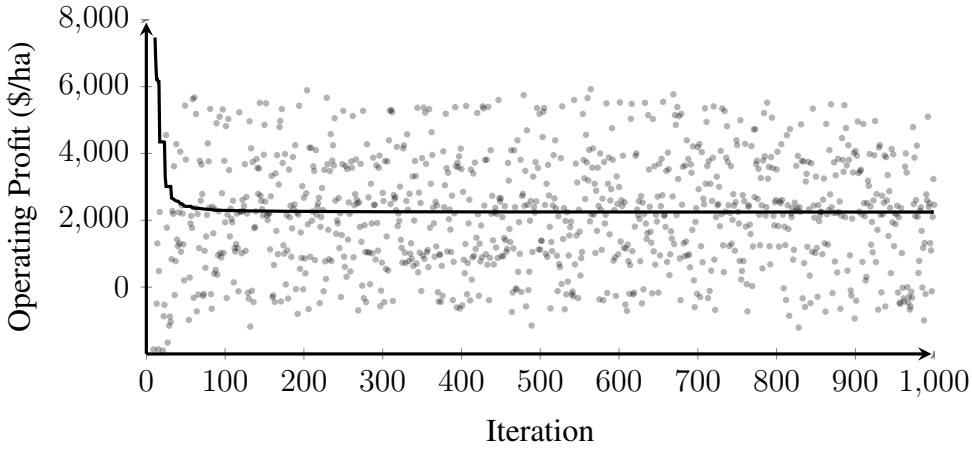


Figure 8.6: Convergence of POWDER against the number of SDDP iterations. Each point represents the objective of one forward simulation of the SDDP algorithm. The solid line plots the evolution of the upper bound over the number of iterations.

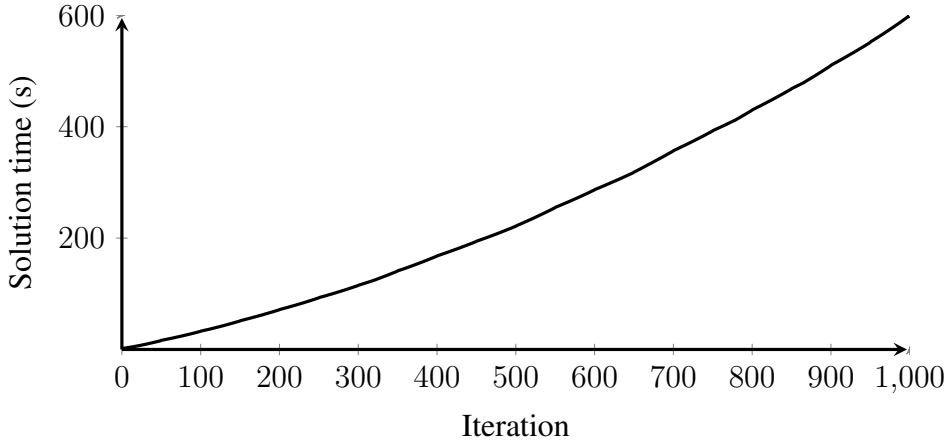


Figure 8.7: Solution time of POWDER against the number of SDDP iterations. Solution time is quadratic in the number of iterations ($r^2 = 1$).

8.2.3 Results

In Table 8.2, we present statistics to compare the simulated results against the historical average for the farm data from the 2013/14 and 2014/15 seasons. In all simulations, POWDER imported

more palm kernel than the farm. This suggests that for the farm, even at a cost of \$500/t for palm kernel (which is larger than the average cost reported in the DairyBase data), feeding a low constant amount of 3 kg/cow/day of palm kernel is profitable. In addition, due to the extra feed imports, the lactation length in almost all simulations was longer than that of the farm (a median of 44 weeks compared with 38.6 weeks). The extra feed and lactation length increased the milk production per hectare from the historical average of 1193 kg/ha to a median value of 1360 kg/ha in the simulations (+14% increase). In turn, this increased operating profit from \$2197/ha to a median value of \$2542/ha. However, in the worst simulated case the operating profit was −\$1608/ha.

	Simulation Percentiles					Historical
	0	25	50	75	100	Avg.
End-of-season Milk Price (\$/kg)	4	5	6	7	8	–
Palm Kernel Cost (\$/t)	500	500	500	500	500	–
Lactation Length (weeks)	31.1	44.0	44.0	44.0	44.0	38.6
Milk Production (kg)						
per Hectare	1006	1302	1360	1407	1511	1193
per Cow	335	434	453	469	504	398
Milk Revenue (\$/ha)	4024	6827	8171	9600	11920	7158
Feed Consumed (t/ha)						
Pasture	9.64	11.83	12.16	12.41	13.14	12.15
Palm Kernel	3.58	3.86	4.32	4.41	4.66	2.85
% Feed Imported	22.1	24.5	26.0	26.8	30.6	19
Palm Kernel Expense (\$/ha)	1790	1929	2162	2203	2330	1425
Fixed Expense (\$/ha)	3536	3536	3536	3536	3536	3536
Operating Profit (\$/ha)	−1608	1243	2542	3873	6157	2197
FEI† Penalty (\$/ha)	96	146	264	297	427	–

Table 8.2: Summary results of 1000 simulations of the SDDP policy. †FEI = Fat Evaluation Index.

In Figure 8.8 and Figure 8.9, we visualise 1000 Monte Carlo weather simulations for two different price scenarios. In Figure 8.8, we plot simulations that sampled the high forecast milk price state during the second half of the year (i.e. $p_{26} = \$7$/kg), whereas in Figure 8.9, we plot simulations that sampled the low forecast milk price state during the second half of the year (i.e. $p_{26} = \$5$/kg).

In each of the subplots, we plot the 0–100 percentiles of the distribution of the plotted variable as a light shaded band. The dark shaded bands correspond to the 10–90$^{\text{th}}$ percentiles. The dotted line corresponds to the 50$^{\text{th}}$ percentile. In addition, we highlight one randomly chosen weather scenario out of the 1000 simulations with a thick, solid line. We chose to highlight the same weather scenario in each plot to show how the weekly farmer decisions vary with different price forecasts.

We plot the first half of each subplot in grey because both figures visualise the same 1000 weather scenarios, and the new price information is not revealed until week 26. Therefore, the

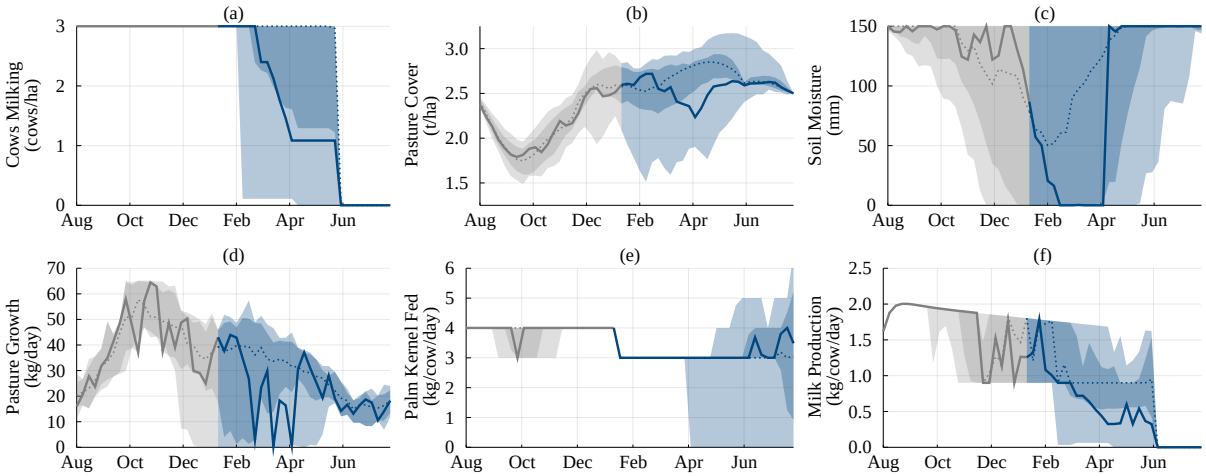


Figure 8.8: Simulated seasons that sample a low forecast milk price during the second half of the season using the optimal policy.

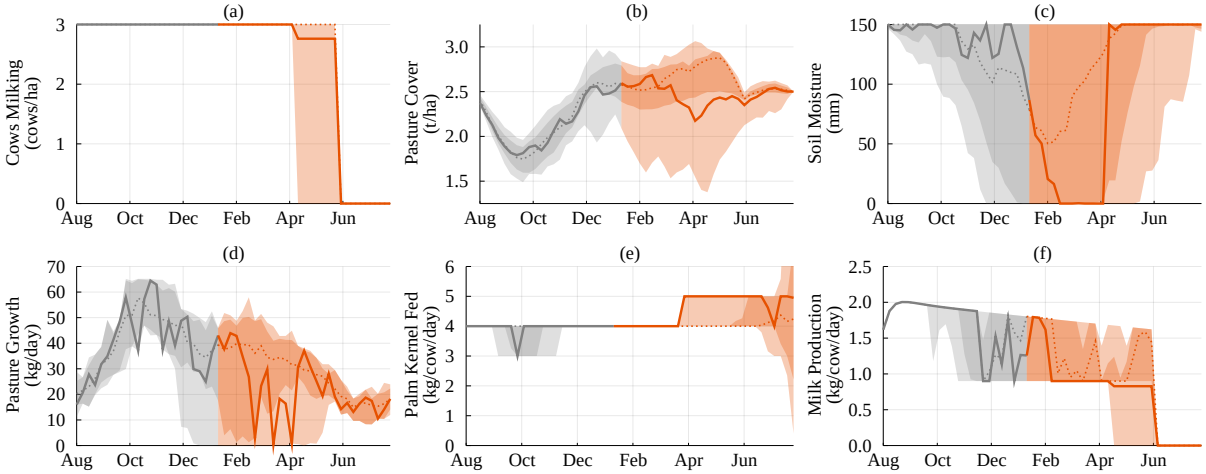


Figure 8.9: Simulated seasons that sample a high forecast milk price during the second half of the season using the optimal policy.

simulations for the first half of the year are identical between Figure 8.8 and Figure 8.9. All the cows are kept milking (Figure 8.8a and Figure 8.9a), and they are fed 4 kg/cow/day of palm kernel (Figure 8.8e and 8.9e). This level of feeding is at the upper end of the acceptable Fat Evaluation Index grade scale. This suggests that the small penalty ($0.25/kg/cow/day) that is incurred is less than the value of the additional milk that is produced. However, differences in rainfall and evapotranspiration between individual simulated scenarios lead to different trajectories for soil moisture (Figure 8.8c and Figure 8.9c) and pasture growth (Figure 8.8d and Figure 8.9d). Despite these differences, the pasture cover follows a clearly observable trend. For the first two months, the rate of grass growth is less than that fed to the cows. This causes the pasture cover to decrease. However, in late September the pasture growth begins to exceed the rate of consumption and the pasture cover increases. Pasture growth peaks in late October before gradually declining over the remainder of the season.

During the second half of the year, the optimal controls diverge depending on the forecast

milk price that is observed. In the high milk price states (Figure 8.9), the optimal management strategy is to keep the entire herd milking for as long as possible (Figure 8.9a) in almost all weather scenarios. This is achieved by increasing the quantity of palm kernel fed to 5 kg/cow/-day in April (Figure 8.9e). This incurs an additional Fat Evaluation Index penalty. However, based on the forecast milk price, the expected value of the milk exceeds the cost of the Fat Evaluation Index penalty.

In contrast, depending on the observed rainfall and evapotranspiration, the trajectories in the low forecast milk price states (Figure 8.8) begin drying off the herd in February. However, if weather conditions are favourable for growing grass, the dry-off decision is delayed. In the trajectory highlighted by the solid blue line, a drought during late February and March (Figure 8.8c) causes the model to begin drying cows off during February (Figure 8.8a). As the drought progressively worsens, more cows are dried-off, and the total pasture cover (Figure 8.8b) declines. By April, the drought breaks, and the pasture cover begins to increase. However, due to the low forecast price, the model does not increase the rate of palm kernel intake (Figure 8.8e) to extend lactation. Instead, the rate of palm kernel intake is increased during June to avoid the penalty arising from the final pasture cover (Figure 8.8b) being lower than the initial pasture cover.

The quantity of milk solids produced per day (Figure 8.8f and Figure 8.9f) tends to switch between the maximum limit and the minimum limit (Eq. 8.11). This bang-bang behaviour is a well-known artefact of linear optimal control problems. This feature can also be observed in Figure 8.8e and Figure 8.9e where, in most cases, the optimal quantity of palm kernel to feed is at one of the break points in the Fat Evaluation Index penalty function (Figure 8.4).

8.2.4 The value of price uncertainty

So far in this chapter, we have modelled the sequence of forecast milk prices by a scenario tree with 9 scenarios. In this section, we analyse the value that the farmer obtains as a result of considering this uncertainty.

To investigate this question, we solve a version of POWDER with the forecast milk price fixed at $p_t = \$6$/kg for weeks $t = 1, 2, \ldots, 52$. In final stage $t = 53$, we realise the same distribution of end-of-season milk prices p_{53} as the original scenario tree. This allows us to quantify the value of learning an updated milk price forecast in week 26 (i.e. when the scenario tree first branches). We call this model the *fixed-price* model. We call the policy obtained from POWDER with the scenario tree based price model the *dynamic-price* model.

After solving the fixed-price model for 1000 SDDP iterations, we conduct a Monte Carlo simulation of the policy with 1000 replications. We also simulate the same 1000 replications of weather and price scenarios through the dynamic-price policy. We provide a density plot of the two distributions of operating profit in Figure 8.10a. Then, for each of those 1000 replications, we calculate the difference between the operating profits of the dynamic- and fixed-price po-

lices. In Figure 8.10b we plot this difference in operating profit against the end-of-season milk price. Positive differences indicate that the dynamic-price model outperformed the fixed-price model. It is also interesting to compare Figure 8.10b to Figure 7.7, which we presented in the previous chapter. Although the exact difference in operating profit is different between the two figures, the same general trend applies.

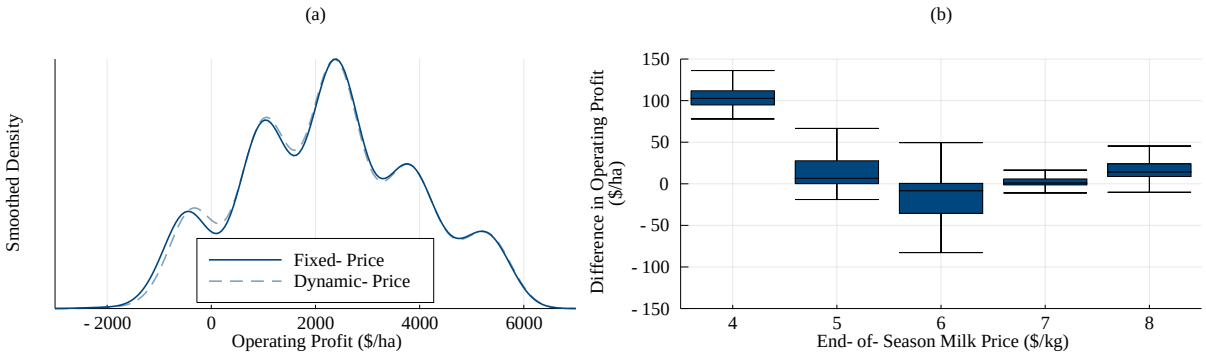


Figure 8.10: Two views of the simulated operating profit from the 1000 Monte Carlo replications of the dynamic- and fixed-price models. (a) Density plot of the operating profit for the dynamic- and fixed-price models. (b) Boxplots of the difference in operating profit between the dynamic- and fixed-price models for 1000 Monte Carlo replications. Positive differences indicate that the dynamic-price model outperformed the fixed-price model.

On average, the dynamic-price model outperformed the fixed-price model by \$11.65/ha (95% confidence interval of $[9.08, 14.22]$). This is small in comparison with the expected operating profit of \$2246/ha. However, when the end-of-season milk price was low (p_{53} =\$4/kg), on average the dynamic-price policy outperformed the fixed-price policy by +\$100/ha. This is because the model is able to dry off earlier and reduce its usage of unprofitable palm kernel. In addition, when the end-of-season milk price was high (p_{53} =\$8/kg), on average the dynamic-price policy outperformed the fixed-price policy by +\$20/ha. This is because the updated price information in week 26 indicates that a higher level of palm kernel feeding is profitable. However, when the end-of-season milk price was p_{53} = \$6/kg, the dynamic-price policy under-performed the fixed-price policy. This is because the dynamic-price policy hedged its decision-making to account for the chance that the end-of-season milk price was not \$6/kg.

In the worst-case, the minimum operating profit increased from −\$2135/ha using the fixed-price policy to −\$1871/ha using the dynamic-price policy (+\$264/ha). Therefore, a policy that is responsive to changes in the forecast milk price is especially attractive to risk-averse farmers.

Although the value of considering price uncertainty is small on average in this example, this may be because there is only a single forecast update during the season. In the next chapter, we model the sequence of forecast milk prices as an auto-regressive process that updates weekly. Since more information can be learned during the season, the value of incorporating price uncertainty is shown to be larger.

8.2.5 A lower stocking rate

The 2015/16 season was characterised by very low international milk prices. As a result, the end-of-season milk price was \$3.90/kg. In order to reduce costs, our farmer reduced their stocking rate from 3 cows/ha to 2.7 cows/ha. To investigate this decision, we re-solved the POWDER model with $\bar{C}$ set to 2.7. All other parameters were kept the same.

In Table 8.3, we present the same set of statistics as Table 8.2. Compared with the case where the stocking rate was 3 cows/ha, the median operating profit per hectare increased from \$2542/ha to \$2846/ha (+12%). This is achieved by increasing per cow production from a median of 453 kg/year to 502 kg/year, despite the quantity of palm kernel imported decreasing from a median of 4.32 t/ha to 3.62 t/ha (-16%) and a similar quantity of pasture being grown (12.16 t/ha compared with 12.15 t/ha). One explanation for this is that by decreasing the stocking rate, the quantity of feed needed to account for fixed costs (such as maintenance and pregnancy) decreases. Therefore, for a similar quantity of feed, more milk can be produced (provided the cows are not producing the biological maximum rate of milk production).

	Simulation Percentiles					Historical
	0	25	50	75	100	Avg.
End-of-season Milk Price (\$/kg)	4	5	6	7	8	–
Palm Kernel Cost (\$/t)	500	500	500	500	500	–
Lactation Length (weeks)	34.6	44.0	44.0	44.0	44.0	38.6
Milk Production (kg)						
per Hectare	1021	1297	1355	1398	1472	1193
per Cow	378	480	502	518	545	398
Milk Revenue (\$/ha)	4103	6798	8128	9556	11702	7158
Feed Consumed (t/ha)						
Pasture	9.64	11.81	12.15	12.39	12.94	12.15
Palm Kernel	2.98	3.23	3.62	3.71	3.98	2.85
% Feed Imported	19.0	21.4	22.8	23.6	28.0	19
Palm Kernel Expense (\$/ha)	1489	1617	1812	1855	1989	1425
Fixed Expense (\$/ha)	3536	3536	3536	3536	3536	3536
Operating Profit (\$/ha)	−1275	1551	2846	4162	6312	2197
FEI† Penalty (\$/ha)	22	72	173	203	349	–

Table 8.3: Summary results of 1000 simulations of the SDDP policy with stocking rate of 2.7 cows/ha. †FEI = Fat Evaluation Index.

In Figure 8.11 we plot the two distributions in operating profit for both stocking rates. The five peaks in the density functions (Figure 8.11a) correspond to the five different milk prices that can be observed. Variations around each peak correspond to the uncertainty associated with the weather. Under every scenario, the simulated operating profit with 2.7 cows/ha was greater than the simulated operating profit with 3 cows/ha (Figure 8.11b), and the expected operating profit (Eq. 8.16) increased from \$2270/ha to \$2646/ha (+17%). Moreover, this average increase was fairly constant across the different end-of-season milk prices.

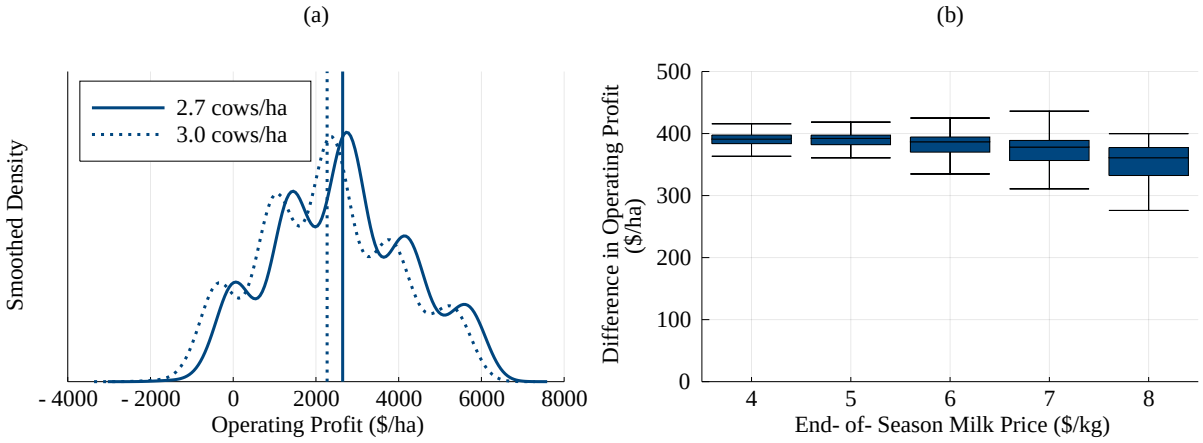


Figure 8.11: (a) Smoothed density of operating profit ($/ha) under differing stocking rates. Vertical lines correspond to the mean of each distribution. (b) Boxplots of the difference in operating profit between the 2.7 cows/ha and 3 cows/ha stocking rate cases. Positive differences indicate that the 2.7 cows/ha model outperformed the 3 cows/ha model.

Reducing the stocking rate has additional benefits for the farmer that the model does not capture. For example, costs that are incurred on a per cow basis (which we amortised into a per hectare figure) such as animal health and young stock management (animals that are too young to produce milk and are grazed off-farm) will decrease. In addition, the environmental impact of the farming operation may decrease as there are fewer animals to produce emissions. Moreover, labour costs will decrease as the time spent performing manual tasks such as milking will decrease. This suggests that by assuming that the fixed cost per hectare was constant between the two configurations, we may have underestimated the financial benefit of reducing the stocking rate.

8.3 Discussion

Our current model only takes ten minutes to solve. This allows scope for a more detailed model. In particular, the stochastic price model used in POWDER is a simplistic approximation for the true price process. For simplicity, we have chosen to ignore recent developments such as the launch of the NZX[8] Milk Price Futures [148], and more detailed milk price forecast models that utilise the twice-monthly GlobalDairyTrade auctions [2]. A more detailed model could be created by extending the scenario tree approach taken in this chapter and allowing the farmer to sell milk within the season rather than just at the end.

One of the largest critiques of this model is the way in which we calculate the net energy requirements of the cow. Readers familiar with cow biology may question the assumption that the cow follows a predetermined body condition trajectory, with any net energy contributing linearly to milk production. In reality, the cow will partition the energy between fat deposition

[8]New Zealand Stock Exchange

and milk production. Attempts to model this partitioning have resulted in non-convex relationships (e.g. [18]). This would create a non-convex Bellman function, precluding the use of SDDP as a solution technique. We justify the approach taken in this model by observing that variations from the typical body condition trajectory typically represent poor health outcomes for the animal [177].

A second critique is that to simplify the model, we have assumed that the cows will consume all of the energy provided by the farmer in the form of grass and palm kernel. This ignores consumption limits, as well as the substitution effect that supplementary feeding has upon pasture intakes [63]. However, when compared with the DairyBase data from the farm used in the case study, POWDER produced realistic results that could be meaningfully interpreted.

A third critique is our approximation of the stochastic processes for rainfall and evapotranspiration potential. In this chapter, we assume that they are stagewise-independent and drawn from historical observations. Future work should be conducted to validate these assumptions and explore alternative processes, such as an auto-regressive process.

Finally, in Chapter 6 we outlined the IDEA (Integrated Dairy Enterprise Analysis) model, which we consider to be the current state-of-the-art for a multistage optimisation model of a dairy farm. Although similar in many respects, IDEA and POWDER answer different questions. IDEA focuses on optimising a detailed model of a farm under deterministic conditions. It models many variables that POWDER does not, including, for example, the impact of stocking rate on pasture utilisation. In contrast, POWDER solves a simplified model of a farm that incorporates stochasticity. This trade-off is necessary in order to maintain tractability of the model. In the future, it would be interesting to incorporate some of the features of IDEA into POWDER, such as a heterogeneous herd. It would also be interesting to simulate IDEA using the dry-off and feeding decisions that arise from POWDER.

8.4 Summary

In this chapter we have described a multistage stochastic optimisation model of a New Zealand dairy farm. In comparison to other more-detailed, but deterministic models, POWDER relies upon a simpler farm-level model to provide insight into farm management practices.

We have shown that optimal management strategies for a farm in the Bay of Plenty region of New Zealand differ based on the combination of economic and weather uncertainties. In particular, the model is able to decide the quantity of palm kernel to feed and when to dry cows off, based on the forecast milk price and current pasture cover. We used the model to analyse the impact of a reduction in stocking rate for the farm. This found that the operating profit improved in every scenario, even if we exclude the additional environmental and economic benefits associated with a reduction in stocking numbers.

Overall, this chapter demonstrates that large multistage stochastic programs in an agricultural context can be solved efficiently to generate meaningful insights for practitioners. In the

next chapter, we extend POWDER to incorporate a more detailed stochastic process for the end-of-season milk price and introduce forward contracting.

Supplementary materials The Julia code to implement POWDER in `SDDP.jl`, as well as the data needed to run the case study and replicate the results in this chapter, is available at `https://github.com/odow/MilkPOWDER`.

Chapter 9

Managing risk on a pastoral dairy farm

In this chapter we extend the POWDER model developed in the previous chapter to incorporate price risk management tools. We demonstrate through simulation that the majority of a farmer's downside risk comes from low-price scenarios instead of drought or poor production. We further show that without price risk management tools, farmers have little ability to manage their exposure to downside risk. However, if farmers have the ability to forward sell their milk, they should forward sell a fixed quantity of milk on the first day of the season, and then operate according to a risk-neutral policy ignoring contracting. Alternative methods of risk management, such as weather derivatives offer some benefit, but this benefit is small due to the large variation in price compared to other factors.

9.1 Introduction

As we mentioned in Chapter 8, New Zealand is responsible for around 28% of international exports of dairy commodities. Therefore, New Zealand farmers are highly exposed volatility in international commodity prices. In Figure 9.1 we plot a historical index of international dairy commodity products since 1986. Before the turn of the millennium, the commodity price was characterised by low volatility about a strong cyclic trend. However, beginning in 2008, the international dairy commodity price has been characterised by high volatility ($>$50%). This coincides with a number of structural changes in the international dairy markets including the launch of GlobalDairyTrade (an auction platform for dairy commodities [87]), increased milk consumption in China (and a number of bilateral of free-trade deals) [144], the removal of quota systems in the European Union [69], and the embargo of milk imports by the Russian Federation [193].

Regardless of the cause, the volatility in international dairy commodity prices has hurt New Zealand dairy farmers. Our analysis of the last 18 years of data from Dairy NZ [49] showed that the yearly operating profit of the average owner-operator (i.e. a farmer who owns the land) had

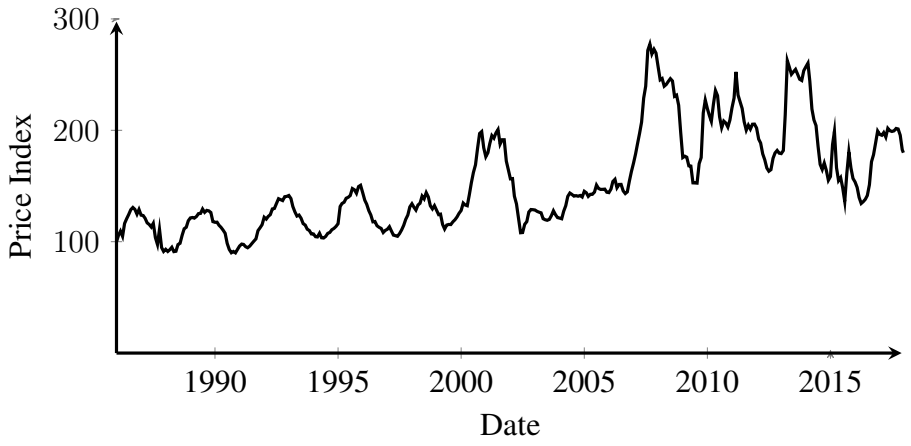


Figure 9.1: Historical price index of international dairy commodities (100 = January 1986). Source: ANZ [5].

a coefficient of variation[1] of 60%. This can lead to severe stress on the financial performance of the farm business, as well as the personal life of the farmer [67, 190]. Managing the two main types of risk (production and price) will reduce the volatility in annual operating profit, resulting in more efficient cash flow management [79], leading to an improvement in farmers' overall well-being [190].

Price-risk management tools such as futures and options contracts are becoming increasingly widespread in global dairy markets as a way of managing price uncertainty [190]. However, the uptake of such tools at the farmer level has been limited, largely due to a lack of education surrounding the instruments and the potential benefits to their business [205, 206]. Therefore, models that can demonstrate the benefit of forward contracting to risk-averse farmers and provide those farmers with an intuitive understanding of when, and how much, they should contract, will contribute towards a greater uptake of price-risk management tools.

In addition to price-risk management tools, various authors have investigated weather derivatives as a mechanism for managing production risk in a dairy context (e.g. [41, 57, 141]). Weather derivatives are financial instruments that can be used to reduce risk associated with adverse or unexpected weather conditions [3]. Ideally, the derivative would settle against the variable that the farmer wishes to manage (e.g. total production). However, this leads to the problems of moral hazard (the farmer may deliberately under-produce in order to claim a payoff) and adverse selection (the farmer is more likely to trade the derivative if their farm is more likely to be affected by drought). Moreover, the compliance cost of assessing damage claims can be large. Index-based derivatives, which settle against a variable reported by a third-party that is correlated with the target variable (e.g. local rainfall reported by governmental weather departments), remove the problems of moral hazard and adverse selection, and lower compliance costs, but introduce basis risk (the reported rainfall will not perfectly match the rainfall on-farm). We can recommend the work of Alexandridis K. and Zapranis [3] as a good introduc-

[1] The standard deviation divided by the mean.

tion to weather derivatives.

One model for solving the pastoral farm management problem is POWDER, which we introduced in the previous chapter. We provide a brief summary for readers who have not yet read that chapter.

Summary of POWDER POWDER decomposes the season into a sequence of 53 weekly blocks (called stages) and links these blocks together by state variables (information that is necessary to communicate the status of the farm between stages). POWDER models the decision process as follows. At the beginning of week $t = 1, 2, \ldots, 52$, the farmer measures the five state variables in the model: the soil moisture W_t, the pasture cover P_t, the quantity of grass in storage Q_t, the number of cows milking C_t, and the quantity of milk solids produced to date M_t. Before choosing an action for the week, the farmer observes the realisation of the three random variables: the potential evapotranspiration[2] e_t^p, the quantity of rainfall r_t, and a forecast for the end-of-season milk price p_t. Taking into account the current state and the observation of the random variables, the farmer decides the quantity of water to apply via irrigation i_t, the quantity of pasture to harvest h_t, the number of cows per hectare to dry-off (i.e. stop milking) u_t, and the quantities of grass from pasture f_t^p, grass from storage f_t^q, and palm kernel s_t to feed the herd. As a result of these actions, the system transitions to a new state that will serve as the incoming state in the next week, and the farmer incurs the cost of purchasing palm kernel, harvesting pasture, and applying irrigation. In the 53rd week, the forecast milk price becomes the end-of-season milk price, and the farmer sells all of the milk produced during the season M_{53} at the end-of-season milk price p_{53}.

As a simple example of the dynamics in the model, consider Q_t, the quantity of grass in storage at the start of week t. This is a state variable with linear dynamics:

$$Q_{t+1} = Q_t + \beta h_t - f_t^q,$$

where h_t is the quantity harvested, and f_t^q is the amount of grass from storage fed to the cows. As a result, the farmer incurs a cost of $c^h \times h_t$ in week t, where c^h is the cost of harvesting one kg of grass (a constant in the model).

POWDER is able to model the effect of weather uncertainty on the management actions of the farmer. However, for price uncertainty, POWDER, uses a scenario tree with nine scenarios, which limits the model to five possible end-of-season milk price outcomes. Moreover, since it assumes a risk-neutral decision maker, POWDER does not consider the ability of the farmer to engage in forward contracting to reduce risk.

In this chapter, we model the forecast for the end-of-season milk price using two variants of a more sophisticated auto-regressive process. This greatly increases the number of possible end-of-season milk price outcomes. In addition, we introduce a forward contract for the

[2] A function of sunlight and temperature that is positively correlated to grass growth.

end-of-season milk price that trades at the conditional expectation, and we study how risk-averse farmers can utilise contracts to manage their downside risk. However, the combination of contracting and an auto-regressive price process leads to a cost-to-go function that is a saddle function. Therefore, we cannot use the SDDP algorithm in the way we we used in the previous chapter. Instead, we use the dynamic interpolation algorithm that we discussed in Chapter 4.

This chapter is laid out as follows. In Section 9.2 we formulate two new models for the sequence of forecasts for the end-of-season milk price in POWDER and explain how we modified the POWDER model to incorporate the price models. Next, we perform two case studies using the farm that we presented in Chapter 7 to study how the farmer utilises the ability to forward sell their milk under different levels of risk aversion: one in Section 9.3 using the first price model, and a second in Section 9.4 using the second price model. Finally, guided by the results of the case studies, we investigate some simple contracting policies in Section 9.5 and explore alternative risk management tools such as weather futures and put options for the end-of-season milk price.

9.2 Formulation

In this section we introduce two new models for modelling the sequence of forecasts for the end-of-season milk price p_{53}, before detailing the modifications we made to POWDER in order to incorporate these price models. The first price model is an extension of the approach taken in the previous chapter and models the sequence of forecast milk prices p_1, p_2, $\ldots$, p_{52} by an auto-regressive process instead of a scenario tree. The second price model approximates the end-of-season milk price as a weighted sum of a series of auction prices over the season. This is similar to the approach taken by AgriHQ [2].

In addition to the price models, we introduce a forward contract for the end-of-season milk price p_{53}. This is assumed to trade at the conditional expectation of the end-of-season milk price in every week (i.e. we assume that there are risk-neutral players in the market). We denote the forward milk price in week t by p_t^f. In any week t, the farmer can sell these contracts at the forward milk price p_t^f, less a transaction cost δ. We call this action of selling contracts *forward selling*. By selling contracts on the forward market, the farmer has *hedged* their position against unfavourable outcomes of the end-of-season milk price. If the farmer forward sells their milk, then they receive p_t^f instead of the end-of-season milk price p_{53} for the sold quantity. In practice, forward selling can occur via a number of ways, including exchange-traded milk price futures, such as those offered by the New Zealand Stock Exchange [148], or through bilateral contracts between the farmer and their processor, such as Fonterra's guaranteed milk price [190]. However, in this thesis, we consider any generic forward market for milk with a forward price p_t^f.

At this point, it is also worth clarifying what we mean by "milk price." We distinguish three types of milk price:

1. There is the end-of-season milk price p_{53}. We shall always refer to this as the *end-of-season* milk price.
2. In each week $t = 1, 2, \ldots, 52$, there is a forecast p_t for the end-of-season milk price, where p_t is the most recent forecast available at the start of week t. We shall always refer to p_t as the *forecast* milk price.
3. In each week $t = 1, 2, \ldots, 52$, milk can be sold at a price p_t^f via the forward market. We shall always refer to p_t^f as the *forward* milk price.

9.2.1 One-state price model

In Chapter 8 the sequence of forecast milk prices is modelled using a scenario tree that branches at two points in time. This is based on the pattern of forecast updates issued by Fonterra, the largest milk processing co-operative in New Zealand. We extend this model by assuming that the sequence of forecast milk prices $p_1, p_2, \ldots, p_{52}$ can be modelled by a mean-reverting auto-regressive process in which the variance of the error term ϕ_t^p diminishes as the end of the season approaches:

$$p_t = \alpha^p p_{t-1} + \beta^p + \phi_t^p, \tag{9.1}$$

where α^p and β^p are constants in the model, and ϕ_t^p is a finite-discrete random variable with mean zero. Furthermore, we assume that the forward milk price in stage t, p_t^f, trades at the conditional expectation of the end-of-season milk price:

$$p_t^f = \mathbb{E}\left[p_{53} \;\middle|\; p_t\right], \tag{9.2}$$

which will typically be different from the unconditional expectation. This assumes that there are risk-neutral speculators in the market who remove any arbitrage opportunity.

In this *one-state* price model, there is a single price-state p_t, the current forecast milk price. In the last stage $t = 53$, the forecast milk price becomes the end-of-season milk price p_{53}.

9.2.2 Two-state price model

In the one-state price model, we assume that the sequence of forecast milk prices can be modelled by an auto-regressive process. However, the end-of-season milk price paid by Fonterra is actually set by a process regulated by the New Zealand Commerce Commission [76].

As an approximation of this process, the end-of-season milk price can be modelled as the average price of five so called *reference commodity products* traded on the GlobalDairyTrade auction platform over the season.[3] The five reference commodity products are: whole milk powder, skim milk powder, anhydrous milk fat, butter, and butter milk powder. The auction prices are averaged by Fonterra to give the end-of-season milk price based on the relative quantities

[3]GlobalDairyTrade (GDT) is an international marketplace for dairy commodity products [186].

of each reference commodity product produced by Fonterra (which we term the *product mix*), as well as the fraction of the annual production sold in each auction over the season (which we term the *sales curve*). In addition, the prices have to be converted from US dollars to NZ dollars and an allowance made for the cost of processing raw milk into the five reference commodity products.

In this improved model, which we term the *two-state* price model, we assume that the product-mix weighted average GlobalDairyTrade price in stage t g_t (henceforth *spot price*), follows an auto-regressive process with lag one:

$$g_t = \begin{cases} \alpha^g g_{t-1} + \beta^g + \phi_t^g, & \text{if an auction is held in week } t \\ g_{t-1}, & \text{otherwise,} \end{cases} \tag{9.3}$$

where α^g and β^g are constants in the model, and ϕ_t^g is a finite-discrete random variable. Note that these are different constants to those in Eq. 9.1. Also, note that this expression accounts for the twice-monthly pattern of GlobalDairyTrade auctions.

We denote the fraction of total production sold by Fonterra during week t (the *sales curve*) by w_t. There is a requirement that $\sum_{t=1}^{52} w_t = 1$. Note that this is Fonterra's sales curve. Therefore, it is independent of the farmer (i.e. we assume that the farmer is a price-taker in the market). w_t is a constant in the model.

Over the season, Fonterra collects raw milk, processes it into other products, and sells it in the GlobalDairyTrade auctions. Therefore, they slowly accumulate sales revenue. We denote the value of accumulated sales revenue per kilogram in stage t by a_t. Since the sales curve w_t is known, and the farmer has observed the realisation of the GlobalDairyTrade spot price $g_1, g_2, \ldots, g_t$, the value of accumulated sales revenue per kilogram can be calculated as:

$$a_t = \sum_{i=1}^{t} w_i g_i. \tag{9.4}$$

It follows that $a_t = a_{t-1} + w_t g_t$. Since both the GlobalDairyTrade spot price g_t and the sales curve w_t are non-negative, the accumulated sales revenue per kilogram of milk solids a_t is monotonically increasing with t. Moreover, the sales curve w_t is independent of the farmer's production. Therefore, in week t, the farmer is guaranteed a minimum price of a_t for each kilogram of milk produced at any time during the season.

Finally, we assume that the forward milk price p_t^f equals the value of accumulated sales revenue a_t, plus the expected future value:

$$p_t^f = \mathbb{E}\left[p_{52} \;\middle|\; a_t, g_t \right] = a_t + \mathbb{E}\left[\sum_{i=t+1}^{52} w_i g_i \;\middle|\; g_t \right]. \tag{9.5}$$

As before, this assumes that there are risk-neutral speculators in the forward market who remove arbitrage opportunities.

In this *two-state* price model there are two price-states: the value of accumulated sales a_t, and the current spot price g_t. In the last stage $t = 53$, the value of accumulated sales revenue over the season a_{52} becomes the end-of-season milk price p_{53}.

9.2.3 Modifications to POWDER

To incorporate the two state-dependent price models into POWDER, we define a control variable $m_t^f \geq 0$ to be the kilograms of milk sold at the current forward milk price p_t^f in stage t. Next, we modify Eq. 8.13 from Chapter 8 so that:

$$M_{t+1} = M_t + m_t/\eta_t^m - m_t^f, \tag{9.6}$$

where M_t is the quantity of milk that has been produced, but not sold on the forward market, at the start of week t.

In addition, for stages $t \leq 52$ we modify the value-to-go function V_t, so that for the one-state price model we have:

$$V_t(x_t, p_t, e_t^p, r_t, \phi_t^p) = \max \quad \underbrace{\mathbb{F}_{e_{t+1}^p,\, r_{t+1},\, \phi_{t+1}^p}\left[V_{t+1}(x_{t+1}, p_{t+1}, e_{t+1}^p, r_{t+1}, \phi_{t+1}^p)\right]}_{\text{future profit}} -$$
$$\ldots \underbrace{\left(c^s s_t + c^h h_t + c^i i_t\right)}_{\text{week } t \text{ cost}} + \underbrace{(p_t^f - \delta) \times m_t^f}_{\text{forward sales}}$$

$$\text{s.t.} \quad \text{the transition function in Chapter 8}$$
$$Eq.\ (9.1),\ (9.2),\ (9.6) \ \text{ above,}$$

and for the two-state price model we have:

$$V_t(x_t, g_t, a_t, e_t^p, r_t, \phi_t^g) = \max \quad \underbrace{\mathbb{F}_{e_{t+1}^p,\, r_{t+1},\, \phi_{t+1}^g}\left[V_{t+1}(x_{t+1}, g_{t+1}, a_{t+1}, e_{t+1}^p, r_{t+1}, \phi_{t+1}^g)\right]}_{\text{future profit}} -$$
$$\ldots \underbrace{\left(c^s s_t + c^h h_t + c^i i_t\right)}_{\text{week } t \text{ cost}} + \underbrace{(p_t^f - \delta) \times m_t^f}_{\text{forward sales}}$$

$$\text{s.t.} \quad \text{the transition function in Chapter 8}$$
$$Eq.\ (9.3) \text{ – } (9.6) \ \text{ above.}$$

In both cases, δ is a constant transaction cost (\$/kg) and $\mathbb{F}$ is a coherent risk measure as discussed in Section 1.3. In the final stage $t = 53$, the farmer receives the end-of-season price p_{53} for each kilogram of unsold milk M_{53} from their processing company. This is identical to the formulation in Chapter 8.

Note that the value function V_t in stages $t = 1, 2, \ldots, 52$ is a saddle function (i.e. concave with respect to x_t and convex with respect to the price states p_t, a_t, and g_t) due to the bilinear

term involving the forward milk price p_t^f and the quantity m_t^f of milk sold on the forward market. In addition, the value function of the final stage V_{53} is a saddle function due to the bilinear term involving the end-of-season milk price p_{53} and the quantity of unsold milk M_{53}. These bilinear terms preclude the use of the traditional SDDP algorithm. In the previous chapter, we overcame this problem by using a Markovian policy graph. However, our problem can be solved without discretisation using the algorithm we outlined in Chapter 4. We call the version of POWDER with contracting and the price processes outlined in this section POWDER-II.

9.3 Case study I

In this section we investigate the application of POWDER-II with the one-state price model to our now familiar farm. Apart from the price process, we use the data from Section 8.2 for all parameters. The transaction cost δ was set at \$0.015/kg (based on current brokerage rates of firms actively trading in the market). We now detail how we calibrated the price process.

9.3.1 Calibration

Historical sequences of forecast milk prices were obtained from Fonterra [75] for the eight seasons from 2009/10 to 2016/17 inclusive. These are plotted in Figure 9.2a. The one-state price model is difficult to calibrate since there are only eight complete seasons of data,[4] and the opening milk price forecasts (i.e. p_1) are not identical. However, for each week in the eight seasons, we calculated the additive change in the forecast milk price compared to the previous week. This resulted in eight values of possible changes (i.e. one for each historical season) for the forecast milk price in each week $t = 1, 2, \ldots, 52$. Then, we calculated a final change between the forecast milk price at the end of week 52 and the end-of-season milk price p_{53} in each season. Finally, we constructed a stochastic process for the forecast milk price by assuming that p_1=\$6/kg, and that the noise term ϕ_t^p is sampled with replacement from the empirical distribution of forecast milk price changes in week t. Therefore, we calibrate Eq. 9.1 as:

$$p_{t+1} = p_t + \phi_t^p. \tag{9.7}$$

We refer to a single realisation of prices $p_1, p_2, \ldots, p_{53}$ as a *price scenario*. In Figure 9.2b we plot 100 price scenarios for illustrative purposes. In addition, one of the 100 scenarios is highlighted as a thick, dark line to demonstrate the dynamics of the process.

Since the value of ϕ_t^p is sampled independently between stages, and accounting for duplicate observations, there are approximately 1.5×10^8 different price scenarios.

Weather uncertainty In this case study, we use the same stagewise-independent stochastic process for the weather as given in Chapter 8. We refer to a single realisation of evapotranspira-

[4] Prior to this, Fonterra used a different mechanism for calculating the milk price.

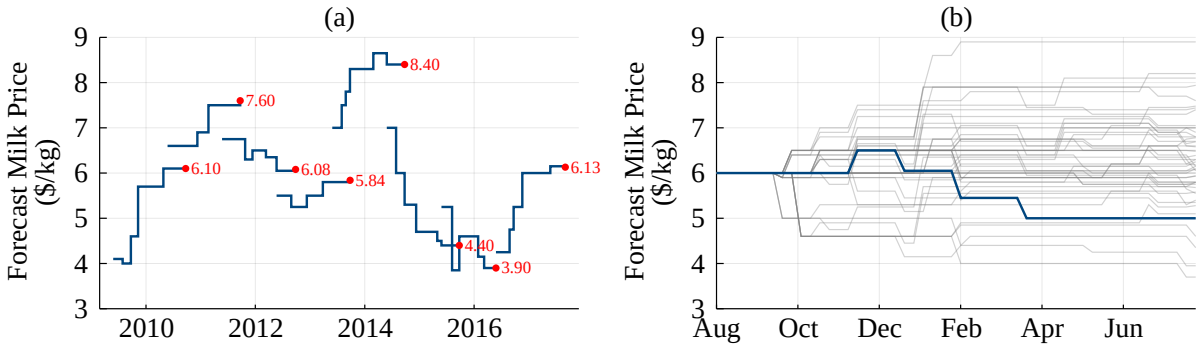


Figure 9.2: (a) Historical sequences of forecast milk prices issued by Fonterra. Annotated points are actual end-of-season prices. (b) Simulated realisations of the sequence of forecast milk prices for one season. Thick, dark line is a single price scenario highlighted for effect.

tion and rainfall outcomes $(e_1^p, r_1), (e_2^p, r_2), \ldots, (e_{52}^p, r_{52})$ as a *weather scenario*. Furthermore, we sample the weather and price scenarios independently. Therefore, including the 20^{52} weather scenarios from Chapter 8, there are approximately 10^{75} weather and price scenarios.

Risk To model the farmer's risk-appetite we use coherent risk measures [6]. In particular, we use a convex combination of the Expectation ($\mathbb{E}$) and Average Value-at-Risk (AV@R) measures proposed by Shapiro et al. [189]:

$$\mathbb{F}[\mathbf{X}] = \lambda\mathbb{E}[\mathbf{X}] + (1-\lambda)\text{AV@R}_\beta[\mathbf{X}], \tag{9.8}$$

where $\text{AV@R}_\beta[\mathbf{X}]$ is the expectation of $\mathbf{X}$ below the β quantile and $\lambda \in [0, 1]$.

In this case study, we set $\beta = 0.25$ and vary λ to represent different levels of risk-aversion. These values were arbitrarily chosen to provide illustrative solutions instead of accurately reflecting the risk appetite of the farmer. We consider the following six cases:

1. risk-neutral with contracting ($\lambda = 1$);
2. risk-neutral without contracting ($\lambda = 1$ and $m_t^f = 0$);
3. medium-risk-averse with contracting ($\lambda = 0.5$);
4. medium-risk-averse without contracting ($\lambda = 0.5$ and $m_t^f = 0$);
5. low-risk-averse with contracting ($\lambda = 0.75$); and
6. high-risk-averse with contracting ($\lambda = 0.25$).

9.3.2 Solution method

Each of the six instances of the POWDER-II model using the one-state price model described above were solved for 20,000 iterations using the dynamic interpolation algorithm that we discussed in Chapter 4 and the `SDDP.jl` library that we discussed in Chapter 3. The initial conditions were $(W_1, P_1, Q_1, C_1, M_1, p_1) = (150, 2500, 0, 3, 0, 6)$. We used a virtual-machine with 16 Intel Xeon E5-2698 CPUs and 64GB of memory. Solution times for 20,000 iterations

were on the order of 16 hours. In total, the solution process for each model involved solving over 200 million linear programs.

Since we sample the weather and price scenarios independently, the solution time is much greater than that reported in Chapter 8 ($\approx$ 10min). Moreover, due to the convergence issues discussed in Section 4.4.4, many more iterations of the algorithm are needed, further increasing the computational effort. Compounding this, the lack of cut selection (discussed in Section 4.4.4) causes the solve time per iteration to increase as the number of cuts added to each subproblem increases.

Plots of the convergence are given in Figure 9.3 for the risk-neutral case and Figure 9.4 for the medium-risk-averse case with contracting. The solid line is the upper bound over the iterations. The semi-transparent circles correspond to the objective of the forward pass on each iteration. Notice that we conduct many more iterations than appears to be necessary by visual inspection. The reason for this is as follows. Initially, we only conducted 1000 iterations (the same as in Chapter 8). However, when simulating the policy, sub-optimal decision-making was observed in a percentage of replications ($<$10%). Observed sub-optimal behaviour included drying cows off in the first half of the year and finishing the year with less than 2,500 kg/ha of pasture cover. We repeated the visualisation test after 10,000 iterations and observed similar behaviour. Therefore, we chose 20,000 iterations. These results highlight the problems with standard stopping rules for SDDP, which we discussed in more detail in Section 2.3. In addition, note that in the risk-averse case (Figure 9.4), the upper bound converges to a value that is less than the objective of the majority of most forward simulations. This is because we are plotting the end-of-horizon outcome of each forward pass, rather than the nested outcome that is our true objective.

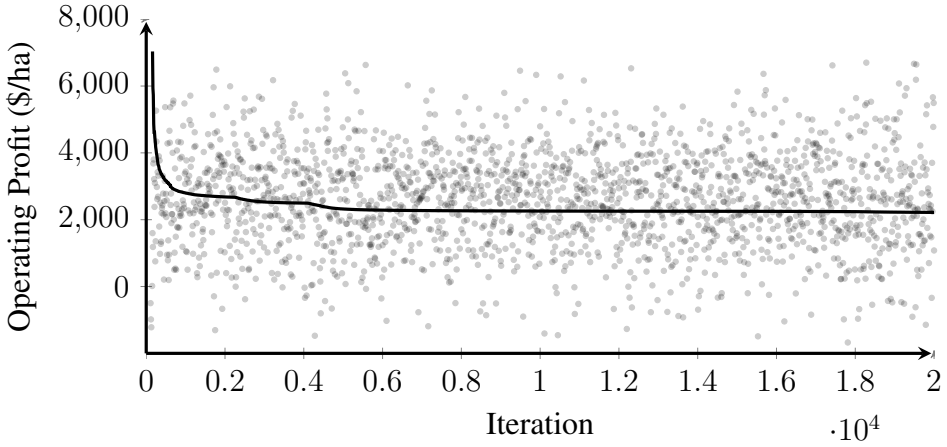


Figure 9.3: Convergence of risk-neutral POWDER-II. Each point represents the objective of one forward simulation of the SDDP algorithm. The solid line plots the evolution of the upper bound over the number of iterations.

After each model had converged, a Monte Carlo simulation of the optimal policy was conducted with 2000 replications. The same 2000 weather and price scenarios were used for each

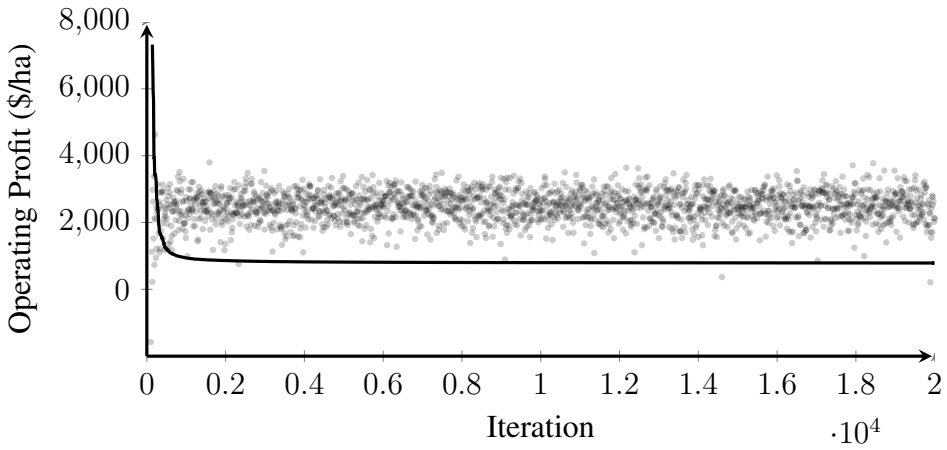


Figure 9.4: Convergence of medium-risk-averse POWDER-II with contracting. Each point represents the objective of one forward simulation of the SDDP algorithm. The solid line plots the evolution of the upper bound over the number of iterations.

simulation. The distribution of these weather and price scenarios over time is shown in Figure 9.5. In each of the subplots within each figure, we plot as shaded bands in order of increasing darkness, the 1–99 and 10–90$^{\text{th}}$ percentiles of the distribution of the plotted variable. The solid line corresponds to the 50$^{\text{th}}$ percentile.

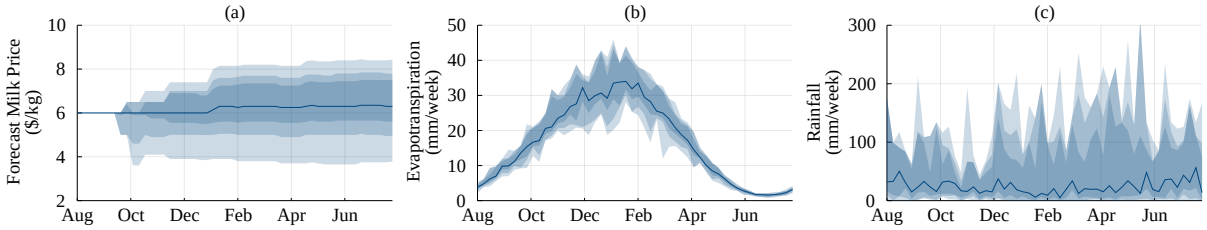


Figure 9.5: Distribution of the random variables over time: (a) the milk price forecast p_t; (b) the potential evapotranspiration e_t^p; and (c) the rainfall r_t.

9.3.3 Results

In Table 9.1 we present the same set of statistics to analyse the simulated results of the risk-neutral policy as those presented in Chapter 8. In each row of the table we give the median value of the 2000 replications of the optimal policy.[5] Note that the values in each row are calculated independently, so the values in a column of the table may not correspond to the same simulation.

In the risk-neutral case the optimal contracting policy is to not sell forward, since any contracting incurs a transaction cost. Therefore, the risk-neutral solution with contracting is the same as the risk-neutral solution without contracting and we omit it from future discussion.

[5] An exception is the last two rows which give the mean and standard deviation of the operating profit. (The mean is the objective of the risk-neutral optimisation.)

Risk Aversion	Neutral	Medium	Low	Medium	High
Contracting	No	No	Yes	Yes	Yes
End-of-season Milk Price ($/kg)	6.25	6.25	6.25	6.25	6.25
Palm Kernel Cost ($/t)	500	500	500	500	500
Lactation Length (weeks)	44.0	44.0	44.0	44.0	40.3
Milk Production (kg)					
per Hectare	1360	1269	1376	1341	1247
per Cow	453	423	471	447	434
Milk Revenue ($/ha)	8580	8000	8546	8078	7842
Feed Consumed (t/ha)					
Pasture	12.17	12.04	12.15	12.02	11.44
Palm Kernel	4.31	3.71	4.29	4.23	4.14
% Feed Imported	26.0	23.5	26.1	26.0	26.4
Palm Kernel Expense ($/ha)	2153	1854	2147	2114	2070
Fixed Expense ($/ha)	3536	3536	3536	3536	3536
Operating Profit ($/ha)					
Median	2640	2491	2638	2518	2015
Mean	2604	2457	2586	2488	2000
Standard Deviation	1412	1342	421	436	490

Table 9.1: Summary of Monte Carlo simulations using the one-state price model.

Compared with the risk-neutral case, total pasture consumption in the medium-risk-averse case without contracting dropped from 12.17 t/ha to 12.04 t/ha, and palm kernel consumption dropped from 4.31 t/ha to 3.71 t/ha. This led to a decrease in milk production from 453 kg/cow to 423 kg/cow. The median operating profit dropped from $2640/ha to $2491/ha. However, when contracting was allowed in the medium-risk-averse case, pasture consumption was similar to the medium-risk-averse case without contracting (12.02 t/ha compared with 12.04 t/ha), but palm kernel consumption increased from 3.71 t/ha to 4.23 t/ha (close to the risk-neutral quantity of 4.31 t/ha). In addition, median operating profit rose from $2491/ha to $2518/ha. This suggests that forward selling milk gives the farmer certainty that feeding palm kernel is profitable.

As the risk-aversion level increases in the risk-averse cases with contracting (i.e. more weight is placed on the AV@R term in Eq. 9.8), the following behaviour is observed. In all cases, the median operating profit is higher than the mean (i.e. the distribution is left-skewed). However, as the level of risk-aversion increases, the mean converges towards the median. This shows how a risk-averse policy shifts the tail of the distribution upwards at the expense of lowering the mean. In addition, the lactation length, milk production, and feed consumption all decrease.

Notably, the low-risk-averse case with contracting has a similar median operating profit to the risk-neutral case ($2640/ha in the risk-neutral case compared with $2638/ha in the low-risk-averse case). However, as expected, the risk-neutral case has a higher mean ($2604/ha in the risk-neutral case compared with $2586/ha in the low-risk-averse case).

Distributions of operating profit In Figure 9.6 we provide boxplots of the operating profit for each of the five cases. The light coloured boxes represent cases where contracting was not allowed, whereas the dark coloured boxes represent cases where contracting was allowed. We see that there is very little difference between the distribution of the risk-neutral case and the medium-risk-averse case without contracting. This suggests that even if the farmer is risk-averse, they have little ability to manage their downside risk through on-farm management actions such as drying-off their herd early. In contrast, there is a clear difference between the distribution operating profit for policies with contracting and policies without contracting. The medium-risk-averse policy with contracting performs similarly to the medium-risk-averse policy without contracting on average (mean of \$2488/ha compared with \$2457/ha), but with a much smaller variance in operating profit (standard deviation of \$436/ha compared with \$1342/ha).

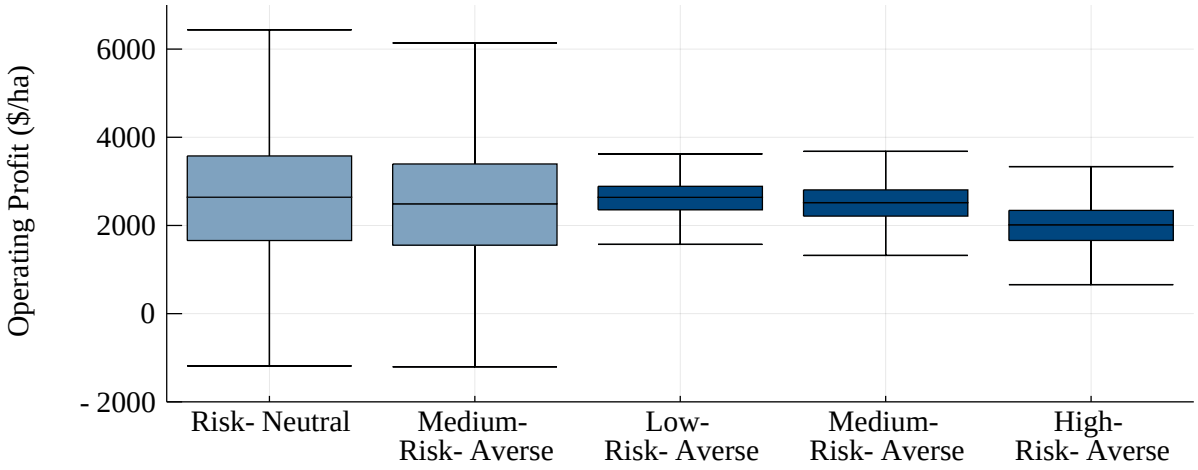


Figure 9.6: Boxplots comparing the distributions of simulated operating profit under different policies using the one-state price model. Light coloured boxes are without contracting. Dark coloured boxes are with contracting.

Counter intuitively, as we increase the level of risk aversion, the variance of the distribution increases, the mean decreases, and the lower tail fattens. This is a result of our use of a nested risk measure. We delay our discussion of this phenomenon until the end of this section.

It is instructive to compare the variation in annual production (in this case, from the risk-neutral policy) with the variation in the end-of-season milk price (Figure 9.7). Milk production is typically within $\pm 5\%$ of the mean but can reach $\pm 20\%$ if growing conditions are good or the farmer is forced to dry-off early. In contrast, the end-of-season milk price can vary by up to $\pm 50\%$ (i.e. between \$3/kg and \$9/kg, given a mean of \$6/kg). This demonstrates that the majority of the variance of the unhedged distributions for operating profit in Figure 9.6 comes from variation in the end-of-season milk price, as opposed to variation in milk production.

To explain the differences in the distribution in operating profit for each of the cases, we present a number of plots visualising the 2000 replications of the Monte Carlo simulations of the policies. In each of the subplots within each figure we plot, as shaded bands in order of

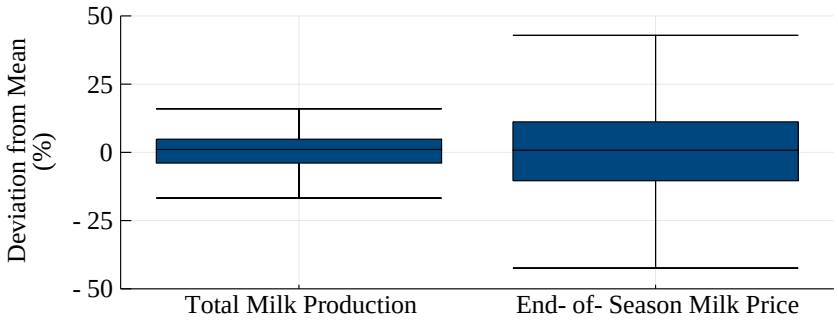


Figure 9.7: Boxplots comparing the distributions of total milk production (kg/ha) (left) and end-of-season milk price ($/kg) (right), as measured by the percentage deviation from the mean of each distribution.

increasing darkness, the 1–99 and 10–90$^{\text{th}}$ percentiles of the distribution of the plotted variable. The solid line corresponds to the 50$^{\text{th}}$ percentile.

Risk-neutral policy In Figure 9.8 we visualise the optimal risk-neutral policy. We see very similar results to those of Chapter 8. (The results are not identical since there is a different distribution for the end-of-season milk price.) In general, most of the herd is kept milking (Figure 9.8a), the pasture cover follows a seasonal trajectory (Figure 9.8b), and each cow is fed 3–5 kg of palm kernel per day (Figure 9.8c), depending on the forecast milk price (Figure 9.5a) and time of year. Over the course of the season, the total quantity of unsold milk accumulates (Figure 9.8d) since there is no forward contracting (Figure 9.8e). Therefore, the farmer sells the entire quantity of milk produced over the season at the end-of-season milk price p_{53}.

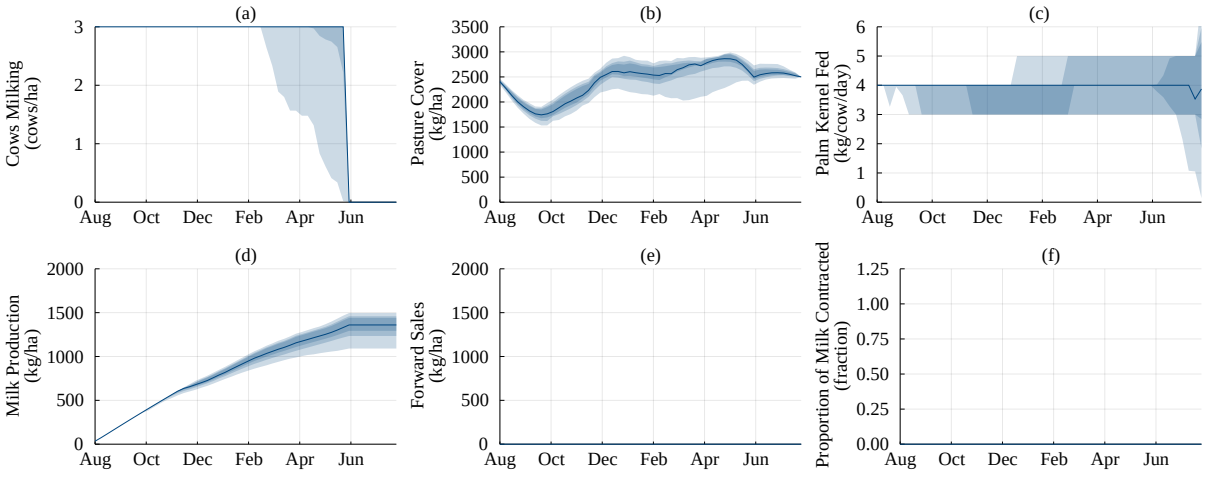


Figure 9.8: Monte Carlo simulation using the risk-neutral policy.

Medium-risk-averse policy: no contracting In Figure 9.9 we visualise the optimal medium-risk-averse policy when contracting is not allowed. Compared with the risk-neutral case (Figure 9.8), the policy dries off more cows earlier (Figure 9.9a) to maintain a greater pasture

cover (Figure 9.9b). This avoids the penalty associated with ending the season with less than 2500 kg/ha of pasture cover (Eq. 8.15). In addition, due to the lower stocking rate, less palm kernel is fed to the cows (Figure 9.9c). Due to the earlier dry-off and lower rates of palm kernel consumption, less milk is produced (Figure 9.9d). Since we explicitly forbid contracting (Figure 9.9e), the farmer sells all the milk produced over the season at the end-of-season milk price p_{52}.

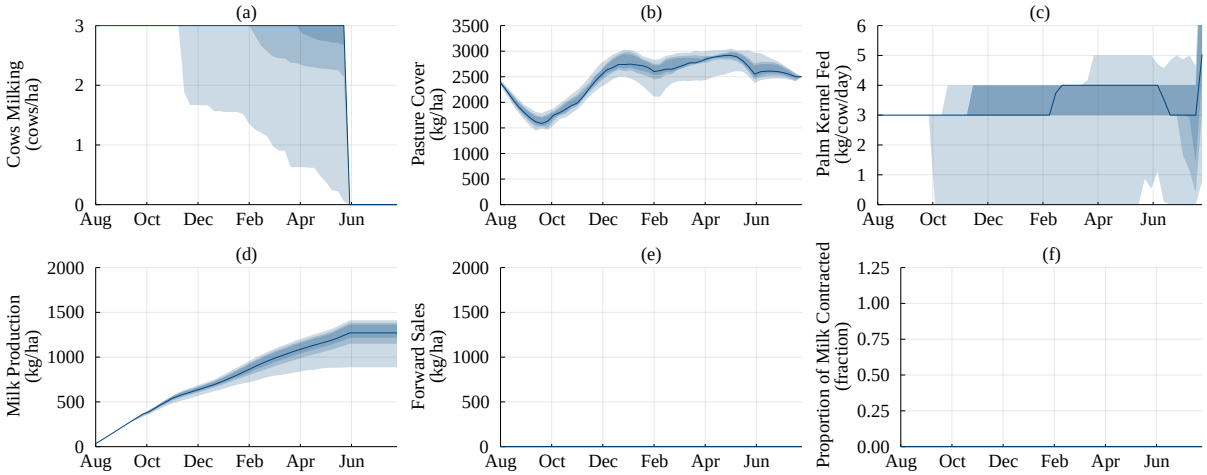


Figure 9.9: Monte Carlo simulation using the medium-risk-averse policy without contracting.

Low-risk-averse policy: with contracting In Figure 9.10 we visualise the low-risk-averse policy when contracting is allowed. The resulting policy is similar to the risk-neutral case (Figure 9.8). The herd is dried-off at approximately the same time (Figure 9.10a) and a similar level of pasture cover is maintained (Figure 9.10b). However, more palm kernel is fed during the last six months of the season (Figure 9.10c).

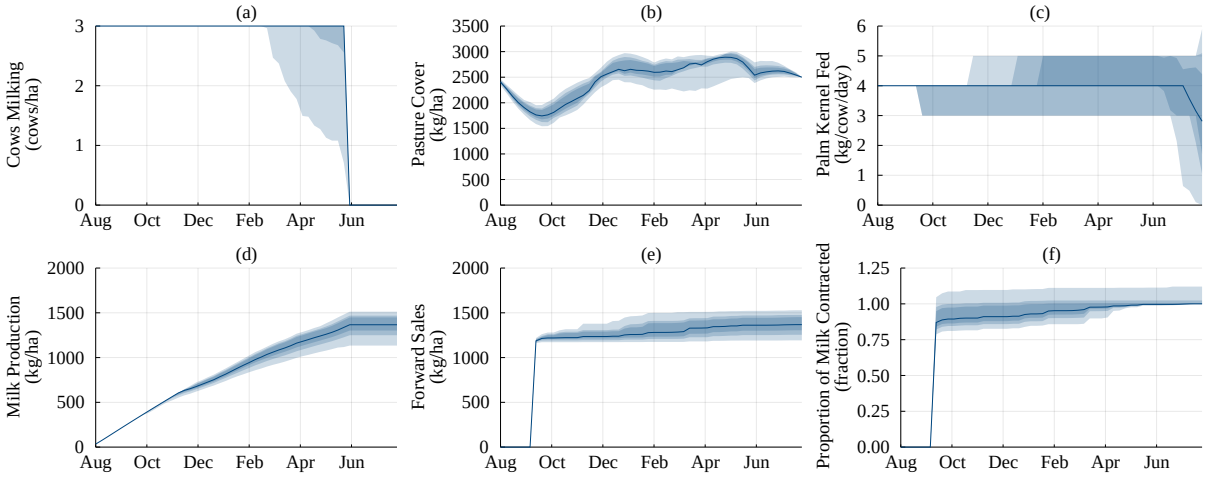


Figure 9.10: Monte Carlo simulation using the low-risk-averse policy with contracting.

In week seven, approximately 1200 kg of milk is sold on the forward market in all scenarios (Figure 9.10e). This equates to roughly 80% of the total average seasonal production (Figure

9.10f). Week seven is the last week in which the forecast milk price remains unchanging at \$6/kg (Figure 9.5a). (In week eight there is a chance that the forecast milk price will be downgraded.) This suggests that the same contracting decision could have been made at any point in the first seven weeks for an equivalent result. Over the remainder of the season, the farmer gradually forward sells their milk (via the forward contracts) in small amounts (Figure 9.10e) as the milk is produced (Figure 9.10d). This attempts to avoid the situation where the farmer sells more forward contracts than they actually produce. When this occurs, the farmer increases their exposure to price risk by engaging in the forward market, rather than reducing it. In the Monte Carlo simulation, over-contracting occurs in approximately 10% of seasons (Figure 9.10f). This coincides with a season in which low pasture growth forces the farmer to dry-off their herd early, thereby reducing the quantity of milk they can produce below their contracted level.

Medium-risk-averse policy: with contracting In Figure 9.11 we visualise the medium-risk-averse policy when contracting is allowed. The resulting policy is similar to the medium-risk-averse policy without contracting (Figure 9.9). The herd is dried-off at approximately the same time (Figure 9.11a), and a similar level of pasture cover is maintained (Figure 9.11b). However, more palm kernel is fed during the first six months of the season (Figure 9.11c).

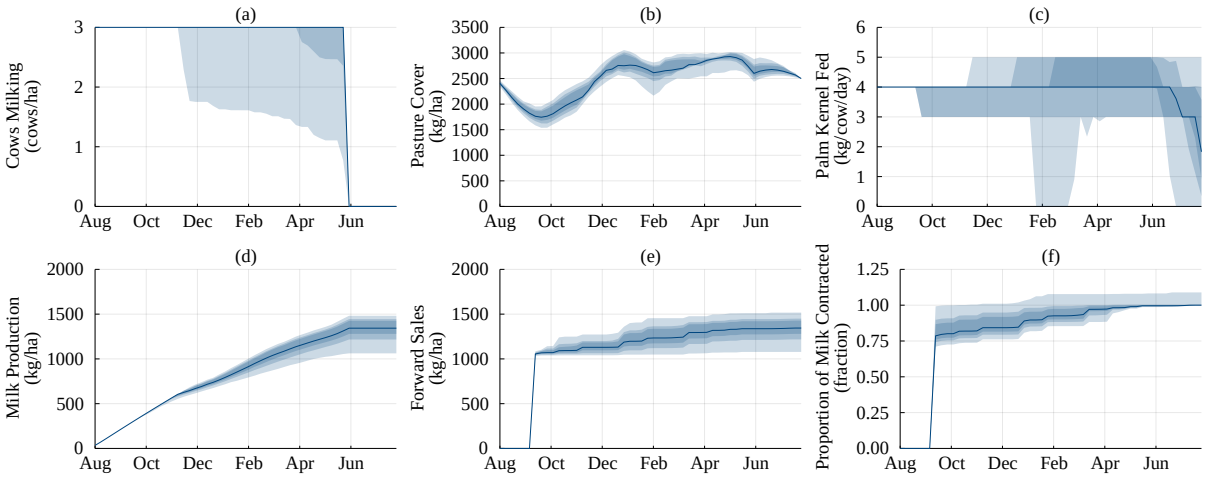


Figure 9.11: Monte Carlo simulation using the medium-risk-averse policy with contracting.

In week seven, approximately 1000 kg of milk is sold on the forward market in all scenarios (Figure 9.11e). This equates to roughly 75% of the total average seasonal production (Figure 9.11f). Following this initial decision, the remainder of the seasons contracting decisions are similar to the low-risk-averse policy (Figure 9.10). The lower initial quantity (1000 kg compared with 1200 kg in the low-risk-averse policy) could be due to a number of factors. First, we already observed in Figure 9.6 that the use of nested risk measures can lead to counter-intuitive results. Second, the model may contract less due to the risk of over-contracting.

High-risk-averse policy: with contracting In Figure 9.12 we visualise the high-risk-averse policy when contracting is allowed. Compared with the medium-risk-averse case with contracting (Figure 9.11), the herd is dried-off earlier (Figure 9.12a), but a similar level of palm kernel is fed (Figure 9.12c). This leads to a slightly higher level of pasture cover over the season (Figure 9.12b).

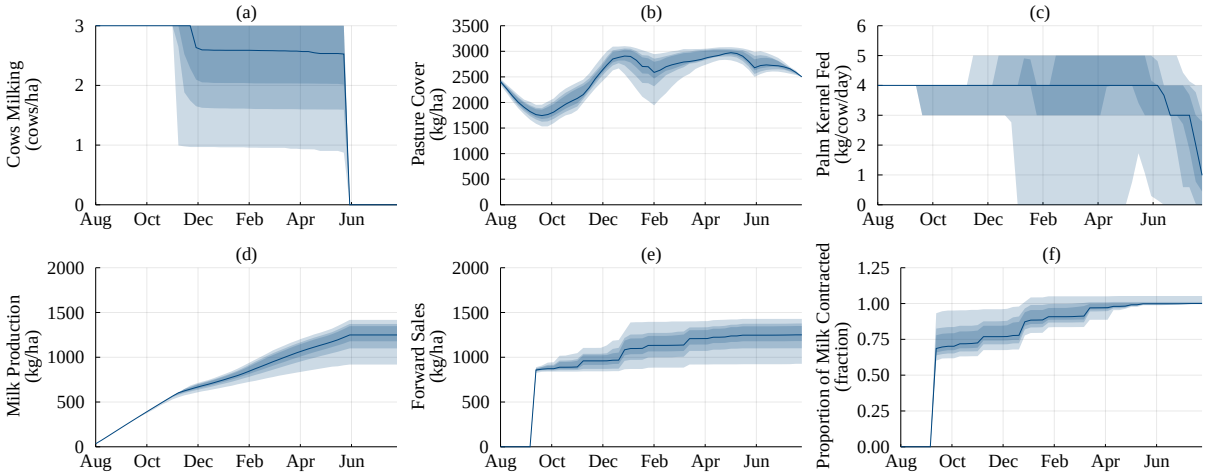


Figure 9.12: Monte Carlo simulation using the high-risk-averse policy with contracting.

In week seven, approximately of 800 kg of milk is sold on the forward market in all scenarios (Figure 9.12e). This equates to roughly 60% of the total average seasonal production (Figure 9.12f). Following this initial decision, the remainder of the season's contracting decisions are similar to the low-risk-averse policy (Figure 9.10).

9.3.4 Discussion

Without access to forward markets, farmers have little ability to offset their risk through on-farm management actions. In contrast, forward selling milk allows the farmer to reduce their exposure to the greatest contributor to variance in operating profit: price risk. Therefore, at the start of the season, risk-averse farmers should forward sell a quantity that they are highly likely to produce over the season. Then, over the season, they should sell further quantities as they become more confident about their total production volume. The initial sales quantity is closely tied to the trade-off between the reduction in price-risk from the sold quantity compared with the increase in risk from over-contracting. However, given the large reduction in variance compared with the reduction in expected operation profit, even farmers who are risk-neutral should consider engaging in forward contracting.

Moreover, the results also demonstrate the counter-intuitive results that can arise as a result of using nested risk measures. Various authors [14, 109, 156] suggest that counter-intuitive results such as these can arise because the nested risk measure may not reflect the modeller's actual risk aversion. In Figure 9.6, we plot the end-of-horizon distribution in operating profit. However, the risk measures we used in each stage were risk-averse against the stage-objective

plus the risk-adjusted expectation of the future stages, which were themselves risk-averse against the risk-adjusted expectation of the future stages. Recall from our example in Figure 1.14 that this nesting behaviour can lead to perverse outcomes (from an end-of-horizon perspective) when the model is risk-averse even in good scenarios (e.g. high grass growth and price). As such, although we have a mathematical understanding of the nested risk measures, we have no intuitive understanding of how the policies will differ as we vary the parameters.

Instead of using nested risk measures, Baucke et al. [14] propose an algorithm for solving multistage stochastic programs using an *end-of-horizon* risk measure. In the farming setting, an *end-of-horizon* risk measure makes intuitive sense as the farmer is likely to be risk-averse about the distribution of the entire season's operating profit, rather than some nested measure.

9.3.5 The value of price uncertainty

In Section 8.2.4 we investigated the value of incorporating price uncertainty into the POWDER model. We found that, on average, modelling the sequence of milk price forecasts by a scenario tree with nine scenarios allowed the farmer to increase their expected operating profit by \$11/ha. However, we conjectured that this benefit may increase if a more complicated price process was modelled. In this section, we analyse the value of modelling the sequence of forecast milk prices using the one-state price model compared with the *fixed-price* model that we developed in Section 8.2.4.

To do so, we take the 2000 replications of the risk-neutral Monte Carlo simulation conducted above and simulate those replications of weather and price uncertainty using the fixed-price policy. We provide a density plot of the two distributions of operating profit in Figure 9.13a. Then, for each of those 2000 replications, we calculate the difference between the operating profit of the risk-neutral policy with the one-state price model and the fixed-price policy. In Figure 9.13b we plot this difference in operating profit against the end-of-season milk price. Positive differences indicate that the risk-neutral model outperformed the fixed-price model.

On average, the risk-neutral model outperformed the fixed-price model by \$17.98/ha (95% confidence interval of $[15.76, 20.20]$). This is larger than the difference observed in the previous chapter ($+\$11$/ha), but still small compared to the expected operating profit of \$2604/ha. Like Figure 8.10, the difference is not uniform over the end-of-season milk prices. In low milk price seasons (i.e. $p_{53} < \$5$/kg) and high milk price seasons (i.e. $p_{53} > \$7.50$/kg), the risk-neutral policy outperformed the fixed-price policy. This reinforces our finding from the previous chapter that on average, there is little value in considering price uncertainty. However, at the extremes of the end-of-season milk price distribution, considering updated forecast milk price information can lead to better decision-making. In the worst case, the minimum operating profit increased from $-\$2315$/ha using the fixed-price policy to $-\$1893$/ha using the risk-neutral policy ($+\$422$/ha). Therefore, a policy that is responsive to changes in the forecast milk price is especially attractive to risk-averse farmers.

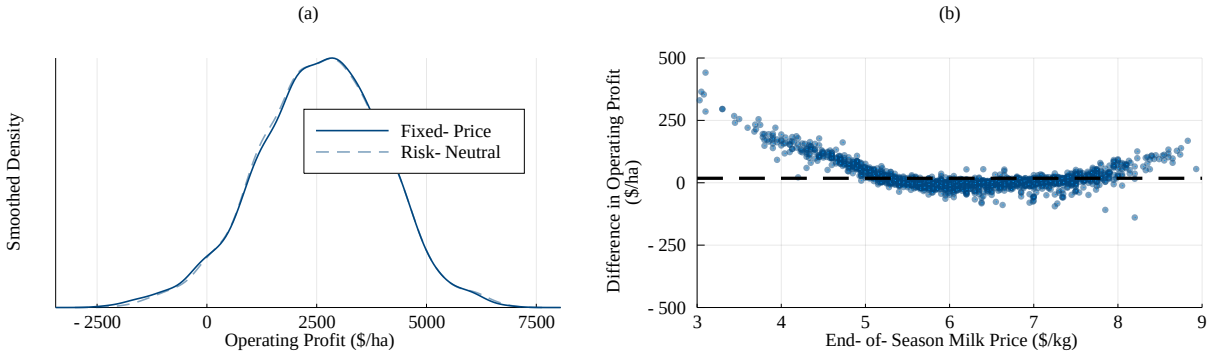


Figure 9.13: Two views of the simulated operating profit from the 2000 Monte Carlo replications of the risk-neutral policy using the one-state price model and the fixed-price model. (a) Density plot of the operating profit for the fixed-price and risk-neutral policies. (b) Scatter plot of the difference in operating profit between the risk-neutral and fixed-price policies. Positive differences indicate that the risk-neutral model outperformed the fixed-price model. Thick, dashed, horizontal line is the mean difference.

9.4 Case study II

In this section we investigate the application of POWDER-II with the two-state price process to the same farm considered in the previous section.

Initially, we modified the model from Section 9.3 by substituting the one-state price process for the two-state price process. In addition, 10 possible observations for the price noise ϕ_t^g (Eq. 9.3) were sampled in each stage. However, the problem was very large. There were 52 stages, five concave states, and two convex states. In addition, we sampled the Cartesian product of the weather and price scenarios. Therefore, there were approximately 10^{92} combinations of weather and price scenarios. Given that POWDER-II with the one-state price process already took orders of magnitude longer to solve than POWDER, we could not solve POWDER-II with the two-state price process as formulated in a reasonable time. Thus, to simplify the problem and make it easier to solve, we make a number of modifications to POWDER-II. These changes are:

1. the aggregation of stages from 52 weeks to 24 twice-monthly blocks;
2. the removal of the feed storage state Q_t so that $Q_t = h_t = f_t^q = 0 \; \forall t$;
3. the reduction in the number of elements in the sample space of the weather noise from 20 to 10; and
4. the reduction in the number of elements in the sample space of the price noise from 10 to 5.

This reduces the number of stages from 52 to 24, the number of states from seven to six, and the total number of scenarios from 10^{92} to 10^{41}. As before, we use the same data as in Section 9.3. We now detail how the two-state price process was calibrated.

9.4.1 Calibration

Historical GlobalDairyTrade data was obtained from June 1st, 2010 (Auction 24) until November 22nd, 2017 (Auction 200).[6] This data included the average winning sales price for the five reference commodity products (whole milk powder, skim milk powder, anhydrous milk fat, butter, and butter milk powder) in each auction. For each auction, the five prices were averaged using the quarterly product mix reported by Fonterra [75] to obtain a *product-weighted* average sales price. Next, the product-weighted average sales price was converted from United States dollars into New Zealand dollars using the average exchange rate achieved by Fonterra over the last seven years (0.75 USD/NZD) [75]. Lastly, the average processing cost of NZ$2.10/kg [75] was subtracted from the product-weighted average sales price to obtain an estimate for the price of milk in each auction that was distributed to the farmers (i.e. g_t in Eq. 9.3). This methodology is also used by AgriHQ [2]. Finally, ordinary least squares was used to estimate the parameters in Eq. 9.3. This gives:

$$g_{t+1} = 0.97658 g_t + 0.14266 + \phi_t^g. \tag{9.9}$$

This is equivalent to the mean-reverting process:

$$g_{t+1} = \lambda g_t + (1-\lambda)\mu + \phi_t^g,$$

where $\lambda = 0.97658$, and the mean $\mu = \$6.09$/kg. For simplicity, the distribution of ϕ_t^g was created by dividing the residuals of the ordinary least squares model into five sets according to the quintiles, taking the mean of each set, and then subtracting the mean of the five values so that $\mathbb{E}[\phi_t^g] = 0$. We refer to a single realisation of GDT prices $g_1, g_2, \ldots, g_{52}$ as a *price scenario*. In Figure 9.14a we plot the historical spot price g_t as a thick, blue line. In addition, we plot 50 simulated price scenarios of the spot price using Eq. 9.9. The thin, black line is one randomly chosen price scenario. The highlighted price scenario demonstrates similar behaviour to that observed in the historical data. The orange, dashed line is the historical long-run mean of the spot price ($6.09/kg). Since the value of ϕ_t^g is sampled independently in each of the 24 stages, there are $5^{24} \approx 6 \times 10^{16}$ possible price scenarios.

To estimate the sales curve w_t, we extracted the monthly contracted sales volume of the reference commodity products from Fonterra [75] for the last five seasons, and then normalised each season to estimate the fraction of sales that were contracted each month. Then, for each month, we averaged the five seasons, before again normalising the average sales curve so that $\sum_{t=1,\ldots,T} w_t = 1$. The final sales curve is shown in Figure 9.14b. Note that we assume that the two auctions in each month have the same sales fraction.

[6] This information was publicly available until July 2016. Since then, historical data has been moved behind a paywall. This chapter uses data from an Excel file published by GlobalDairyTrade prior to July 2016, as well as data manually collected by the author that is published at `http://www.globaldairytrade.info/en/product-results/` after each trading event.

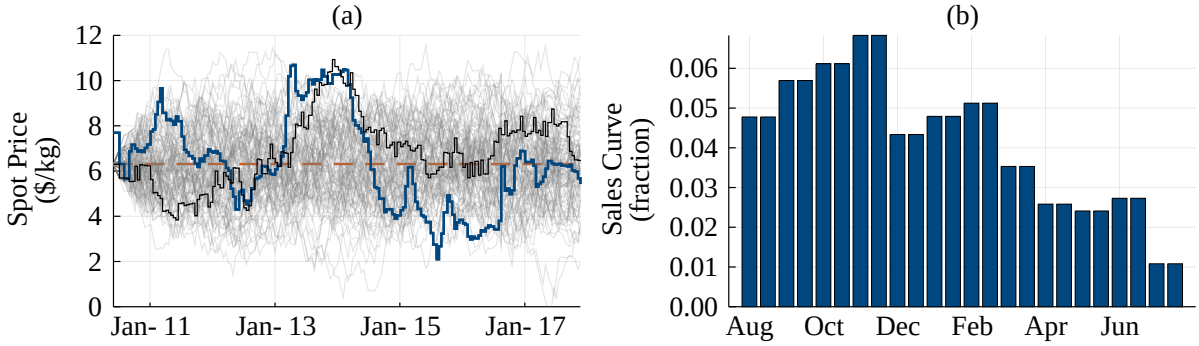


Figure 9.14: (a) Simulated sequences (grey) of GDT spot price g_t. The thin, black line is one randomly chosen sequence. The thick, blue line is the historical sequence. The dashed, horizontal line is the long-run average spot price ($6.09). (b) Estimated sales curve w_t.

9.4.2 Solution method

The POWDER-II model using the two-state price process was solved for 20,000 iterations using the dynamic interpolation method from Chapter 4 in a risk-averse setting ($\lambda = 0.5$) with contracting. The initial conditions were $(W_1, P_1, C_1, M_1, g_1, a_1) = (150, 2500, 3, 0, 6, 0)$. After the model had converged, a Monte Carlo simulation of the optimal policy was conducted with 5000 replications. The model was solved using a virtual-machine with 16 Intel Xeon E5-2698 CPUs and 64GB of memory. The solution time for 20,000 iterations was on the order of 10 hours. In total, the solution process involved solving over 52 million linear programs.

9.4.3 Results

In Table 9.2 we present statistics from the 5000 replications of the Monte Carlo simulation. Compared to the one-state price model policies (Table 9.1), the two-state policy consumes more grass (median of 12.87 t/ha compared with ≈12.1 t/ha) and produces more milk (median of 1445 kg/ha compared with ≈1370 kg/ha). However, due to the lower median end-of-season milk price in the two-state price model ($6.10/kg) compared with the one-state price model ($6.25/kg), the median operating profit is similar to the low-risk-averse one-state solution with contracting ($2627/kg compared with $2638/kg).

The distribution of the simulated end-of-season milk prices using the two-state price model (first row of Table 9.2) fits the distribution of the eight historical examples of the Fonterra end-of-season milk price (Figure 9.5a): the simulated median was $6.10/kg compared with the historical median of $6.09/kg, while two out the eight years fell below the 25$^{\text{th}}$ percentile ($4.40/kg in 2014/15 and $3.90/kg in 2015/16), and two out the eight years fell above the 25$^{\text{th}}$ percentile ($7.60/kg in 2010/11 and $8.40/kg in 2013/14).

In Figure 9.15 we visualise the optimal risk-averse policy with contracting and the two-state price process using subplots similar to those presented in Section 9.3. However, instead of plotting the distribution of all 5000 replications, we partitioned the replications into two subsets

	Simulation Percentiles				
	0	25	50	75	100
End-of-Season Milk Price ($/kg)	3.32	5.58	6.10	6.60	8.67
Palm Kernel Cost ($/t)	500	500	500	500	500
Lactation Length (months)	7.5	10.0	10.0	10.0	10.0
Milk Production (kg)					
per Hectare	1004	1365	1425	1469	1636
per Cow	335	455	475	490	545
Milk Revenue ($/ha)	7026	8436	8794	9064	10580
Feed Consumed (t/ha)					
Pasture	10.84	12.66	12.87	13.07	13.73
Palm Kernel	1.5	3.47	4.03	4.22	5.52
% Feed Imported	10.9	21.3	23.7	24.8	30.8
Palm Kernel Expense ($/ha)	751	1735	2017	2109	2759
Fixed Expense ($/ha)	3536	3536	3536	3536	3536
Operating Profit ($/ha)	821	2451	2627	2806	3647
Mean Operating Profit ($/ha)			2625		

Table 9.2: Summary of Monte Carlo simulation for risk-averse policy with two-state price model.

in order to highlight how different sequences of forecast milk prices lead to different sequences of actions. The first subset, plotted in blue, is the set of all replications that sampled a spot price in the bottom 20% of values in the 12$^{\text{th}}$ stage (i.e. $g_{12} \leq$\$5/kg). The second, plotted in orange, is the set of all replications that sampled a spot price in the top 20% of values in the 12$^{\text{th}}$ stage (i.e. $g_{12} \geq$\$7.05/kg). Since there were 5000 replications, each of these subsets contains 1000 replications. In addition to the subplots (a) to (f), we present three new subplots: Figure 9.15g, the spot price g_t; Figure 9.15h, the accumulated value of sales revenue a_t; and Figure 9.15i, the forward milk price p_t^f.

Accounting for the difference in staging, we see very similar results to those of the risk-neutral policy using the one-state price process. (The results are not identical as there is a different distribution for the end-of-season milk price.) The herd is dried-off at approximately the same time (Figure 9.15a), a similar level of pasture cover is maintained (Figure 9.15b), and a similar quantity of palm kernel is fed (Figure 9.15c). Compared with the low-price replications (plotted in blue), the high-price replications (plotted in orange) milk the cows for longer (Figure 9.15a), approach the end of the season with less pasture cover (Figure 9.15b), and feed more palm kernel (Figure 9.15c). The result of these three changes is to increase total milk production (Figure 9.15d).

Notably, the solution over-contracts (i.e. it sells more milk forward than is produced over the season) in approximately 10% of simulations (Figure 9.15e and f), even though doing so exposes the farmer to increased price risk.

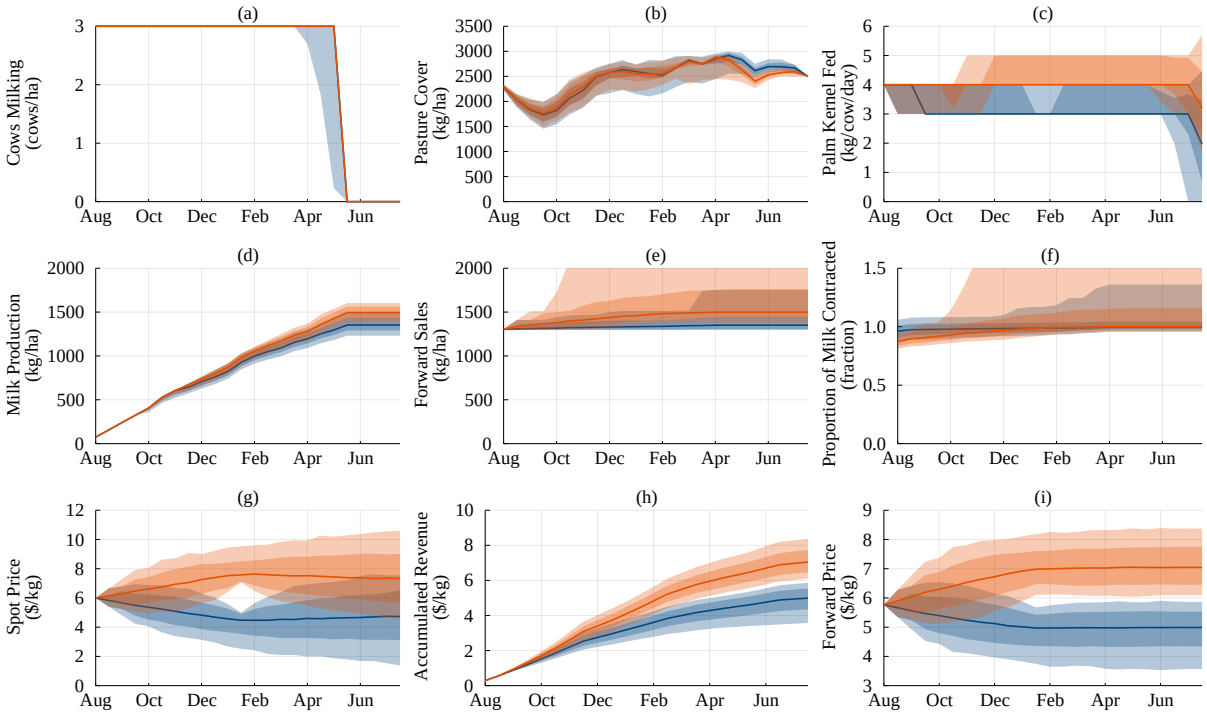


Figure 9.15: Monte Carlo simulation using the optimal risk-averse policy with contracting and the two-state price process.

9.4.4 Discussion

We cannot directly compare the policies of the one-state and two-state price models with contracting since, even for the same price and weather scenarios, they will have different estimates for the conditional expectation of the end-of-season milk price (i.e. the forward milk price). Therefore, the policies will incorrectly perceive an arbitrage opportunity and make sub-optimal decisions.

We conjecture that over-contracting is not an optimal decision, and that seeing such decisions in the simulation of the policy arises from terminating SDDP before the policy has converged in all possible states. If we perform more iterations, the policy will converge; however, this is computationally expensive. Nevertheless, the general trends observed in both case studies appear consistent. An initial quantity of milk is contracted early in the season (although the exact quantity depends upon the farmer's risk appetite), and if necessary, more is sold during the season. Furthermore, the farmer feeds between three and four kilograms of palm kernel per cow per day over the season, increasing to five kilograms if the price is high, and dropping to three is the price is low. These results also suggest that the simplifications we made (reducing the number of stages, removing the storage state Q_t, and reducing the number of noise realisations) did not have a large impact on the general farm management policy. However, at the same time, incorporating a more detailed price process did not yield any new insights. This is because the decisions of the farmer in week t are largely based on the forward milk price p_t^f. Assuming that this price is an unbiased estimator for the end-of-season milk price, the farmer does not need to

consider whether the spot price g_t is high (and therefore likely to mean revert downwards) or low (and therefore likely to mean revert upwards) since that information will be reflected in the forward milk price. Therefore, in the remainder of this chapter, we only consider the results of the one-state price process.

9.5 Pre-season contracting

In the previous two sections we described the solution of large multistage stochastic programs with integrated risk management. In this section, we explore a class of simpler risk management strategies.

9.5.1 Milk price futures

In the results of the case studies, the general contracting trend is to sell a large quantity of milk on the forward market early in the season, and then slowly sell additional contracts as the season unfolds. Therefore, we propose a heuristic that contracts a fixed amount on the first day of the season and then operates according to a policy we constructed in Section 9.3, ignoring the optimal contracting decisions. These represent a simple and implementable class of contracting policies, although the results will not be optimal since the farmer could have made different management decisions during the season as a result of knowing the initial contracting decisions.

To investigate this contracting heuristic, we first choose a policy from Section 9.3 (i.e. the risk neutral model with the one-state price process). Then, we take the 2000 replications of the corresponding Monte Carlo simulation and post-process the data assuming different contracting decisions were made prior to the start of the season. We denote the set of 2000 replications as our sample space Ω, and each replication as a scenario ω in that sample space. The unhedged return in scenario ω, R_ω, is the cumulative cost of the scenario assuming that no contracting decisions were made.

If the farmer sells h_p kg of milk price futures on the first day of the season, then the hedged return X_ω for the farmer in scenario $\omega \in \Omega$ is:

$$X_\omega = R_\omega + h_p \left(\mathbb{E}_\omega[p_{53,\omega}] - p_{53,\omega} \right), \tag{9.10}$$

where $p_{53,\omega}$ is the end-of-season milk price in scenario ω. For simplicity, we ignore transaction costs.

In Figure 9.16, we plot the distribution in operating profit that arises from applying this heuristic to three policies: (1) the *fixed-price* policy from Section 8.2.4 (Figure 9.16a), (2) the risk-neutral policy (Figure 9.16b), and (3) the medium-risk-averse with contracting policy (Figure 9.16c). On the x-axis of each subplot, we plot the initial contracting amount h_p, and

on the y-axis, we plot the hedged operating profit X_ω. The dotted grey lines represent selected percentiles of the distribution in operating profit (0, 10, 25, 50, 75, 90, and 100) as we vary the initial contracting amount h_p. The thick, black line is the mean of the distribution of the hedged operating profit X_ω, while the thick, curved, dashed line is the standard deviation of the hedged operating profit. The vertical line is the contracting amount that minimises the standard deviation. This quantity is well known in the literature as the *Minimum Variance Hedge Ratio* [26]. (See Appendix F for a derivation showing why this is the case.)

Definition 33. *Minimum Variance Hedge Ratio:* Given a vector of unhedged returns R_ω and a vector of the prices of the hedge instrument p_ω, such that the hedged return X_ω is:

$$X_\omega = R_\omega + h(\mathbb{E}_\omega[p_\omega] - p_\omega),$$

the quantity h^* that minimises the variance of X_ω is:

$$h^* = \frac{\text{cov}(R, p)}{\text{var}(p)}.$$

We refer to h^* as the *minimum variance hedge ratio*.

For our experiments, the minimum variance hedge ratio is 1366 kg for the fixed-price case (Figure 9.16a), 1344 kg for the risk-neutral case (Figure 9.16b), and 1329 kg for the medium-risk-averse case with contracting (Figure 9.16c). The values are approximately equal to the expected annual production in each case. By examining the figures, it is possible to identify other desirable hedge ratios based on different risk measures. For example, a smaller quantity than the minimum variance hedge ratio allows the farmer to capture more of the upside potential without negatively affecting their downside risk. Note that the expected return (thick black line) does not change with the contracting amount as we ignored transaction costs.

In each of Figures 9.16d, 9.16e, and 9.16f, we present scatter plots of the operating profit for the unhedged (dark) and optimally hedged (using the minimum variance hedge ratio) (light) cases against the end-of-season milk price. In the unhedged case, there is a strong correlation ($r \approx 0.95$) between the operating profit and end-of-season milk price p_{53}. However, in the hedged case, the relationship between end-of-season milk price and operating profit has been removed ($r = 0.0$). Moreover, there is very little difference between the fixed-price, risk-neutral, and medium-risk-averse policies. This is also shown by the density plots in Figures 9.16g – 9.16i, which plot the distribution of the hedged (dashed) and unhedged (solid) operating profit for each policy.

Therefore, a potentially good hedging strategy is to choose a policy (e.g. the risk-neutral policy), simulate the distributions of price and operating profit using that policy, contract the minimum variance hedge on the first day of the season, and then operate according to the chosen policy. On our farm, the minimum variance hedge ratio is very close to the expected milk production. Therefore, the hedging "rule of thumb" that could be communicated to the farmer

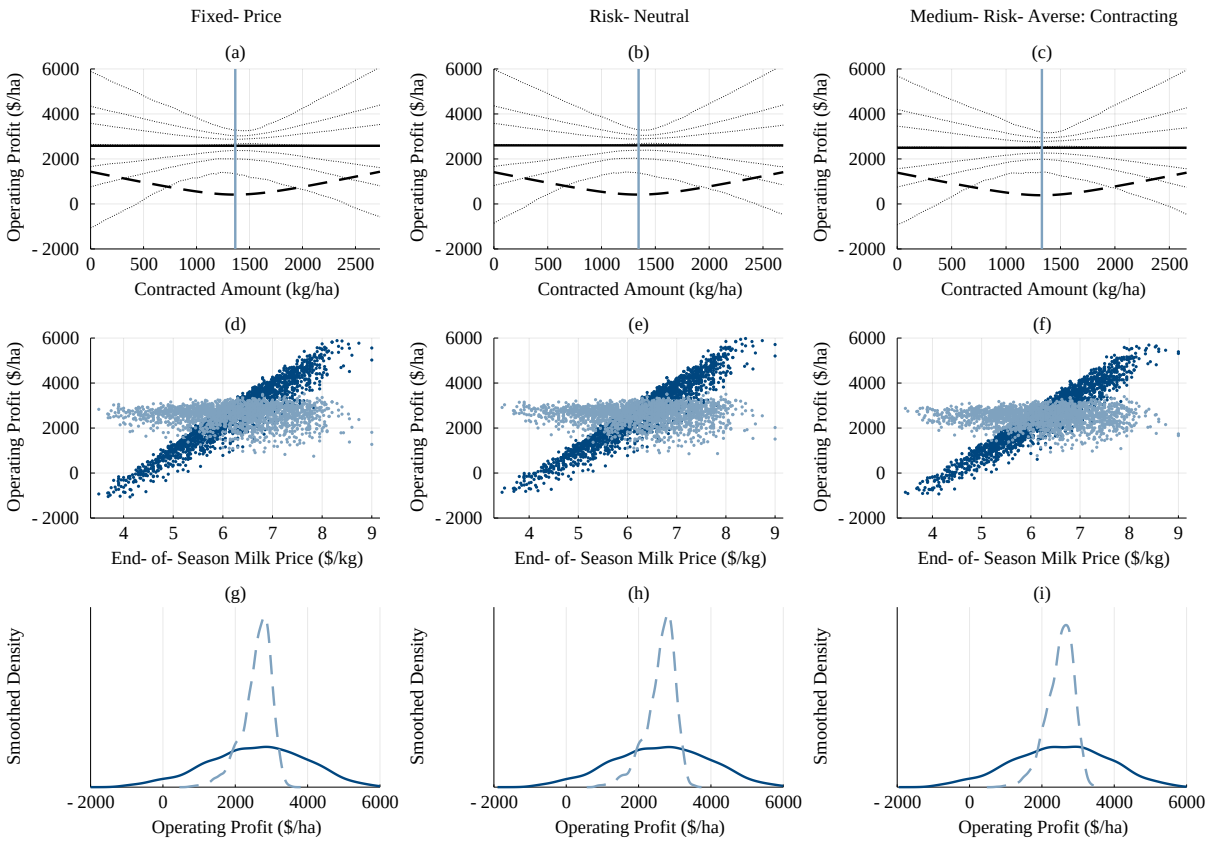


Figure 9.16: Distribution in operating profit assuming a single contracting opportunity on the first day of the season. Figures (a) – (c): plots of operating profit against the contracted amount h_p. Light, dashed lines are selected percentiles of the distribution in operating profit. Solid, horizontal line is the mean of the distribution. Curved, dashed line is the standard deviation of the distribution. Vertical line is the contracting amount that minimises the standard deviation. Figures (d) – (f): scatter plot of operating profit against the end-of-season milk price. Dark coloured dots are the unhedged returns, light coloured dots are the returns after hedging the milk price futures at the minimum variance hedge ration. Figures (g) – (i): density plots of the distribution of operating profit for the hedged (dashed) and unhedged (solid) returns.

is "on the first day of each year, hedge the average of your production over the last five seasons." This is a simple, intuitive contracting policy that can easily be understood by the farmer.

The choice of operating policy is up to the farmer. However, our results suggest that for the farm in our case study, there is little difference between a policy that ignores price (i.e. the fixed-price policy), a policy that is risk-neutral and considers the forecast milk price on a weekly basis (i.e. the risk-neutral policy), and a risk-averse policy that considers the forecast milk price on a weekly basis (i.e. the medium-risk-averse policy with contracting). Therefore, it is likely that our farmer does not need to change their current operational decisions when they forward contract for milk.

Moreover, many large farms in New Zealand are run by corporations who own multiple farms. Such corporations centralise their financial operations while delegating on-farm operations to local managers. These results suggest that these corporations can significantly reduce

their exposure to price risk by forward selling a proportion of their total milk production, without needing to change on-farm management practices.[7]

9.5.2 Weather derivatives

Milk price futures enable the farmer to remove a large component of the variance in operating profit. However, they are still exposed to production risk as a result of the weather. Weather derivatives are financial instruments that can be used to reduce the risk associated with adverse or unexpected weather conditions [3]. In what follows, we introduce three hypothetical weather futures, and then assess their utility as risk management tools. To do so, we follow a similar methodology to that described for the milk price contracting above. However, we only analyse the risk-neutral policy.

Grass-growth derivative

First, let us consider a futures contract h_g that settles against the total quantity of grass grown over the course of a season on the farm. We assume that it works as follows. If the annual grass growth is less than the average, then the seller of each contract receives \$1 for every kilogram of the difference between the expected and actual growth. If the annual grass growth is greater than the expected growth, then the seller of each contract must pay the buyer \$1 for every kilogram of the difference. The buyer is likely to be someone whose profit is negatively correlated with grass growth (e.g. a supplier of palm kernel, since the farmer will purchase less palm kernel if they have excess grass).

Since grass growth is a function of farm management decisions, the contract h_g is not an index derivative.[8] Therefore, it is unlikely to be offered in practice. Nevertheless, we investigate its performance. After the milk price has been hedged using the forward contract h_p at the minimum variance hedge ratio, the correlation between the hedged operating profit and the cumulative annual grass growth is ≈ 0.97. Therefore, hedging both the milk price and the grass growth will remove the majority of the variance in operating profit.

If the farmer sells h_g kg of grass-growth contracts, then the hedged return X_ω for the farmer in scenario ω of the 2000 Monte Carlo replications is:

$$X_\omega = R_\omega + h_g \left(\mathbb{E}_\omega \left[\sum_{t=1}^{52} g_{t,\omega} \right] - \sum_{t=1}^{52} g_{t,\omega} \right), \tag{9.11}$$

where R_ω is the unhedged return in scenario ω, and $g_{t,\omega}$ is the quantity of grass grown during stage t in scenario ω.

[7] See Brealy et al. [34, Ch. 26] for a related discussion on the intra-firm transfer of risk from each division (i.e. farm) to a central treasury.

[8] Recall than an index derivative settles against an underlying variable reported by a third-party.

Evapotranspiration derivative

POWDER assumes that grass growth is linearly related to evapotranspiration (which, we recall, is a function of air-temperature and sunlight). Therefore, although actual grass growth is also limited by the pasture cover and soil moisture, it is likely that the grass growth g_t is correlated with the potential evapotranspiration e_t^p. (Note that the potential evapotranspiration in week t, e_t^p, is distinct from the actual evapotranspiration e_t.) Since the potential evapotranspiration is a random variable that is not controlled by the farmer, it is an attractive index against which to settle a derivative. However, after the end-of-season milk price has been hedged (again at the minimum variance hedge ratio), the correlation between the operating profit and the total potential evapotranspiration is ≈ 0.003. Therefore, we can expect that hedging the evapotranspiration will have a minimal effect on the remaining variance.[9] Despite this, we consider a futures contract h_e that settles against the cumulative potential evapotranspiration over the season. We assume that it works as follows. If the cumulative evapotranspiration is less than the average, then the seller of each contract receives \$1 for every mm of the difference between the mean and actual accumulation. If the cumulative evapotranspiration is greater than the average, then the seller of each contract must pay the buyer \$1 for every mm of the difference. The buyer is likely to be someone whose profit is negatively correlated with potential evapotranspiration (e.g. a supplier of palm kernel since the farmer will purchase less palm kernel if they have excess grass).

If the farmer sells h_e mm of potential evapotranspiration contracts, then the hedged return X_ω for the farmer in scenario ω of the 2000 Monte Carlo replications is:

$$X_\omega = R_\omega + h_e \left(\mathbb{E}_\omega \left[\sum_{t=1}^{52} e_{t,\omega}^p \right] - \sum_{t=1}^{52} e_{t,\omega}^p \right), \tag{9.12}$$

where R_ω is the unhedged return in scenario ω, and $e_{t,\omega}^p$ is the potential evapotranspiration during stage t in scenario ω.

Rainfall derivative

The poor correlation ($r \approx 0.003$) between potential evapotranspiration and operating profit is surprising. Other indices (such as annual rainfall) were investigated but found to also correlate poorly with operating profit. However, one index that is correlated ($r = 0.57$) to the operating profit is the log of the cumulative rainfall during the summer period of stages 21 to 30 (i.e. December to March). (The log function accounts for the fact that very low rainfall can force the farmer to dry-off early, so the reduction in operating profit is nonlinear with declining rainfall.) Since rainfall is not controlled by the farmer and is strongly correlated with pasture growth, it is an attractive index derivative which has been explored in the literature (dating back to

[9] The minimum variance hedge ratio is proportional to the correlation coefficient.

Chakravarti [40][10]). If the farmer sells h_r rainfall contracts, then the hedged return X_ω for the farmer in scenario ω of the 2000 Monte Carlo replications is:

$$X_\omega = R_\omega + h_r \left(log \left(\mathbb{E}_\omega \left[\sum_{t=21}^{30} r_{t,\omega} \right] \right) - log \left(\sum_{t=21}^{30} r_{t,\omega} \right) \right), \tag{9.13}$$

where $r_{t,\omega}$ is the quantity of rainfall during stage t in scenario ω.

9.5.3 Comparison and discussion

In Figure 9.17 we plot the distribution of operating profit for six different hedging strategies. The first two strategies (lighted coloured boxes) are from the first case study in this chapter (Section 9.3). They are:

1. Risk-Neutral: the risk-neutral policy without contracting; and
2. Low-Risk-Averse: the low-risk-averse policy with contracting.

The other four policies (dark coloured boxes) use the simple contracting heuristics outlined in this section. They use the 2000 replications of the risk-neutral Monte Carlo simulation as the sample space Ω. In each case, we assume that the farmer sells the minimum variance hedge ratio. The four strategies are:

3. Price Only: sell h_p (Eq. 9.10);
4. Price + Evapotranspiration: sell h_p (Eq. 9.10) and then h_e (Eq. 9.12);
5. Price + Rainfall: sell h_p (Eq. 9.10) and then h_r (Eq. 9.13); and
6. Price + Growth: sell h_p (Eq. 9.10) and then h_g (Eq. 9.11);

The distributions for 'Growth Only' (i.e. only sell h_g), 'Evapotranspiration Only' (i.e. only sell h_e), and 'Rainfall Only' (i.e. only sell h_r) are indistinguishable from the 'Risk-Neutral' case, so we omit them.

Compared with the 'Risk-Neutral' distribution, all hedging strategies reduce the variance in operating profit. The 'Low-Risk-Averse' policy, which is able to dynamically make contracting decisions during the season, has a distribution that is very similar to the 'Price Only' distribution. This suggests that the ability to sell contracts during the season offers limited benefit to the farmer compared with only selling on the first day of the season. The 'Price Only' and 'Price+Evapotranspiration' distributions are identical. This is due to the low correlation between the annual cumulative potential evapotranspiration and the operating profit. Moreover, even though the log of the summer rainfall was moderately correlated with operating

[10]This monograph was re-discovered by Mishra [137]. The use of the multiplicative, rather than additive means to assess rainfall also has historical precedent dating back to Chakravati: "the key point to consider is not the absolute amount of rainfall in any particular year but the percentage of deviation from the average" [137]. However, Chakravati's proposed scheme was a form of insurance that paid a fixed lump sum if the cumulative rainfall was below a certain threshold, rather than the contract-for-difference proposed here.

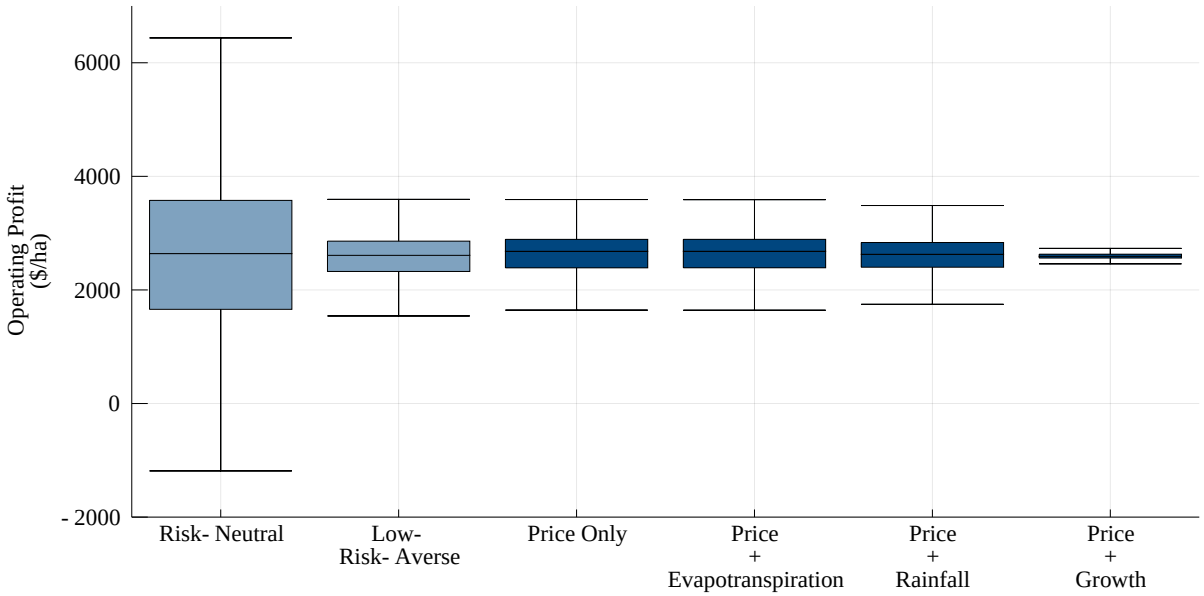


Figure 9.17: Boxplots comparing the distributions of operating profit under different hedging strategies.

profit ($r = 0.57$), the distribution of the 'Price+Rainfall' hedging policy is similar to the 'Price Only' distribution – although the standard deviation has decreased from \$415/ha to \$338/ha. In contrast, hedging price and grass growth ('Price+Growth' distribution) removes almost all the variance in operating profit. The remaining variance is due to differences in timing of when the grass grew and different sequences of the forecast milk prices during the season.

These results suggest that after hedging price risk, the next greatest contributor to variation in operating profit is variation in grass growth. However, as we described earlier, derivatives that settle against actual grass growth are impractical to provide in practice, while simple index derivatives such as summer rainfall are limited by the basis risk that they introduce. Authors such as Vedenov and Barnett [200] reach similar conclusions and note that "rather complicated combinations of weather variables must be used in order to achieve reasonable fits of the relationship between weather and yield." Expanding on this idea, another option is to provide an index derivative that settles against a simulated estimate of grass growth. Commercial grass growth estimation tools such as FARMAX already exist [72], and there is a large literature on different modelling approaches [36, 114, 138, 175]. A key benefit of this approach is that since the inputs to the model are not controllable by the farmer, there is no moral hazard. However, such models would need extensive calibration in order to reduce the basis risk.

Another approach that greatly reduces compliance costs (although not the moral hazard) is to use remote sensing to estimate the grass growth. Attempts to use satellite imagery to estimate grass growth date back to at least Taylor et al. [194] and have been used with some success (e.g. [105, 106]). In the New Zealand setting, at least one commercial tool based on satellite imagery is already in development (LIC SPACE™ – Satellite Pasture And Cover Evaluation).[11] Remote

[11] https://www.lic.co.nz/products-and-services/space/

sensing data could also be used to calibrate other simulation models such as the ones described above.

9.5.4 Milk price options

In the sections above, we have limited our analysis to futures contracts. However, another widely used class of derivative is the option contract. In particular, we consider the farmer buying *European put options*, one contract of which gives them the right to sell one kilogram of milk at a predetermined strike price s_o at the end of the season.[12] In return for this right, the farmer pays the seller of the put option a premium c_o. As before, we assume that this contracting decision happens on the first day of the season and that we operate according to the risk-neutral policy using the one-state price model. We set Ω to be the set of 2000 Monte Carlo replications simulated for the risk-neutral policy in Section 9.3.3

If the farmer buys h_o put options with a strike price s_o at a cost c_o per put, then the hedged return X_ω for the farmer in scenario $\omega \in \Omega$ is:

$$X_\omega = R_\omega + h_o \times \left(\max\left\{0, s_o - p_{53,\omega}\right\} - c_o\right), \tag{9.14}$$

where R_ω is the unhedged return in scenario ω, and $p_{53,\omega}$ is the end-of-season milk price in scenario ω. We assume that the cost of each put c_o is priced so as to make the option actuarially fair (i.e. $\mathbb{E}[X] = \mathbb{E}[R]$). This effectively sets a minimum price for h_o kilograms of milk of $(s_o - c_o)$, whilst allowing the farmer to profit from any high-price realisations.

Since a put option skews the distribution of operating profits, variance is no longer a useful metric to optimise. Instead, we reprise the AV@R risk measure. Given a quantile β and a put option o, we can calculate the optimal number of put contracts to buy that minimises the average value-at-risk by solving the following (linear) optimisation problem:

$$\begin{aligned}
\text{AV@R}_{1-\beta}[X] = \quad & \min_{\zeta, h_o, z_\omega, X_\omega} \quad \zeta + \tfrac{1}{\beta|\Omega|} \sum_{\omega\in\Omega} z_\omega \\
& \text{s.t.} \quad z_\omega \geq X_\omega - \zeta, \quad \forall \omega \in \Omega \\
& \qquad\; z_\omega \geq 0, \quad \forall \omega \in \Omega \\
& \qquad\; X_\omega = -R_\omega - h_o \times \left(\max\{0, s_o - p_{53,\omega}\} - c_o\right), \quad \forall \omega \in \Omega.
\end{aligned}$$

Note that we have negated the last constraint to turn profits into costs. This optimisation problem can also be used to find the hedge ratio that minimises the average value-at-risk for any of the other contracts described above by replacing the last constraint with the appropriate hedged returns. For example, to calculate the quantity of milk price futures (Eq. 9.10) that minimise

[12] In contrast, an American option allows the buyer to exercise the option at any time before the settlement date. See Brealy et al. [34, Ch. 20] for a good introduction and overview of option contracts.

the average value-at-risk, we replace the last constraint with:

$$X_\omega = -R_\omega - h_p(\mathbb{E}_\omega[p_{53,\omega}] - p_{53,\omega}).$$

To investigate how a farmer might use options contracts, we solve the optimisation problem above given a put option with a strike price of \$6.0, the value-at-risk quantile $\beta = 0.25$, and Ω chosen to be the set of 2000 replications of the Monte Carlo simulation of the risk-neutral policy. In addition, we calculate the quantity of the milk futures contract h_p that also minimised the average value-at-risk when $\beta = 0.25$ using the same set of 2000 replications. The distributions of the profit are plotted in Figure 9.18. Compared with the futures distribution, the options distribution retains the right-hand tail at the expense of shifting the peak of the distribution downwards. Ultimately, the decision of whether to manage price-risk using futures or options contracts (if at all) is up to the personal choice of each farmer.

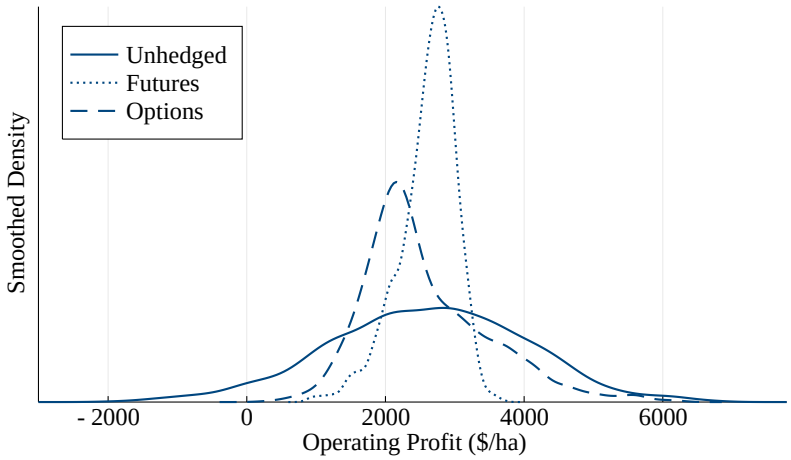


Figure 9.18: Distribution in operating profit using milk price futures and options.

9.6 Summary

In this chapter we extended the POWDER model to incorporate an auto-regressive process for price and the ability of the farmer to sell milk on a forward market. We have shown that risk-averse farmers who do not forward sell their milk have little ability to manage their downside risks as the majority of this risk is due to price volatility. In contrast, farmers who sell their milk forward can greatly reduce the variance in their operating profit with little reduction in their expected operating profit. We also investigated various types of weather derivatives, but these were shown to offer limited benefit in comparison to hedging price.

Supplementary materials The Julia code to implement POWDER-II in `SDDP.jl`, as well as the data needed to run the case study and replicate the results in this chapter, is available at `https://github.com/odow/MilkPOWDERII`.

Chapter 10

Conclusion

10.1 Main results and contributions

In Part II of this thesis we applied the methodology and toolset developed in Part I to problems in the New Zealand dairy industry. We briefly summarise our main results and contributions.

- We developed MOO, a dynamic program which approximates a nonlinear simulation model of a cow.

  Dynamic programming allows us to obtain provably optimal solutions. This is in contrast to similar models such as IDEA [63], which use nonlinear optimisation and may find solutions that are sub-optimal.

- Using MOO, we demonstrated that the stocking rate of the farm investigated in the case study was sub-optimal.

  In particular, we identified that the farmer should either increase their intensity of supplementation from 2.85 t/ha to 5 t/ha or decrease their stocking rate from 3 cows/ha to 2.75 cows/ha. The farmer has since lowered the stocking rate to 2.7 cows/ha which is close to MOO's recommendation.

- Using MOO, we demonstrated that purchasing a fixed quantity of supplement at the start of the season can increase risk.

  In our opinion, risk is an area that is under-explored in the literature as most authors only consider deterministic models.

- We developed POWDER, a large multistage stochastic optimisation model of a pastoral dairy farm.

  POWDER reduces the gap in the literature identified by Borodin et al. [32], who noted that there were few large, real-world applications of multistage stochastic optimisation in an agricultural context.

- Using POWDER, we demonstrated that the current stocking rate of the farm investigated in the case study was sub-optimal.

  We showed that reducing the stocking rate from 3 cows/ha to 2.7 cows/ha increased the operating profit in every simulated scenario. This reinforces our findings from MOO.

- We developed two models for forecasting the Fonterra end-of-season milk price.

  In addition to their use in POWDER-II, these models can also be used to value the NZX (New Zealand Stock Exchange) milk price futures and options [148].

- We developed POWDER-II, an extension to POWDER with an improved model for price uncertainty and integrated risk management.

  POWDER-II further reduces the gap in the literature identified by Borodin et al. [32]. It also demonstrates that risk management tools can be successfully integrated into a multistage stochastic optimisation model and solved to find an approximately optimal policy.

- Using POWDER-II, we showed that for the farm investigated in the case study, the variation in operating profit due to price variation is much greater than the variation in operating profit due to weather variation.

  This reinforces our view that the impact of price volatility on models of farm decision-making is an area that is under-explored in the literature as most authors only consider fixed milk prices.

- We demonstrated how milk price futures and options can improve the welfare of a risk-averse farmer.

  The large reduction in the variance of a farmer's operating profit compared to the small reduction in expected profit indicates that the NZX milk price futures and options are valuable tools for New Zealand dairy farmers. Educating farmers about these tools, how they work, and the benefits they provide, should be given high priority by the industry in order to overcome the limited adoption of these tools that has been observed in other dairy markets [205, 206].

- In contrast, we showed how weather derivatives are not able to improve the welfare of a risk-averse farmer.

  This is because derivatives that settle against actual pasture growth are impractical to provide in practice due to the problems of adverse selection and moral hazard, while simple index derivatives such as cumulative rainfall are limited by the basis risk that they introduce.

Summarised into a single finding, this half of the thesis suggests that price volatility plays a major role in the decision-making and profitability of New Zealand dairy farmers. The current literature has focused on improving profitability by increasing production and reducing operating costs. We suggest that this analysis is inadequate without consideration of the associated variation in operating profit and downside risk.

10.2 Future work

Part II focused on applications in the New Zealand dairy industry. However, there is nothing in our methodology that limits the work to pastoral dairy farms. In its simplest form, the POWDER model comprises four models: weather, crop, animal, and price. In pastoral dairy farming, the crop is grass and the animal is a cow. One could easily change the cow for a sheep, or remove the animal model and change the crop model to wheat, soy, or rice. This greatly expands the potential areas of application.

It is also interesting to consider this research in the context of climate change. Extreme weather events are likely to become more frequent in the future and will increase risks and uncertainties within the global food system [203]. In an article on the impact of food security due to climate change in *Science*, Wheeler and von Braun [203] write:

> [There is] the need for considerable investment in adaptation and mitigation actions to prevent the impacts of climate change from slowing progress in eradicating global hunger and under-nutrition. [....] Building agricultural resilience, or 'climate-smart agriculture,' through improvements in technology and management systems is a key part of this.

In our opinion, multistage stochastic optimisation is well equipped to deal with this problem of 'climate-smart agriculture.' We therefore suggest the following topics as avenues of future research.

- Apply multistage stochastic optimisation to other agricultural systems.
- Extend the models to multi-year time horizons.
- Investigate the impact of climate volatility on optimal management policies.
- Explore optimal adaptation strategies. Should a given cropping farm grow solely wheat? Or should it diversify into wheat and corn?

There are also a number of more specific directions relating to the models we developed in this thesis.

- Evaluate MOO and POWDER on different farms.

In Part II, we focused on a single farm in the Bay of Plenty of New Zealand. Using MOO and POWDER, we were able to obtain insight into optimal management decisions for this farm (e.g. we can increase operating profit by decreasing the stocking rate, and by selling milk on the forward market we can reduce the variance in operating profit). It would be interesting to apply the model to a range of farms in order to compare and contrast the different optimal management policies.

- Improve the animal model in POWDER.

  In order to solve the POWDER model using SDDP, we used a simplified model of a cow. Compared to the detailed *E-Cow* model which proportions net energy into fat deposition (i.e. change in body condition) and milk production, our simplified version assumes a fixed body condition trajectory with any net energy apportioned to milk production. Work could be done to investigate better convex approximations of a dairy cow.

- Allow the farmer to change the stocking rate during the season.

  One limitation of POWDER is that the stocking rate is fixed. The ability to buy and sell stock during the year is an additional management action that we have not considered in this thesis.

- Investigate the benefit of a corporate ownership structure compared with a single owner-operator.

  If multiple farms are managed collectively do the optimal management actions change? If the farms are in different geographical locations (i.e. experiencing different climatic conditions), can stock be transferred between farms to manage risk?

- Incorporate emissions into the model.

  The environmental impact of dairy farming is large. It would be interesting to incorporate nitrogen leeching and methane emissions into the model, and then investigate the trade-off between operating profit and the environmental impact.

- Solve POWDER-II using other risk measures.

  Finally, we noted in Chapter 9 how the use of nested risk measures led to counter-intuitive solutions: more risk-averse single period risk measures led to increased risk in the end-of-horizon distribution in operating profit. It would be interesting to solve POWDER using other risk measures such as expected conditional risk measures [109] and the end-of-horizon risk measures of Baucke et al. [14].

Summary

The focus of this thesis has been decision making under uncertainty. In particular, we considered the problem of making an optimal sequence of decisions in time when the future is uncertain. The thesis was split into two distinct parts. The first, *Multistage Stochastic Optimisation*, addressed the theory and computation of multistage stochastic optimisation. The second, *Applications in the Dairy Industry*, applied the theory and techniques developed in the first part of the thesis to the problems faced by pastoral dairy farmers in New Zealand.

There are two main contributions from our work. First, our main contribution from Part I is the `SDDP.jl` library for solving multistage stochastic optimisation problems using stochastic dual dynamic programming. Second, although we developed a number of specific models for the New Zealand dairy industry, our main contribution from Part II is to demonstrate that multistage stochastic optimisation models can be formulated and solved in an agricultural context.

Given the challenges facing the global community posed by climate change, a growing population, and an increasing awareness of the environmental impact of agriculture, the need to farm smarter in the face of uncertainty has never been more pressing. This thesis provides evidence that multistage stochastic optimisation is an ideal tool to help participants in the agricultural sector make better decisions under uncertainty.

Part III

Appendices

Appendix A

DynamicProgramming.jl: a Julia package for stochastic dynamic programming

In this appendix we detail `DynamicProgramming.jl`, an open-source software library we have developed to solve multistage stochastic optimisation problems using backward recursion. The library is written in the Julia programming language [28], and it has a similar interface to `SDDP.jl`, which we introduced in Chapter 3. `DynamicProgramming.jl` takes advantage of Julia's in-built parallelism functionality to scale the solution of large models from the single core of a laptop computer, to multiple cores of a single workstation, to multiple distributed machines in a High-Performance-Computing cluster.

A.1 Introduction

A variety of methods exist for solving multistage stochastic optimisation problems. We introduced one of these, stochastic dual dynamic programming, in Chapter 2. However, we noted that the method requires a number of restrictive assumptions (notably, convexity of the cost-to-go function with respect to the state variables). This limits the types of problems that SDDP can solve. Another solution method is called *backward recursion*. In this method, the state, control, and noise spaces are discretised. Then, in a recursive manner, traversing backward from the leaf nodes of the policy graph to the root node, a complete enumeration is conducted to evaluate the optimal control and cost-to-go from each state in the discretised state-space. We shall not describe the algorithm in detail since it is well known in the literature. Readers are directed to Bellman [23], Bellman and Dreyfus [24], Bertsekas [27], and Birge and Louveaux [30] for a more thorough treatment.

Backward recursion is a simple method for solving multistage stochastic optimisation problems. However, it is computationally demanding since it performs a complete enumeration of the discretised state- and control-spaces. This limits the method to problems with few state and control variables. However, it does not require special assumptions like convexity, and so can

be used to solve more general problems.

A.2 Example: the air conditioner problem

In this section we describe how to solve the air conditioner problem from Chapter 3 using `DynamicProgramming.jl`. We recommend reading Chapter 3 about `SDDP.jl`, and in particular, Section 3.2 (in which we describe how to solve the air conditioner problem using `SDDP.jl`), *before* reading this section, since the syntax and modelling approach of the two libraries are similar.

In the air conditioner problem, the manager of a factory seeks a production plan for producing air conditioners over a period of three months. During standard working hours, the factory can produce 200 units per month at a cost of \$100/unit. Unlimited overtime can be scheduled; however, the cost increases to \$300/unit during those hours. In the first month, there is a known demand of 100 units. In each of months two and three, there is an equally likely demand of either 100, or 300, units. Air conditioners can be stored between months at a cost of \$50/unit, and all demand must be met.

A.2.1 Formulating the problem

We can formulate the air conditioner problem as a multistage stochastic optimisation problem. It has three stages – one for each month of operation. This can be represented by the linear policy graph shown in Figure A.1.

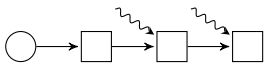


Figure A.1: Policy graph of the air conditioner problem.

We now identify the states, controls, and noise terms in the model. There is one state variable in the model: x_t, the quantity of air conditioners in storage at the start of stage (i.e. month) t. Since the policy graph is linear, x_t can also be thought of as the quantity of air conditioners in storage at the end of stage $t-1$. Therefore we have $x'_t = x_{t+1}$ for $t = 1, 2, 3$. There is one noise in the model: ω_t, the demand for air conditioners in month t. There are two control variables: p_t, the number of air conditioners produced during standard working hours during month t; and o_t, the number of air conditioners produced during overtime working hours during month t. We can formulate each subproblem in the policy graph indexed by $t = 1, 2, 3$ as:

$$
\begin{aligned}
\mathbf{SP}_t: \quad V_t(x_t, \omega_t) = \max_{p_t, o_t, x_{t+1}} \quad & 100p_t + 300o_t + 50x_{t+1} + \mathbb{E}_{\omega_{t+1}}[V_{t+1}(x_{t+1}, \omega_{t+1})] \\
\text{s.t.} \quad & x_t + p_t + o_t - \omega_t = x_{t+1} \\
& 0 \le p_t \le 200 \\
& x_t, x_{t+1}, o_t \ge 0,
\end{aligned}
$$

where $V_4(\cdot,\cdot) = 0$. In each stage, ω_t is independently sampled from a finite discrete distribution so that:

$$\omega_1 = \begin{cases} 100, & \text{with probability } 1, \end{cases}$$

$$\omega_t = \begin{cases} 100, & \text{with probability } 0.5 \\ 300, & \text{with probability } 0.5 \end{cases} \qquad \text{for } t = 2, 3.$$

The root node ($t = 0$) has the initial condition $x_1 = 0$. Therefore, the full optimisation problem faced by the factory manager is:

$$\max \quad \mathbb{E}_{\omega_1}\left[V_1(0, \omega_1)\right]. \tag{A.1}$$

In the next section, we describe how to communicate the formulation of the subproblems $\mathbf{SP}_t$ to `DynamicProgramming.jl`.

A.2.2 Communicating the problem to the solver

First, we need to load the `DynamicProgramming.jl` package. Unlike `SDDP.jl`, we do not need to load JuMP, or a MathProgBase solver such as Clp.

```
using DynamicProgramming
```

Next, we initialise the `SDPModel` object. This is very similar to the `SDDPModel` object in `SDDP.jl`. However, it has fewer keywords and only supports linear policy graphs.

```
m = SDPModel(
        stages = 3,
        sense  = :Min
                    ) do sp, t
    ... subproblem definition ...
end
```

We now need to describe the states, controls, and noise terms. Compared with `SDDP.jl`, which uses JuMP, `DynamicProgramming.jl` has its own syntax. There is one state in the air conditioner problem: the number of units in storage at the start of stage t, which we denote `x`. This state can be created in `DynamicProgramming.jl` as follows.

```
@states(sp, begin
    x in [0, 100, 200, 300]
end)
```

Note that we need to specify how to discretise the domain of `x`. There can only be one `@states` block in each subproblem. If there are multiple states, each state should be on a new line in the

`@states` block. This design was chosen to make it difficult for users to add large numbers of state variables since this will result in the "curse of dimensionality."

There are two controls: `p`, the quantity of air conditioners to produce during normal operating hours; and `o`, the quantity of air conditioners to produce during overtime operating hours. They can be created as follows.

```
@controls(sp, begin
    p in [0, 100, 200]
    o in [0, 100, 200, 300]
end)
```

Note that we define multiple controls in the same `@controls` block, with each control on a new line. Like `@states`, there can only be one `@controls` block in each subproblem.

Finally, the demand for air conditioners is 100 in the first stage, and equally likely to be 100 or 300 in the subsequent stages. We denote the demand by `d`.

```
DEMAND      = [ [100], [100, 300], [100, 300] ]
PROBABILITY = [ [1.0], [0.5, 0.5], [0.5, 0.5] ]
@noises(sp, begin
    d in DiscreteDistribution(DEMAND[t], PROBABILITY[t])
end)
```

Like the controls and states, we can have multiple noise terms; however, these are sampled independently in each stage.

Having defined the states, controls, and noise terms, we now need to define the transition function T_t, the feasibility set U_t, and the stage-objective C_t. First, we describe how constraints are added.

The constraint in the air conditioner problem is that we need to ensure that there are enough air conditioners to meet demand. Therefore, there is a constraint:

$$x_t + p_t + o_t >= d_t.$$

This can be added to the subproblem `sp` as follows.

```
constraints!(sp) do state_in, control, noise
    state_in[x] + control[p] + control[o] >= noise[d]
end
```

This construct uses the same Julia short-hand for anonymous functions that we use in the `SDPModel() do sp, t ... end` constructor. It is equivalent to the following.

```
function foo(state_in, control, noise)
    state_in[x] + control[p] + control[o] >= noise[d]
end
constraints!(foo, sp)
```

The arguments to the function `foo` are: a vector containing values of the incoming state variable, a vector containing values of the current control, and a vector containing values of the current noise realisation. We index these vectors using the names that we defined in the `@states`, `@controls`, and `@noises` macros. We can add multiple constraints by chaining Boolean expressions together. However, the function `foo` must return a single Boolean indicating if the control is feasible given the incoming state and realisation of the noise.

The transition function is more complicated. It takes four arguments: the outgoing state variable, the incoming state variable, the control, and the noise. It should return the stage-objective. Like the constraints function, we can index the state, control, and noise vectors using the names that we defined in the `@states`, `@controls`, and `@noises` macros.

For the air conditioner problem, the transition function is:

$$x_{t+1} = x_t + p_t + o_t - d_t,$$

and the stage-objective is:

$$100p_t + 300o_t + 50x_{t+1}.$$

We can express these in `DynamicProgramming.jl` as follows.

```
dynamics!(sp) do state_out, state_in, control, noise
    state_out[x] = state_in[x] + control[p] + control[o] - noise[d]
    return 100control[p] + 300control[o] + 50state_out[x]
end
```

A.2.3 Solving the problem

Unlike `SDDP.jl`, which only supports Hazard-Decision realisations for the noise, or the generic policy graph framework which permits a mixture of Hazard-Decision and Decision-Hazard nodes, `DynamicProgramming.jl` supports three types of uncertainty realisation.

The first, `WaitAndSee` assumes that every node in the linear policy graph is Hazard-Decision. Thus, we can solve the air conditioner model as follows.

```
solve(m, realisation=WaitAndSee)
```

The second, `realisation=HereAndNow`, assumes that every node in the linear policy graph is Decision-Hazard. The third, `realisation=ExpectedValue`, solves a determin-

istic problem using the expected value of the noise terms. (See Birge and Louveaux [30] for an introduction to these concepts in Stochastic Programming.)

A.2.4 Understanding the solution

Like `SDDP.jl`, we can simulate the policy using identical syntax to `SDDP.jl`.

```
sims = simulate(m, 100, [:x, :p, :o, :d])
```

We can also query the cost-to-go from the start of stage 1 when the incoming state variable $x = 0$ as follows:

```
getbound(m, 1, x=0)
```

This has the same optimal objective value of $62,500 as we found using `SDDP.jl`.

A.3 Correctness

We tested the correctness of the algorithm on a number of problems. These include the inventory control problem of Bertsekas [27, Example 1.3.2], the air conditioner problem described above, the river chain problem from Downward et al. [65], and the one- and two-state price process widget producer examples from Chapter 4. In each case, we obtained the optimal solution. This also helped to validate our implementation of the `SDDP.jl` package which we described in Chapter 3.

A.4 Parallel scaling

Backward recursion is amenable to parallelisation. This is because, at each stage t, given an approximation for the cost-to-go function $\mathcal{V}_{t+1}$, the problem of evaluating the cost-to-go $\mathcal{V}_t(x_t)$ at each point x_t in the discretised state-space is *embarrassingly parallel* [104].

To demonstrate the parallel scaling properties of `DynamicProgramming.jl`, we solved the case study in Section 7.3, using a virtual machine with 20 Intel E5-2698 CPUs with 64GB of memory, for different numbers of Julia processes. The results are plotted in Figure A.2. The dashed grey line represents perfect scaling (i.e. $T_N = T_1/N$, where T_N is the solution time with N processes).

When one Julia process is used, the solution time is approximately 7060 seconds. When two processes are used, we could expect that the solution time is approximately 3530 seconds (i.e. 50% of the initial time). However, the actual solution time is 4250 seconds ($\approx 60\%$). This is because there is some initial overhead in order to set up the parallel solution algorithm. As we increase the number of processes, the solution time scales linearly (on a log-log graph). This demonstrates the efficiency of the parallel implementation.

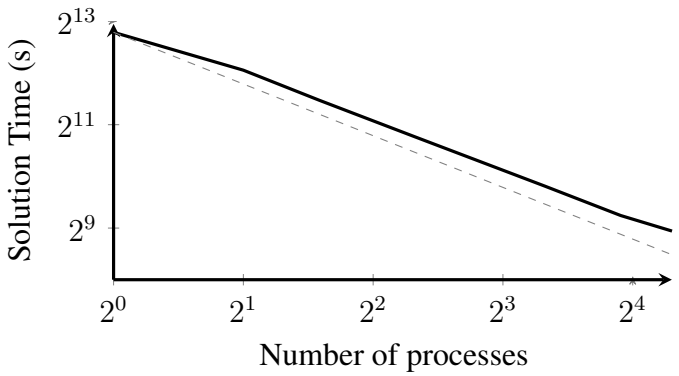


Figure A.2: Solution time against the number of processes, showing the actual (solid) and ideal (dashed) scaling rates.

A.5 Conclusion

In this appendix we introduced `DynamicProgramming.jl`, a Julia library for solving multistage stochastic optimisation problems using backward recursion.

Supplementary materials Source-code and documentation are available as supplementary materials at `https://github.com/odow/DynamicProgramming.jl`.

Appendix B

Addendum for Chapter 2

In this appendix we outline two bugs in the Gurobi Optimiser [98], a state-of-the-art commercial mathematical programming solver that is used in a wide variety of industries and problem settings. The two bugs discovered existed in all versions before version 7.0 of the Gurobi Optimiser.

We begin by examining a simple Linear Program (LP) with a number of unique characteristics. Next, we walk-through a sequence of steps required to trigger the bugs. Lastly, we identify the mechanism behind the observed behaviour and propose a simple fix that can prevent this particular bug from occurring.

Let us consider the following linear program, which could be a first-stage Benders subproblem:

$$
\begin{array}{lrcrcrl}
\min & & & & - & \theta & \\
\text{s.t.} & x & + & y & & & \geq \quad 0 \\
& 2x & - & y & & & \geq \quad 0 \\
& & & & & \theta & \geq -2 \\
& x & & & - & \theta & \geq \quad 0 \\
& -1.000001x & + & y & - & \theta & \geq \quad 0 \\
\multicolumn{7}{c}{x \in [0,1] \quad y \in [0,\infty) \quad \theta \in (-\infty, 1].}
\end{array}
$$

The optimal solution to this LP is $(x, y, \theta) = (1, 2, 0.99999)$ with the objective -0.99999. In addition, the LP has a number of unique features that are worth discussing. The three features are necessary, but not sufficient, to trigger the bug.

1. Both x and y are bounded below by 0. Therefore, the first constraint $x + y \geq 0$ can only be binding at $(x, y, \theta) = (0, 0, \cdot)$ and so is redundant.
2. When $y = 2x$ (i.e. the second constraint is binding), the final two constraints become almost co-linear ($x \geq \theta$ and $0.99999x \geq \theta$).
3. θ is finitely bounded above by 1 and not bounded below. However there exists an additional constraint that bounds θ below by -2.

Computer codes

We now walk through a sequence of commands to solve the above model using the Python [170] interface to the Gurobi Optimiser. However, we note that the bug is not limited to the Python interface and can be triggered via any of the Gurobi APIs.

First, we initialise the model:

```
from gurobipy import *
m=Model()
m.modelSense = GRB.MINIMIZE
```

Next, we add the three variables with appropriate variable bounds:

```
x = m.addVar(lb=0, ub=1)
y = m.addVar(lb=0, ub=GRB.INFINITY)
theta = m.addVar(lb=-GRB.INFINITY, ub=1, obj=1)
m.update()
```

Third, we add the first four constraints and solve the model:

```
m.addConstr(  x + y          >=  0)
m.addConstr(2*x - y          >=  0)
m.addConstr(           theta >= -2)
m.addConstr(  x      - theta >=  0)
m.update()
m.optimize()
```

Gurobi correctly returns the solution status `optimal` with the solution $(x, y, \theta) = (1, 0, 1)$ and objective value -1. We then add the new constraint and re-solve:

```
m.addConstr(-1.000001*x + y - theta >= 0)
m.update()
m.optimize()
```

Gurobi correctly returns the solution status `optimal`, the primal solution (x, y, θ) = (1, 2, 0.99999), and the objective value -0.99999. Now is the point at which we trigger the bug: we query the objective vector and then reset the objective vector with the queried values:

```
c = m.getAttr("Obj", m.getVars())
m.setAttr("Obj", m.getVars(), c)
```

Logic would dictate that these two lines should not change the underlying model in any way. However, if we re-optimise the model (`m.optimize()`) we get a different solution. The status is reported as `optimal`. The objective value is reported as -2 (not -1), and the solution is reported as $(x, y, \theta) = (0, 0, -2)$. While the solution is feasible (but not optimal), the objective value is incorrect. Our objective is $min \ -\theta$ which would imply an objective of $+2$ if θ took the value -2 in the solution. Therefore, Gurobi has returned a sub-optimal solution with an incorrect objective value, but reported the status `optimal`. We also note that although the requirement to reset the objective vector may seem unusual, the modelling package JuMP [66] performs this operation silently before every solve.

What went wrong?

After the first call to `m.optimize()`, Gurobi returned the correct solution. Then, a new constraint was added and the model updated. If we print out the LP file of the problem (using `m.write("model.lp")`), we get a representation of the model that is identical to what we expect.

Next, we call `m.optimize()` and again print out the LP file of the problem. This time, we obtain a different representation of the model compared to what we expect. It shows us that Gurobi has reformulated its internal representation of the model. We call this the *reformulated* problem as the model is now:

$$
\begin{array}{rrcrcrcr}
\min & & & & & + & \theta & \\
\text{s.t.} & x & + & y & & & \geq & 0 \\
& 2x & - & y & & & \geq & 0 \\
& & & & - & \theta & \geq & -2 \\
& x & & & + & \theta & \geq & 0 \\
& -1.000001x & + & y & + & \theta & \geq & 0 \\
& x \in [0,1] & & y \in [0,\infty) & & \theta \in [-1,\infty). & &
\end{array}
$$

The sign of the θ variable has been flipped, and the relevant coefficients in the objective and constraint matrix, as well as variable bounds have been flipped. This illustrates the first (harmless) bug in the Gurobi Optimiser:

Bug 1: Gurobi prints the LP file of the internally reformulated model rather than the user representation of the model.

When we query the objective coefficients, Gurobi correctly "undoes" the reformulation and provides the correct objective vector $[0, 0, -1]$. Therefore, it is only possible to identify that this reformulation has occurred due to the bug in the LP writer. Next, we attempt to set the objective coefficients, triggering the more serious second bug:

Bug 2: When we call to set the objective coefficient vector, we set the vector in the *reformulated* problem.

Therefore, the model Gurobi now contains in memory is:

$$
\begin{array}{lrcrcrcr}
\min & & & & & - & \theta & \\
\text{s.t.} & x & + & y & & & \geq & 0 \\
& 2x & - & y & & & \geq & 0 \\
& & & & & - & \theta \geq & -2 \\
& x & & & & + & \theta \geq & 0 \\
& -1.000001x & + & y & + & \theta & \geq & 0 \\
& x \in [0,1] & & y \in [0,\infty) & & \theta \in [-1,\infty). & &
\end{array}
$$

This has the solution $(x, y, \theta) = (0, 0, 2)$ with the objective -2. However, when we query the solution vector, it again "undoes" the reformulation and returns $(x, y, \theta) = (0, 0, -2)$. Through personal correspondence with Gurobi and our personal experience, we believe this bug is triggered by a reformulation heuristic that runs when a combination of three things occur:

1. a variable exists that is finitely bounded above but not below;
2. no internal scaling is applied to the problem by Gurobi; and
3. (nearly) co-linear constraints exist in the model.

These three necessary conditions are common to first-stage Benders subproblems. However, to avoid this issue, one needs only to define a valid finite lower bound for every variable in a model.

Appendix C

Addendum for Chapter 3

This appendix contains the Julia code necessary to run the air conditioner example in Chapter 3.

Listing C.1: `SDDP.jl` code for the air conditioner example.

```
using JuMP, SDDP, Clp

m = SDDPModel(
              stages = 3,
    objective_bound = 0.0,
               sense = :Min,
              solver = ClpSolver()
                         ) do sp, t
    @variables(sp, begin
        0 <= p <= 200
             o >= 0
             s >= 0
    end)
    @state(sp, xtp1 >= 0, xt == 0)
    @constraint(sp, xt  + p + o - s == xtp1)
    D = [ [100], [100, 300], [100, 300] ]
    @rhsnoise(sp, w=D[t], s == w)
    P = [ [1.0], [0.5, 0.5], [0.5, 0.5] ]
    setnoiseprobability!(sp, P[t])
    @stageobjective(sp, 100 * p + 300 * o + 50 * xtp1)
end

status = solve(m, max_iterations=10)
getbound(m) # should be 62,500

sims = simulate(m, 100, [:xtp1, :p, :o, :s])
```

Appendix D

Addendum for Chapter 7

In this appendix, we present plots of the policies simulated in Section 7.4.2. Each figure contains six subplots. The variables plotted are:

(a) the body condition score x_t^b;
(b) the energy required for maintenance x_t^e;
(c) the quantity of milk solids produced;
(d) the pasture cover x_t^g;
(e) the quantity of grass consumed per day; and
(f) the quantity u_t^q of palm kernel consumed per day.

In some subplots, the trajectory oscillates up and down between weeks (e.g. Figure D.3c). It is likely that a smoother trajectory exists (e.g. gradually reducing the quantity of milk produced) with an objective that is near optimal, if not equal in value to the optimal objective. This is because, provided the end-of-horizon constraints are met (e.g. $x^b 53 \geq 3.2$ and $x_{53}^g \geq 2500$), there is little difference between feeding the cow 10 kg of grass today and 11 kg of grass tomorrow, and the reverse.

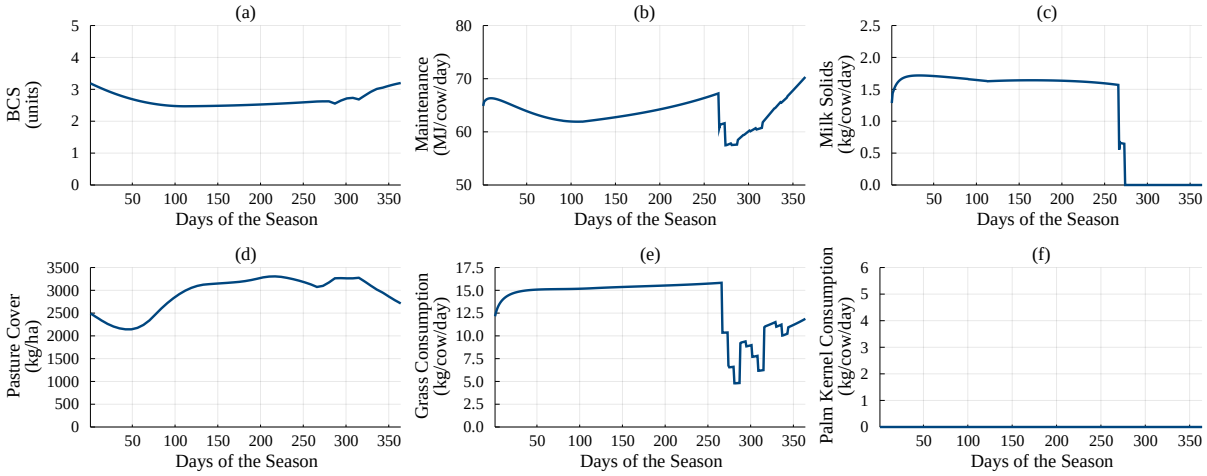


Figure D.1: Simulation of the "low intensity" policy: $(x_1^\rho, x_1^q) = (2.4, 0)$.

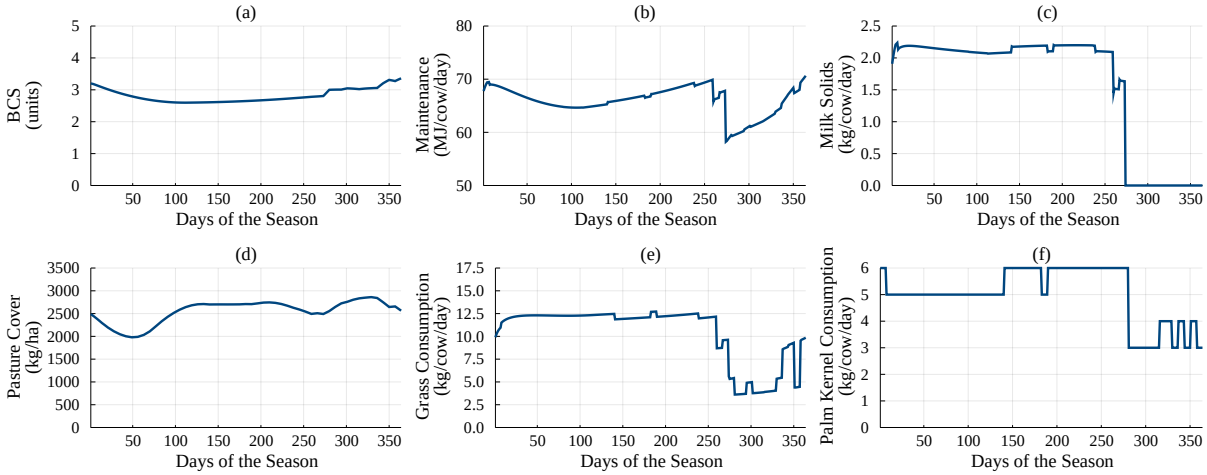


Figure D.2: Simulation of the "medium intensity" policy: $(x_1^\rho, x_1^q) = (3.2, 6000)$.

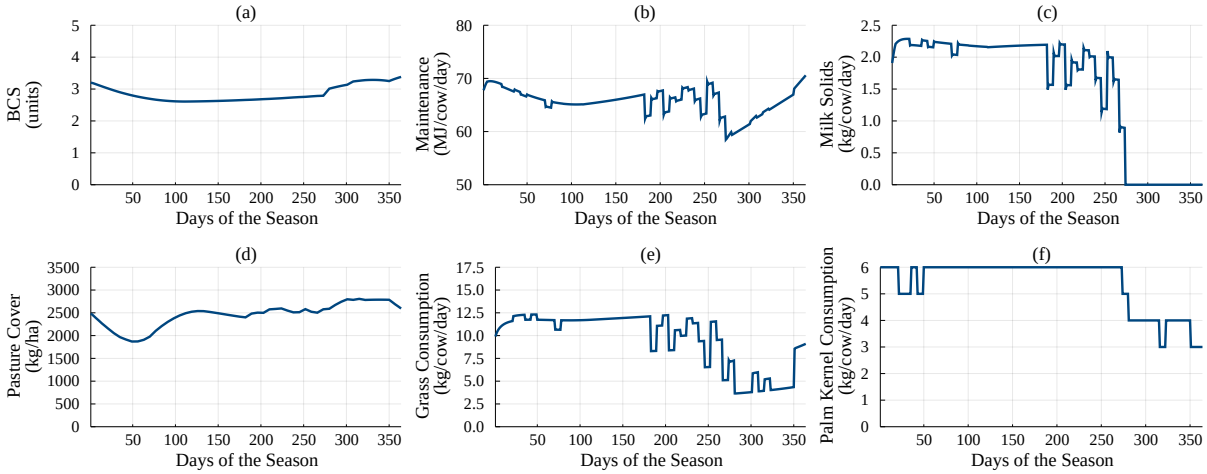


Figure D.3: Simulation of the "high intensity" policy: $(x_1^\rho, x_1^q) = (3.45, 7000)$.

Appendix E

Addendum for Chapter 8

In the following, we give the full formulation of the subproblem $\mathbf{SP}_{t,p_t}$ for stages $t = 1, 2, \ldots, 52$. However, compared to our description in Chapter 8, there are notable differences to: the pasture cover transition constraint, the actual evapotranspiration constraint, and the Fat Evaluation Index penalty. We elaborate on these differences below.

Pasture cover

First, recall the pasture cover transition constraint (Eq. 8.5):

$$P_{t+1} = P_t + \min\left\{\kappa_t \times e_t,\ 4\frac{\Delta P_{\max}}{P_{\max}} \times P_t\left(1 - \frac{P_t}{P_{max}}\right)\right\} - h_t - f_t^p,$$

where P_t is the pasture cover on the farm at the start of week t, κ_t is the soil fertility index in week t, e_t is the actual evapotranspiration during week t, $\Delta P_{\max}$ is the maximum rate of grass growth, $P_{\max}$ is the maximum pasture cover, h_t is the quantity of grass harvested in week t, and f_t^p is the quantity of grass from pasture fed to the cows during week t.

As formulated, this constraint has a $\min(\cdot,\cdot)$ function. Therefore, it cannot be directly solved as an LP. To overcome this issue, we introduce a grass growth rate variable g_t, and bound it from above by two less-than constraints so that:

$$g_t \leq \kappa_t \times e_t,$$

and

$$g_t \leq 7 \times \frac{4\Delta P_{\max}}{P_{\max}} \times P_t\left(1 - \frac{P_t}{P_{\max}}\right).$$

Note the multiplication by 7 to convert between the daily and weekly growth rates. The pasture cover transition constraint is now:

$$P_{t+1} = P_t + g_t - h_t - f_t^p.$$

However, the second less-than constraint is a concave, quadratic function. Therefore, we approximate it by the first-order Taylor series expansion:

$$g_t \leq g(\bar{P}_j) + g'(\bar{P}_j) \times (P_t - \bar{P}_j),$$

where

$$g(\bar{P}_j) = \frac{4\Delta P_{\max}}{P_{\max}} \times \bar{P}_j \times \left(1 - \frac{\bar{P}_j}{P_{\max}}\right),$$

and

$$g'(\bar{P}_j) = \frac{4\Delta P_{\max}}{P_{\max}} \times \left(1 - \frac{2\bar{P}_j}{P_{\max}}\right).$$

Note that the $4\Delta P_{\max}/P_{\max}$ term scales the growth rate so that $g(P_{\max}/2) = \Delta P_{\max}$. The approximated function is shown in Figure E.1.

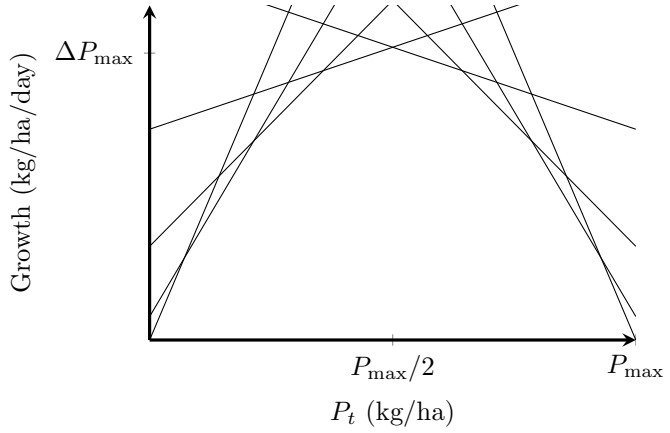


Figure E.1: Logistic pasture growth function approximated by a set of linear constraints.

Actual evapotranspiration

Second, recall the actual evapotranspiration constraint Eq. 8.3:

$$e_t = \min\left\{e_t^p,\ W_t + r_t\right\},$$

where e_t is the actual evapotranspiration, e_t^p is the potential evapotranspiration during week t, W_t is the soil moisture at the start of week t, and r_t is the incident rainfall during week t.

Like the pasture cover constraint above, this also has a $\min$ function. As above, we replace the $\min$ function by the two constraints:

$$\begin{aligned} &e_t \leq e_t^p, && \text{and} \\ &e_t \leq W_t + r_t. \end{aligned} \tag{E.1}$$

The Fat Evaluation Index penalty

Third, we gave a graphical description of the Fat Evaluation Index penalty (which we reproduce in Figure E.2), but we did not describe how this was implemented. Instead, we denoted the penalty that was incurred by Δ_t^q.

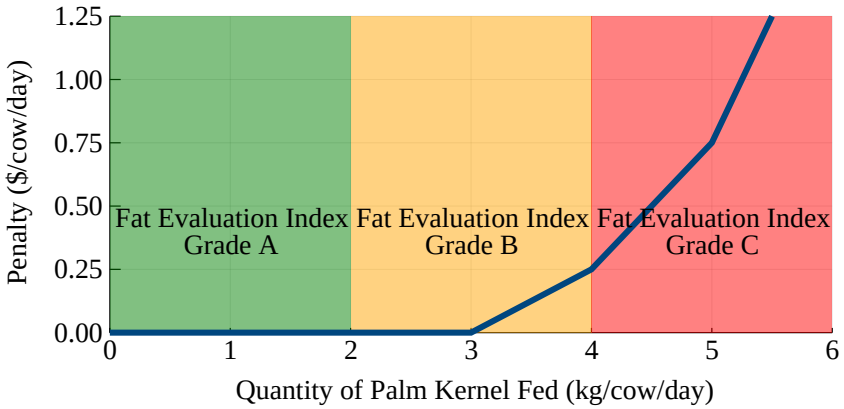


Figure E.2: Fat Evaluation Index Penalty for differing levels of intake. Shaded vertical bands represent different Fat Evaluation Index grades: A in green, B in orange, and C in red (D not shown).

By design, the Fat Evaluation Index penalty function is piecewise-linear and convex with respect to the quantity of supplement fed per cow per day. Therefore, we can approximate it from below by a collection of linear constraints.

Before we detail the linear constraints, recall that we denote the stocking rate (cows/ha) by ρ and the quantity of palm kernel to feed the herd each week by s_t. Therefore, the quantity that we feed each cow each day is $s_t/(7\rho)$.

We now detail the four segments of the piecewise linear penalty function and their corresponding constraints:

1. between 0 and 3 kg/cow/day, the penalty is \$0/kg/cow/day:

$$\Delta_t^q \geq 0;$$

2. between 3 and 4 kg/cow/day, the penalty increases at a rate of \$0.25/kg/cow/day:

$$\Delta_t^q \geq (7\rho) \times (0.00 + 0.25 \times (s_t/(7\rho) - 3));$$

3. between 4 and 5 kg/cow/day, the penalty increases at a rate of \$0.50/kg/cow/day:

$$\Delta_t^q \geq (7\rho) \times (0.25 + 0.50 \times (s_t/(7\rho) - 4)); \text{ and}$$

4. above 5 kg/cow/day, the penalty increases at a rate of \$1.00/kg/cow/day:

$$\Delta_t^q \geq (7\rho) \times (0.75 + 1.00 \times (s_t/(7\rho) - 5)).$$

The complete formulation

Incorporating the modifications above, the full formulation of the subproblem $\mathbf{SP}_{t,p_t}$ for stages $t = 1, 2, \ldots, 52$ is:

$$\mathbf{SP}_{t,p_t}: \; V_{t,p_t}(x_t, e_t^p, r_t) = \max \; \underbrace{\mathbb{E}\left[V_{t+1,p_{t+1}}(x_{t+1}, e_{t+1}^p, r_{t+1})\right]}_{\text{future profit}} - \underbrace{\left(c^s s_t + c^h h_t + c^i i_t + \Delta_t^q\right)}_{\text{week } t \text{ cost}}$$

s.t.

State Transitions:
$$C_{t+1} = C_t - u_t$$
$$M_{t+1} = M_t + \tfrac{m_t}{\eta_t^m}$$
$$P_{t+1} = P_t + g_t - h_t - f_t^p$$
$$Q_{t+1} = Q_t + \beta h_t - f_t^q$$
$$W_{t+1} \leq \bar{W}$$
$$W_{t+1} \leq W_t + r_t - e_t + i_t$$

Evapotranspiration:
$$e_t \leq e_t^p$$
$$e_t \leq W_t + r_t$$

Grass growth:
$$g_t \leq \kappa_t \times e_t$$
$$g_t \leq 7\,[g(0) + g'(0) \times (P_t - 0)]$$
$$g_t \leq 7\,[g(500) + g'(500) \times (P_t - 500)]$$
$$\vdots$$
$$g_t \leq 7\,[g(3500) + g'(3500) \times (P_t - 3500)]$$

Energy balance:
$$\eta^p \times (f_t^p + f_t^q) + \eta^s \times s_t = C_t \times \varepsilon_t^{\text{lac}} + \ldots (\bar{C} - C_t) \times \varepsilon_t^{\text{dry}} + m_t$$
$$\underline{\nu} C_t \leq m_t \leq \overline{\nu}_t C_t$$

Fat Evaluation Index Penalty:
$$\Delta_t^q \geq 0$$
$$\Delta_t^q \geq (7\rho) \times (0.00 + 0.25 \times (s_t/(7\rho) - 3))$$
$$\Delta_t^q \geq (7\rho) \times (0.25 + 0.50 \times (s_t/(7\rho) - 4))$$
$$\Delta_t^q \geq (7\rho) \times (0.75 + 1.00 \times (s_t/(7\rho) - 5))$$

Non-negativity of states:
$$C_{t+1}, M_{t+1}, P_{t+1}, Q_{t+1}, W_{t+1} \geq 0$$

Non-negativity of controls:
$$e_t, f_t^p, f_t^q, g_t, h_t, i_t, s_t, u_t \geq 0.$$

Appendix F

Addendum for Chapter 9

Consider a random milk price p_ω (\$/kg) for $\omega \in \Omega$, and denote the farmer's unhedged return in scenario ω by R_ω. If a farmer contracts h kg at the expected price $\mathbb{E}_\omega[p_\omega]$, then their hedged return is:

$$X_\omega = R_\omega + h(\underset{\omega}{\mathbb{E}}[p_\omega] - p_\omega).$$

In what follows, we show the hedge ratio h that minimises the variance of X_ω is the *Minimum Variance Hedge Ratio* [26] of the total unhedged return R using the hedge instrument p.

Theorem 2. *The hedge ratio h that minimises the variance of X_ω is cov(R, p)/var(p).*

Proof. Consider choosing a hedge quantity h to minimise the variance of X:

$$\min_h \left\{ \underset{\omega}{\mathbb{E}}[X_\omega^2] - \underset{\omega}{\mathbb{E}}[X_\omega]^2. \right\} \tag{F.1}$$

The expectation of X_ω is:

$$\begin{aligned} \underset{\omega}{\mathbb{E}}[X_\omega] &= \mathbb{E}_\omega[R_\omega + h(\mathbb{E}_\omega[p_\omega] - p_\omega)] \\ &= \mathbb{E}_\omega[R_\omega] + h\,\mathbb{E}_\omega[\omega] - h\,\mathbb{E}_\omega[p_\omega] \\ &= \mathbb{E}_\omega[R_\omega]. \end{aligned}$$

Expanding X^2 we get:

$$\begin{aligned} X^2 &= (R_\omega + h(\mathbb{E}_\omega[p_\omega] - p_\omega))^2 \\ &= R_\omega^2 + 2R_\omega h(\mathbb{E}_\omega[p_\omega] - p_\omega) + h^2(\mathbb{E}_\omega[p_\omega] - p_\omega)^2 \\ &= R_\omega^2 + 2\,\mathbb{E}_\omega[p_\omega]R_\omega h - 2R_\omega p_\omega h + \mathbb{E}_\omega[p_\omega]^2 h^2 - 2\,\mathbb{E}_\omega[p_\omega]p_\omega h^2 + p_\omega^2 h^2. \end{aligned}$$

Taking the expectation of X^2 we get:

$$\begin{aligned} \underset{\omega}{\mathbb{E}}[X^2] &= \mathbb{E}_\omega\left[R_\omega^2 + 2\,\mathbb{E}_\omega[p_\omega]R_\omega h - 2R_\omega p_\omega h + \mathbb{E}_\omega[p_\omega]^2 h^2 - 2\,\mathbb{E}_\omega[p_\omega]p_\omega h^2 + p_\omega^2 h^2\right] \\ &= \mathbb{E}_\omega[R_\omega^2] + 2\,\mathbb{E}_\omega[p_\omega]\,\mathbb{E}_\omega[R_\omega]h - 2\,\mathbb{E}_\omega[R_\omega p_\omega]h - \mathbb{E}_\omega[p_\omega]^2 h^2 + \mathbb{E}_\omega[p_\omega^2]h^2. \end{aligned}$$

Therefore Eq. F.1 is:

$$\min_h \left\{ h^2 \left(\mathop{\mathbb{E}}_{\omega}[p_\omega^2] - \mathop{\mathbb{E}}_{\omega}[p_\omega]^2 \right) + 2h \left(\mathop{\mathbb{E}}_{\omega}[p_\omega] \mathop{\mathbb{E}}_{\omega}[R_\omega] - \mathop{\mathbb{E}}_{\omega}[R_\omega p_\omega] \right) + \mathop{\mathbb{E}}_{\omega}[R_\omega^2] - (\mathop{\mathbb{E}}_{\omega}[R_\omega])^2 \right\}.$$

Differentiating w.r.t. h and setting to zero we get:

$$h \left(\mathop{\mathbb{E}}_{\omega}[p_\omega^2] - \mathop{\mathbb{E}}_{\omega}[p_\omega]^2 \right) + \left(\mathop{\mathbb{E}}_{\omega}[p_\omega] \mathop{\mathbb{E}}_{\omega}[R_\omega] - \mathop{\mathbb{E}}_{\omega}[R_\omega p_\omega] \right) = 0.$$

Re-arranging and using the identity $\text{var}(a) = \mathbb{E}[a^2] - \mathbb{E}[a]^2$, we get:

$$h = \frac{\mathbb{E}_\omega[R_\omega p_\omega] - \mathbb{E}_\omega[p_\omega]\,\mathbb{E}_\omega[R_\omega]}{\text{var}(p)}.$$

By the identity $\text{cov}(a, b) = \mathbb{E}[ab] - \mathbb{E}[a]\,\mathbb{E}[b]$, we get:

$$h = \frac{\text{cov}(R, p)}{\text{var}(p)}.$$

□

Bibliography

[1] A. Adler, G. Doole, A. Romera, and P. Beukes. Managing greenhouse gas emissions in two major dairy regions of New Zealand: A system-level evaluation. *Agricultural Systems*, 135:1–9, 2015.

[2] AgriHQ. Farm Gate Milk Price Calculator, 2016. URL `http://agrihq.co.nz/toolbox/farmgate-milk-price-calculator/`. [Online; accessed 2016-05-03].

[3] A. Alexandridis K. and A. D. Zapranis. *Weather Derivatives*. Springer New York, New York, NY, 2013.

[4] A. Alvarez, J. del Corral, D. Solís, and J. Pérez. Does Intensification Improve the Economic Efficiency of Dairy Farms? *Journal of Dairy Science*, 91(9):3693–3698, 2008.

[5] ANZ. Commodity Price Index, 2018. URL `http://www.anz.co.nz/about/media/library/cpi/Indexforpublication.xlsx`. [Online; accessed 2018-03-19].

[6] P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath. Coherent measures of risk. *Mathematical Finance*, 9(3):203–228, 1999.

[7] J. Aryal, D. Kulasiri, and D. Liu. Optimization Approaches for a Complex Dairy Farm Simulation Model. *International Journal of Computer, Information, and Systems Science, and Engineering*, 2(3):156–162, 2008.

[8] T. Asamov and W. B. Powell. Regularized Decomposition of High-Dimensional Multistage Stochastic Programs with Markov Uncertainty. *arXiv preprint arXiv:1505.02227*, 2015. URL `http://arxiv.org/abs/1505.02227`.

[9] E. Balas, M. Fischetti, and A. Zanette. On the enumerative nature of Gomory's dual cutting plane method. *Mathematical Programming*, 125(2):325–351, 2010.

[10] J. Ballingal and D. Pambudi. Dairy trade's economic contribution to New Zealand. Technical report, NZIER, 2017. URL `https://nzier.org.nz/publication/dairy-trades-economic-contribution-to-new-zealand`.

[11] M. Bandarra and V. Guigues. Multicut decomposition methods with cut selection for multistage stochastic programs. *arXiv preprint arXiv:1705.08977*, 2017. URL `https://arxiv.org/abs/1705.08977`.

[12] R. Baucke. An algorithm for solving infinite horizon Markov dynamic programmes. *Optimization Online*, 2018. URL `http://www.optimization-online.org/DB_HTML/2018/04/6565.html`.

[13] R. Baucke, A. Downward, and G. Zakeri. A deterministic algorithm for solving multistage stochastic programming problems. *Optimization Online*, 2017. URL `http://www.optimization-online.org/DB_FILE/2017/07/6138.pdf`.

[14] R. Baucke, A. Downward, and G. Zakeri. A deterministic algorithm for solving multistage stochastic minimax dynamic programmes. *Optimization Online*, 2018. URL `http://www.optimization-online.org/DB_FILE/2018/02/6449.pdf`.

[15] J. Baudracco. *Effects of Feeding Level and Genetic Merit on the Efficiency of Pasture-Based Dairy Systems: Field and Modelling Studies*. PhD thesis, Massey University, Palmerston North, New Zealand, 2011.

[16] J. Baudracco, N. Lopez-Villalobos, and M. Zamateo. E-Cow: Simulation of a dairy cow's response, 2008. URL `http://e-cow.net`. [Online; accessed 2014-07-22].

[17] J. Baudracco, N. Lopez-Villalobos, C. Holmes, and K. Macdonald. Prediction of herbage dry matter intake for dairy cows grazing ryegrass-based pastures. *Proceedings of the New Zealand society of Animal Production*, 70:80–85, 2010.

[18] J. Baudracco, N. Lopez-Villalobos, C. Holmes, E. Comeron, K. Macdonald, T. Barry, and N. Friggens. E-Cow: An animal model that predicts herbage intake, milk yield and live weight change in dairy cows grazing temperate pastures with and without supplementary feeding. *Animal*, 6(6):980–993, 2011.

[19] J. Baudracco, N. Lopez-Villalobos, C. Holmes, E. Comeron, K. Macdonald, T. Barry, and N. Friggens. E-Dairy: A dynamic and stochastic whole-farm model that predicts biophysical and economic performance of grazing dairy systems. *Animal*, 7(5):870–878, 2012.

[20] G. Bayraksan and D. P. Morton. A Sequential Sampling Procedure for Stochastic Programming. *Operations Research*, 59(4):898–913, 2011.

[21] G. Bayraksan, D. P. Morton, and A. Partani. Simulation-based optimality tests for stochastic programs. In G. Infanger, editor, *Stochastic Programming*, number 150 in International series in operations research & management science, pages 37–55. Springer, New York, 2011.

[22] R. Bellman. The Theory of Dynamic Programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[23] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[24] R. Bellman and S. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, Princeton, NJ, 1962.

[25] J. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[26] S. Benninga, R. Eldor, and I. Zilcha. The optimal hedge ratio in unbiased futures markets. *Journal of Futures Markets*, 4(2):155–159, 1984.

[27] D. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, Bellmont, MA, third edition, 2005.

[28] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, 2017.

[29] J. R. Birge and F. Louveaux. A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research*, 34(3):384–392, 1988.

[30] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research and Financial Engineering. Springer New York, New York, NY, 2011.

[31] J. R. Birge, M. A. Dempster, H. I. Gassmann, E. Gunn, A. J. King, and S. W. Wallace. A standard input format for multiperiod stochastic linear programs. IIASA Working Paper, WP-87-118, IIASA, Laxenburg, Austria, 1987.

[32] V. Borodin, J. Bourtembourg, F. Hnaien, and N. Labadie. Handling uncertainty in agricultural supply chain management: A state of the art. *European Journal of Operational Research*, 254(2):348–359, 2016.

[33] R. Brazendale. Do supplements reduce risk? *The Official Journal of the New Zealand Institute of Primary Industry Management Incorporated*, 13(3):13–14, 2009.

[34] R. Brealy, S. Myers, and F. Allen. *Principles of Corporate Finance*. McGraw-Hill Education, New York, NY, twelfth edition, 2017.

[35] J. Bryant. *Quantifying Genetic Variation in Environmental Sensitivity of New Zealand Dairy Cattle to Apply in the Development of a Dairy Cattle Simulation Model for Pastoral Systems*. PhD thesis, Massey University, Palmerston North, New Zealand, 2006.

[36] J. Bryant and V. Snow. Modelling pastoral farm agro-ecosystems: A review. *New Zealand Journal of Agricultural Research*, 51(3):349–363, 2008.

[37] M. R. Bussieck, M. C. Ferris, and T. Lohmann. GUSS: Solving collections of data related models within GAMS. In J. Kallrath, editor, *Algebraic Modeling Systems: Modeling and Solving Real World Optimization Problems*, pages 35–56. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[38] L. Cambier and D. Scieur. FAST, 2018. URL `https://web.stanford.edu/~lcambier/fast/`. [Online; accessed 2018-04-05].

[39] G. Cann. The great water debate: What impact would a charge have? *Stuff*, Aug. 2017. URL `https://www.stuff.co.nz/national/politics/96230227/water-policy-will-have-an-effect-on-expensive-cabbages-and-wine`. [Online; accessed 2018-03-19].

[40] J. Chakravarti. *Agricultural Insurance: A Practical Scheme Suited to Indian Conditions*. Government Press, Bangalore, India, 1920.

[41] G. Chen, M. C. Roberts, and C. S. Thraen. Managing dairy profit risk using weather derivatives. *Journal of Agricultural and Resource Economics*, pages 653–666, 2006.

[42] Z. L. Chen and W. B. Powell. Convergent Cutting-Plane and Partial-Sampling Algorithm for Multistage Stochastic Linear Programs with Recourse. *Journal of Optimization Theory and Applications*, 102(3):497–524, 1999.

[43] COIN-OR. Clp (Coin-or Linear Programming), 2016. URL `https://projects.coin-or.org/Clp`.

[44] Dairy NZ. Palm Kernel Extract. Farm Fact 1-71, Dairy NZ, 2008.

[45] Dairy NZ. *Dairy NZ Economic Survey 2007-2008*. Dairy NZ, 2009.

[46] Dairy NZ. Drying off. Technote 16, Dairy NZ, 2012.

[47] Dairy NZ. *Facts and Figures*. Dairy NZ, 2 edition, 2012.

[48] Dairy NZ. The 5 Production Systems, 2014. URL `http://www.dairynz.co.nz/farm/farm-systems/the-5-production-systems/`. [Online; accessed 2014-04-02].

[49] Dairy NZ. *Dairy NZ Economic Survey 2014-15*. Dairy NZ, 2016.

[50] Dairy NZ. DairyBase, 2017. URL `https://www.dairynz.co.nz/business/dairybase/`. [Online; accessed 2017-10-17].

[51] Dairy NZ. Palm Kernel Extract, 2017. URL `https://www.dairynz.co.nz/feed/supplements/palm-kernel-extract-pke/`. [Online; accessed 2017-10-27].

[52] Dairy NZ and LIC. *New Zealand Dairy Statistics 2015-16*. Dairy NZ, 2016.

[53] G. Dantzig. Linear Programming. In J. Lenstra, A. Rinnoy Kan, and A. Schrijver, editors, *History of Mathematical Programming - A Collection of Personal Reminiscences*. Elsevier Science Publishers, Amsterdam, The Netherlands, 1991.

[54] G. B. Dantzig. Linear Programming Under Uncertainty. *Management Science*, 1:197–206, 1955.

[55] V. L. de Matos, A. B. Philpott, and E. C. Finardi. Improving the performance of Stochastic Dual Dynamic Programming. *Journal of Computational and Applied Mathematics*, 290:196–208, 2015.

[56] V. L. de Matos, D. P. Morton, and E. C. Finardi. Assessing policy quality in a multistage stochastic program for long-term hydrothermal scheduling. *Annals of Operations Research*, 253(2):713–731, 2017.

[57] X. Deng, B. J. Barnett, D. V. Vedenov, and J. W. West. Hedging dairy production losses using weather-based index insurance. *Agricultural Economics*, 36(2):271–280, 2007.

[58] C. J. Donohue and J. R. Birge. The abridged nested decomposition method for multistage stochastic linear programs with relatively complete recourse. *Algorithmic Operations Research*, 1(1), 2006.

[59] G. Doole. Economic feasibility of supplementary feeding on dairy farms in the Waikato region of New Zealand. *New Zealand Journal of Agricultural Research*, 57(2):90–99, 2014.

[60] G. Doole. Least-cost greenhouse gas mitigation on New Zealand dairy farms. *Nutrient Cycling in Agroecosystems*, 98(2):235–251, 2014.

[61] G. Doole. Improving the profitability of Waikato dairy farms: Insights from a whole-farm optimisation model. *New Zealand Economic Papers*, 49(1):44–61, 2015.

[62] G. Doole and A. Romera. Detailed description of grazing systems using nonlinear optimisation methods: A model of a pasture-based New Zealand dairy farm. *Agricultural Systems*, 122:33–41, 2013.

[63] G. Doole, A. Romera, and A. Adler. A Mathematical Optimisation Model of a New Zealand Dairy Farm: The Integrated Dairy Enterprise (IDEA) Framework. Working Paper in Economics, Department of Economics, Waikato University, Waikato, New Zealand, 2012. URL `http://researchcommons.waikato.ac.nz/handle/10289/6699`.

[64] G. Doole, A. Romera, and A. Adler. An optimization model of a New Zealand dairy farm. *Journal of Dairy Science*, 96(4):2147–2160, 2013.

[65] A. Downward, O. Dowson, and R. Baucke. On the convergence of a cutting plane method with price uncertainty. *Optimization Online*, 2018. URL `http://www.optimization-online.org/DB_HTML/2018/02/6454.html`.

[66] I. Dunning, J. Huchette, and M. Lubin. JuMP: A Modeling Language for Mathematical Optimization. *SIAM Review*, 59(2):295–320, 2017.

[67] S. Edmunds. Stress too much for farmers. *NZ Herald*, Jan. 2015. URL `http://www.nzherald.co.nz/business/news/article.cfm?c_id=3&objectid=11384751`. [Online; accessed 2018-04-03].

[68] D. Espinoza. Avoiding Numerical Issues in Optimization Models, 2018. URL `https://www.gurobi.com/resources/seminars-and-videos/numerical-issues-webinar`. [Online; accessed 2018-04-03].

[69] European Commission. The end of milk quotas, 2015. URL `https://ec.europa.eu/agriculture/milk-quota-end_en`. [Online; accessed 2018-03-19].

[70] E. Fama. The Behavior of Stock-Market Prices. *The Journal of Business*, 38(1):34–105, 1965.

[71] S. Fariña, A. Alford, S. Garcia, and W. Fulkerson. An integrated assessment of business risk for pasture-based dairy farm systems intensification. *Agricultural Systems*, 115:10–20, 2013.

[72] FARMAX. Pasture Growth Forecaster: Long Term Forecast, 2016. URL `https://apps.farmax.co.nz/pasture/Dairy/ProjectFarms/LongTermForecast`. [Online; accessed 2018-04-03].

[73] G. Feola, C. Sattler, and A. K. Saysel. Simulation models in Farming Systems Research: Potential and challenges. In I. Darnhofer, D. Gibbon, and B. Dedieu, editors, *Farming Systems Research into the 21st Century: The New Dynamic*, pages 281–306. Springer Netherlands, 2012.

[74] O. Flaten and G. Lien. Stochastic utility-efficient programming of organic dairy farms. *European Journal of Operational Research*, 181(3):1574–1583, 2007.

[75] Fonterra. Farmgate Milk Price Statement, 2017. URL `http://www2.fonterra.com/our-financials/milk-price-methodology`.

[76] Fonterra. Farmgate Milk Prices, 2018. URL `https://www.fonterra.com/nz/en/our-financials/farmgate-milk-prices/milk-price-methodology.html`. [Online; accessed 2018-05-10].

[77] M. Freer, H. Dove, and J. Nolan. *Nutrient Requirements of Domesticated Ruminants*. CSIRO Publishing, 2007.

[78] N. Friggens, K. Ingvartsen, and G. Emmans. Prediction of Body Lipid Change in Pregnancy and Lactation. *Journal of Dairy Science*, 87(4):988–1000, 2004.

[79] P. Fusaro and G. Vasey. *Energy and Environmental Hedge Funds: The New Investment Paradigm*. John Wiley & Sons, Singapore, 2006.

[80] S. Garcia. *Systems, Component, and Modelling Studies of Pasture-Based Dairy Systems in Which the Cows Calve at Different Times of the Year.* PhD Thesis, Massey University, Palmerston North, New Zealand, 2000.

[81] H. I. Gassmann and B. Kristjansson. The SMPS format explained. *IMA Journal of Management Mathematics*, 19(4):347–377, 2007.

[82] H. U. Gerber and G. Pafum. Utility Functions: From Risk Theory to Finance. *North American Actuarial Journal*, 2(3):74–91, 1998.

[83] H. Gevret, N. Langrené, J. Lelong, and X. Warin. STochastic OPTimization library in C++. Technical report, EDF, 2016. URL `https://hal.archives-ouvertes.fr/hal-01361291v5/document`.

[84] P. Girardeau, V. Leclère, and A. B. Philpott. On the Convergence of Decomposition Methods for Multistage Stochastic Convex Programs. *Mathematics of Operations Research*, 40(1):130–145, 2015.

[85] A. Gjelsvik, M. Belsnes, and A. Haugstad. An algorithm for stochastic medium term hydro thermal scheduling under spot price uncertainty. In *PSCC: 13th Power Systems Computation Conference : Proceedings*, pages 1079–1085, Trondheim, 1999. Executive Board of the 13th Power Systems Computation Conference, 1999.

[86] A. Gjelsvik, B. Mo, and A. Haugstad. Long- and Medium-term Operations Planning and Stochastic Modelling. In P. M. Pardalos, S. Rebennack, M. V. F. Pereira, and N. A. Iliadis, editors, *Handbook of Power Systems I*, Energy Systems, pages 33–55. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[87] GlobalDairyTrade. About Us, 2018. URL `http://www.globaldairytrade.info/en/about-us/about-us/`. [Online; accessed 2018-03-19].

[88] P. Gregorini, P. Beukes, A. Romera, G. Levy, and M. Hanigan. A model of diurnal grazing patterns and herbage intake of a dairy cow, MINDY: Model description. *Ecological Modelling*, 270:11–29, 2013.

[89] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, Oxford, second edition, 1992.

[90] J. Groot, G. Oomen, and W. Rossing. Multi-objective optimization and design of farming systems. *Agricultural Systems*, 110:63–77, 2012.

[91] Z. Guan. *Strategic Inventory Models for International Dairy Commodity Markets*. PhD Thesis, University of Auckland, Auckland, New Zealand, 2008.

[92] Z. Guan and A. Philpott. A multistage stochastic programming model for the New Zealand dairy industry. *International Journal of Production Economics*, 134(2):289–299, 2011.

[93] V. Guigues. Convergence Analysis of Sampling-Based Decomposition Methods for Risk-Averse Multistage Stochastic Convex Programs. *SIAM Journal on Optimization*, 26(4): 2468–2494, 2016.

[94] V. Guigues. Inexact cuts in Deterministic and Stochastic Dual Dynamic Programming applied to linear optimization problems. *Optimization Online*, 2018. URL `http://www.optimization-online.org/DB_HTML/2018/01/6426.html`.

[95] V. Guigues. Multistage stochastic programs with a random number of stages: Dynamic programming equations, solution methods, and application to portfolio selection. *Optimization Online*, 2018. URL `http://www.optimization-online.org/DB_HTML/2018/03/6530.html`.

[96] V. Guigues and M. Lejeune. Regularized Decomposition Methods for Deterministic and Stochastic Convex Optimization and Application to Portfolio Selection with Direct Transaction and Market Impact Costs. *SSRN Electronic Journal*, 2017. URL `http://www.ssrn.com/abstract=2899448`.

[97] Gurobi Optimization. Gurobi 6.5 Reference Manual. Technical report, Gurobi Optimization, 2016. URL `http://www.gurobi.com/documentation/6.5/refman/index.html`.

[98] Gurobi Optimization. Gurobi 7.0 Reference Manual. Technical report, Gurobi Optimization, 2017. URL `http://www.gurobi.com/documentation/7.0/refman/index.html`.

[99] Gurobi Optimization. Gurobi 7.5 Performance Benchmarks. Technical report, Gurobi Optimization, 2017. URL `https://gurobi.com/pdfs/benchmarks.pdf`.

[100] M. T. Harrison, B. R. Cullen, and D. Armstrong. Management options for dairy farms under climate change: Effects of intensification, adaptation and simplification on pastures, milk production and profitability. *Agricultural Systems*, 155:19–32, 2017.

[101] R. Hart, M. Larcombe, R. Sherlock, and L. Smith. Optimisation techniques for a computer simulation of a pastoral dairy farm. *Computers and Electronics in Agriculture*, 19 (2):129–153, 1998.

[102] T. Heikkinen and K. Pietola. Investment and the dynamic cost of income uncertainty: The case of diminishing expectations in agriculture. *European Journal of Operational Research*, 192(2):634–646, 2009.

[103] A. Helseth and H. Braaten. Efficient Parallelization of the Stochastic Dual Dynamic Programming Algorithm Applied to Hydropower Scheduling. *Energies*, 8(12):14287–14297, 2015.

[104] M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, Burlington, MA, rev. ed. edition, 2012.

[105] M. J. Hill, G. E. Donald, P. J. Vickery, A. D. Moore, and J. R. Donnelly. Combining satellite data with a simulation model to describe spatial variability in pasture growth at a farm scale. *Australian Journal of Experimental Agriculture*, 39(3):285–300, 1999.

[106] M. J. Hill, G. E. Donald, M. W. Hyder, and R. C. Smith. Estimation of pasture growth rate in the south west of Western Australia from AVHRR NDVI and climate data. *Remote Sensing of Environment*, 93(4):528–545, 2004.

[107] M. Hindsberger. ReSa: A method for solving multistage stochastic linear programs. *Journal of Applied Operational Research*, 6(1):2–15, 2014.

[108] T. Homem-de Mello and G. Bayraksan. Stochastic Constraints and Variance Reduction. In *Handbook of Simulation Optimization*, number 216 in International Series in Operations Research and Management Science, pages 245–276. Springer New York, New York, 2015.

[109] T. Homem-de Mello and B. K. Pagnoncelli. Risk aversion in multistage stochastic programming: A modeling and algorithmic perspective. *European Journal of Operational Research*, 249(1):188–199, 2016.

[110] T. Homem-de Mello, V. L. de Matos, and E. C. Finardi. Sampling strategies and stopping criteria for stochastic dual dynamic programming: A case study in long-term hydrothermal scheduling. *Energy Systems*, 2(1):1–31, 2011.

[111] R. S. Hudson and A. Gregoriou. Calculating and comparing security returns is harder than you think: A comparison between logarithmic and simple returns. *International Review of Financial Analysis*, 38:151–162, 2015.

[112] IBM Corp. IBM ILOG CPLEX Optimization Studio CPLEX User's Manual Version 12 Release 6, 2014.

[113] I. Johnson. *DairyMod and the SGS Pasture Model: A Mathematical Description of the Biophysical Model Structure*. IMJ Consultants, Dorrigo, NSW, Australia, 2013.

[114] I. R. Johnson, J. France, J. H. M. Thornley, M. J. Bell, and R. J. Eckard. A generic model of growth, energy metabolism, and body composition for cattle and sheep. *Journal of Animal Science*, 90(13):4741–4751, 2012.

[115] L. Kapelevich. *DOASA in Julia*. Honours Thesis, The University of Auckland, Auckland, New Zealand, 2016.

[116] L. Kapelevich. SDDiP.jl: SDDP extension for integer local or state variables, 2018. URL `https://github.com/lkapelevich/SDDiP.jl`. [Online; accessed 2018-01-19].

[117] R. Kellogg. The Effect of Uncertainty on Investment: Evidence from Texas Oil Drilling. *American Economic Review*, 104(6):1698–1734, 2014.

[118] E. Klotz. Identification, Assessment, and Correction of Ill-Conditioning and Numerical Instability in Linear and Integer Programs. In A. M. Newman, J. Leung, J. C. Smith, and H. J. Greenberg, editors, *Bridging Data and Decisions*, pages 54–108. INFORMS, 2014.

[119] V. Knips. Developing Countries and the Global Dairy Sector. Part I: Global Overview. PPLPI Working Paper 30, Food and Agriculture Organization, Rome, Italy, 2005.

[120] V. Kozmík and D. Morton. Risk-averse stochastic dual dynamic programming. *Optimization Online*, 2013. URL `http://www.optimization-online.org/DB_FILE/2013/02/3794.pdf`.

[121] V. Kozmík and D. P. Morton. Evaluating policies in risk-averse multi-stage stochastic programming. *Mathematical Programming*, 152(1-2):275–300, 2015.

[122] T. Kristensen, J. Sorensen, and S. Clausen. Simulated effect on dairy cow and herd production of different grazing intensities. *Agricultral Systems*, 55:123–138, 1997.

[123] M. Larcombe. *The Effects of Manipulating Reproduction on the Productivity and Profitability of Dairy Herds Which Graze Pasture*. PhD Thesis, University of Melbourne, Australia, 1989. URL `https://minerva-access.unimelb.edu.au/handle/11343/35569`.

[124] V. Leclère, P. Carpentier, J.-P. Chancelier, A. Lenoir, and F. Pacaud. Exact converging bounds for Stochastic Dual Dynamic Programming via Fenchel duality. *Optimization Online*, 2018. URL `http://www.optimization-online.org/DB_FILE/2018/04/6575.pdf`.

[125] V. Leclère, F. Pacaud, T. Rigaut, and H. Gerard. StochDynamicProgramming.jl, 2018. URL `https://github.com/JuliaOpt/StochDynamicProgramming.jl`. [Online; accessed 2018-04-05].

[126] B. Legat. StructDualDynProg.jl, 2018. URL `https://github.com/blegat/StructDualDynProg.jl`. [Online; accessed 2018-04-05].

[127] K. Linowsky and A. B. Philpott. On the Convergence of Sampling-Based Decomposition Algorithms for Multistage Stochastic Programs. *Journal of Optimization Theory and Applications*, 125(2):349–366, 2005.

[128] P. Liu, X. Cai, and S. Guo. Deriving multiple near-optimal solutions to deterministic reservoir operation problems. *Water Resources Research*, 47(8):1–20, 2011.

[129] N. Löhndorf, D. Wozabal, and S. Minner. Optimizing Trading Decisions for Hydro Storage Systems Using Approximate Dual Dynamic Programming. *Operations Research*, 61 (4):810–823, 2013.

[130] F. Louveaux. A Solution Method for Multistage Stochastic Programs with Recourse with Application to an Energy Investment Problem. *Operations Research*, 28(4):889–902, 1980.

[131] M. Lubin and I. Dunning. Computing in Operations Research Using Julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.

[132] K. Macdonald, J. Penno, P. Nicholas, J. Lile, M. Coulter, and J. Lancaster. Farm systems – Impact of stocking rate on dairy farm efficiency. *Proceedings of the New Zealand Grassland Association*, 63:223–227, 2001.

[133] K. Macdonald, J. Penno, J. Lancaster, and J. Roche. Effect of Stocking Rate on Pasture Production, Milk Production and Reproduction of Dairy Cows in Pasture-Based Systems. *Journal of Dairy Science*, 91(5):2151–2163, 2008.

[134] K. Macdonald, D. Beca, J. Penno, J. Lancaster, and J. Roche. Effect of stocking rate on the economics of pasture-based dairy farms. *Journal of Dairy Science*, 94(5):2581–2586, 2011.

[135] M. E. P. Maceira, V. S. Duarte, D. D. J. Penna, L. A. M. Moraes, and A. C. G. Melo. Ten years of application of stochastic dual dynamic programming in official and agent studies in Brazil: Description of the NEWAVE program. *16th PSCC, Glasgow, Scotland*, pages 14–18, 2008. URL `http://pscc.ee.ethz.ch/uploads/tx_ethpublications/pscc2008_704.pdf`.

[136] M. Miltenberger, T. Ralphs, and D. Steffy. Exploring the Numerics of Branch-and-Cut for Mixed Integer Linear Optimization. ZIB Report 17/43, Zuse Institute Berlin, Berlin, Germany, 2017.

[137] P. K. Mishra. Is rainfall insurance a new idea? Pioneering work revisited. *Economic and Political Weekly*, pages A84–A88, 1995.

[138] J. L. Moir, D. R. Scotter, M. J. Hedley, and A. D. Mackay. A climate-driven, soil fertility dependent, pasture production model. *New Zealand Journal of Agricultural Research*, 43(4):491–500, 2000.

[139] D. P. Morton and G. Infanger. Cut sharing for multistage stochastic linear programs with interstage dependency. *Mathematical Programming*, 75:241–256, 1996.

[140] B. Murtagh. *Advanced Linear Programming: Computation and Practice*. McGraw-Hill International Book Co., New York, 1981.

[141] D. Nadolnyak and D. Vedenov. Information value of climate forecasts for rainfall index insurance for pasture, rangeland, and forage in the Southeast United States. *Journal of Agricultural and Applied Economics*, 45(1):109–124, 2013.

[142] G. Nannicini, E. Traversi, and R. W. Calvo. A Benders squared (B2) framework for infinite-horizon stochastic linear programs. *Optimization Online*, 2017. URL `http://www.optimization-online.org/DB_HTML/2017/06/6101.html`.

[143] M. Neal, R. Drynan, W. Fulkerson, G. Levy, M. Wastney, E. Post, B. Thorrold, C. Palliser, P. Beukes, and C. Folkers. Optimisation of a whole farm model. In *Proceedings of the Australian Agricultural and Resource Economics Society (AARES) Conference. Coffs Harbour, NSW, Australia*, pages 1–29, 2005.

[144] New Zealand Foreign Affairs and Trade. NZ-China FTA upgrade, 2018. URL https://www.mfat.govt.nz/en/trade/free-trade-agreements/free-trade-agreements-in-force/nz-china-free-trade-agreement. [Online; accessed 2018-03-19].

[145] N. Newham. *Power System Investment Planning Using Stochastic Dual Dynamic Programming*. PhD thesis, University of Canterbury, Christchurch, New Zealand, 2008.

[146] NIWA. CliFlo: NIWA's National Climate Database on the Web, 2017. URL http://cliflo.niwa.co.nz. [Online; accessed 2017-10-25].

[147] NIWA. El Niño and La Niña, 2018. URL https://www.niwa.co.nz/climate/information-and-resources/elnino. [Online; accessed 2018-04-03].

[148] NZX. NZX Dairy Derivatives, 2018. URL http://www.nzxfutures.com/dairy. [Online; accessed 2018-04-03].

[149] K. I. Ourani, C. G. Baslis, and A. G. Bakirtzis. A Stochastic Dual Dynamic Programming model for medium-term hydrothermal scheduling in Greece. In *Universities Power Engineering Conference (UPEC), 2012 47th International*, pages 1–6. IEEE, 2012.

[150] A. Papavasiliou. Stochastic Dual Dynamic Programming, 2017. URL https://perso.uclouvain.be/anthony.papavasiliou/public_html/SDDP.pdf. [Online; accessed 2018-04-03].

[151] P. Parpas, B. Ustun, M. Webster, and Q. K. Tran. Importance sampling in stochastic programming: A Markov chain Monte Carlo approach. *INFORMS Journal on Computing*, 27(2):358–377, 2015.

[152] R. Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011.

[153] M. Pereira and L. Pinto. Multi-stage stochastic optimization applied to energy planning. *Mathematical Programming*, 52:359–375, 1991.

[154] L. Pfeiffer, R. Apparigliato, and S. Auchapt. Two methods of pruning Benders' cuts and their application to the management of a gas portfolio. *Optimization Online*, 2012. URL http://www.optimization-online.org/DB_FILE/2012/11/3683.pdf.

[155] G. C. Pflug and A. Pichler. Time-Consistent Decisions and Temporal Decomposition of Coherent Risk Functionals. *Mathematics of Operations Research*, 41(2):682–699, 2016.

[156] G. C. Pflug and A. Pichler. Time-inconsistent multistage stochastic programs: Martingale bounds. *European Journal of Operational Research*, 249(1):155–163, 2016.

[157] A. Philpott. On the Marginal Value of Water for Hydroelectricity. In T. Terlaky, M. F. Anjos, and S. Ahmed, editors, *Advances and Trends in Optimization with Engineering Applications*, pages 405–425. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017.

[158] A. Philpott and G. Pritchard. EMI-DOASA. Technical report, Electric Power Optimization Centre, 2013. URL http://www.emi.ea.govt.nz/Content/Tools/Doasa/DOASA%20paper%20by%20SOL.pdf.

[159] A. Philpott, V. de Matos, and E. Finardi. On Solving Multistage Stochastic Programs with Coherent Risk Measures. *Operations Research*, 61(4):957–970, 2013.

[160] A. B. Philpott and V. L. de Matos. Dynamic sampling algorithms for multi-stage stochastic programs with risk aversion. *European Journal of Operational Research*, 218(2): 470–483, 2012.

[161] A. B. Philpott and Z. Guan. On the convergence of sampling-based methods for multi-stage stochastic linear programs. *Operations Research Letters*, 36:450–455, 2008.

[162] A. B. Philpott, V. de Matos, and L. Kapelevich. Distributionally Robust SDDP. Technical report, Electric Power Optimization Centre, Auckland, New Zealand, 2017. URL `http://www.epoc.org.nz/papers/DROPaperv52.pdf`.

[163] R. J. Pinto, C. T. Borges, and M. E. P. Maceira. An Efficient Parallel Algorithm for Large Scale Hydrothermal System Operation Planning. *IEEE Transactions on Power Systems*, 28(4):4888–4896, 2013.

[164] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley series in probability and statistics. Wiley, Hoboken, N.J, 2nd ed edition, 2011.

[165] W. B. Powell. Clearing the Jungle of Stochastic Optimization. In A. M. Newman, J. Leung, and J. C. Smith, editors, *Bridging Data and Decisions*, TutORials in Operations Research, pages 109–137. INFORMS, 2014.

[166] W. B. Powell. A Unified Framework for Optimization under Uncertainty. In A. Gupta, A. Capponi, and J. C. Smith, editors, *Optimization Challenges in Complex, Networked and Risky Systems*, TutORials in Operations Research, pages 45–83. INFORMS, 2016.

[167] C. Priestley and R. Taylor. On the assessment of surface heat flux and evaporation using large-scale parameters. *Monthly Weather Review*, 100:81–92, 1972.

[168] PSR. Software | PSR, 2016. URL `http://www.psr-inc.com/softwares-en/`. [Online; accessed 2018-04-03].

[169] M. Puterman. *Markov Decision Processes*. Wiley series in probability and statistics. John Wiley & Sons, New York, NY, 1994.

[170] Python Software Foundation. Python, 2017. Version 2.7.12.

[171] Quantego. QUASAR, 2018. URL `http://quantego.com/`. [Online; accessed 2018-04-05].

[172] R. Rahmaniani, T. G. Crainic, M. Gendreau, and W. Rei. The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017.

[173] S. Rebennack. Combining sampling-based and scenario-based nested Benders decomposition methods: Application to stochastic dual dynamic programming. *Mathematical Programming*, 156(1-2):343–389, 2016.

[174] L. Relund Nielsen, E. Jørgensen, and S. Højsgaard. Embedding a state space model into a Markov decision process. *Annals of Operations Research*, 190(1):289–309, 2011.

[175] D. Rickard, S. McBride, and P. Fitzgerald. The effect of soil moisture deficits on pasture yield. *NZ Agricultural Science*, 20(1):7–12, 1986.

[176] F. Riedel. Dynamic coherent risk measures. *Stochastic Processes and their Applications*, 112(2):185–200, 2004.

[177] J. Roche, N. Friggens, J. Kay, M. Fisher, K. Stafford, and D. Berry. Body condition score and its association with dairy cow productivity, health and welfare. *Journal of Dairy Science*, 96(12):5769–5801, 2009.

[178] R. T. Rockafellar and R. J.-B. Wets. Scenarios and Policy Aggregation in Optimization Under Uncertainty. *Mathematics of Operations Research*, 16(1):119–147, 1991.

[179] T. R. Rockafellar and S. P. Uryasev. Conditional Value-at-Risk for General Loss Distributions. *Journal of Banking and Finance*, 26:1443–1471, 2002.

[180] A. Romera and G. Doole. Optimising the interrelationship between intake per cow and intake per hectare. *Animal Production Science*, 55(3):384, 2015.

[181] A. Romera and G. Doole. Integrated analysis of profitable stocking-rate decisions in pasture-based dairy systems. *Grass and Forage Science*, 71(1):90–101, 2016.

[182] C. Rotz, D. Mertens, D. Buckmaster, M. Allen, and J. Harrison. A Dairy Herd Model for Use in Whole Farm Simulations. *Journal of Dairy Science*, 82(12):2826–2840, 1999.

[183] C. Rougé and A. Tilmant. Using stochastic dual dynamic programming in problems with multiple near-optimal solutions. *Water Resources Research*, 52(5):4151–4163, 2016.

[184] A. Ruszczyński. Risk-averse dynamic programming for Markov decision processes. *Mathematical Programming*, 125(2):235–261, 2010.

[185] SCA. *Feeding Standards for Australian Livestock: Ruminants*. CSIRO Publications, Australia, 1990.

[186] M. Shadbolt and D. Apparo. Factors Influencing the Dairy Trade from New Zealand. *International Food and Agribusiness Management Review*, 19(B):241–255, 2016.

[187] A. Shapiro. Analysis of stochastic dual dynamic programming method. *European Journal of Operational Research*, 209(1):63–72, 2011.

[188] A. Shapiro, D. Dentcheva, and A. Ruszczynski. *Lectures on Stochastic Programming: Modelling and Theory*. Society for Industrial and Applied Mathematics, Philadelphia, 2009.

[189] A. Shapiro, W. Tekaya, J. P. da Costa, and M. P. Soares. Risk neutral and risk averse Stochastic Dual Dynamic Programming method. *European Journal of Operational Research*, 224(2):375–391, 2013.

[190] S. K. Singh. Dairy Price Risk Management for Dairy Farmers. Nuffield New Zealand Farming Scholarship, Nuffield New Zealand, 2015. URL `http://nuffieldinternational.org/rep_pdf/1483654397Satwant_Singh_2015_Report.pdf`.

[191] V. Snow, C. Rotz, A. Moore, R. Martin-Clouaire, I. Johnson, N. Hutchings, and R. Eckard. The challenges – and some solutions – to process-based modelling of grazed agricultural systems. *Environmental Modelling & Software*, 62:420–436, 2014.

[192] J. Sterman. *Business Dynamics: Systems Thinking and Modeling for a Complex World*. McGraw-Hill Higher Education, 2000.

[193] N. Stirling. Russian dairy ban continues, 2016. URL `https://agrihq.co.nz/section/dairy/dairy-products/russian-dairy-ban-continues`. [Online; accessed 2018-03-19].

[194] B. F. Taylor, P. W. Dini, and J. W. Kidson. Determination of seasonal and interannual variation in New Zealand pasture growth from NOAA-7 data. *Remote Sensing of Environment*, 18(2):177–192, 1985.

[195] G. Taylor and N. Atherfold. New Zealand Dairy Companies Review. Technical report, TDB Advisory, Wellington, New Zealand, 2017.

[196] A. Tversky and D. Kahneman. Availability: A Heuristic for Judging Frequency and Probability. *Cognitive Psychology*, 5:207–232, 1973.

[197] D. Valladao, T. Silva, and M. Poggi. High-dimensional risk-constrained dynamic asset allocation via Markov stochastic dual dynamic programming. *Optimization Online*, page 26, 2017. URL `http://www.optimization-online.org/DB_FILE/2017/01/5818.pdf`.

[198] W. van Ackooij, W. de Oliveira, and Y. Song. On level regularization with normal solutions in decomposition methods for multistage stochastic programming problems. *Optimization Online*, 2017. URL `http://www.optimization-online.org/DB_FILE/2017/01/5806.pdf`.

[199] R. M. Van Slyke and R. Wets. L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming. *SIAM Journal on Applied Mathematics*, 17(4): 638–663, 1969.

[200] D. V. Vedenov and B. J. Barnett. Efficiency of weather derivatives as primary crop insurance instruments. *Journal of Agricultural and Resource Economics*, pages 387–403, 2004.

[201] L. Vetharaniam, S. Davis, M. Upsdell, E. Kolver, and A. Pleasants. Modelling the Effect of Energy Status on Mammary Gland Growth and Lactation. *Journal of Dairy Science*, 86(10):3148–3156, 2003.

[202] F. Wahid. *River Optimization: Short-Term Hydro-Bidding under Uncertainty*. PhD Thesis, University of Auckland, Auckland, New Zealand, 2017.

[203] T. Wheeler and J. von Braun. Climate Change Impacts on Global Food Security. *Science*, 341(6145):508–513, 2013.

[204] P. Wilson. Decomposing variation in dairy profitability: The impact of output, inputs, prices, labour and management. *The Journal of Agricultural Science*, 149(04):507–517, 2011.

[205] C. Wolf. Dairy farmer use of price risk management tools. *Journal of Dairy Science*, 95 (7):4176–4183, 2012.

[206] C. A. Wolf and N. J. O. Widmar. Adoption of milk and feed forward pricing methods by dairy farmers. *Journal of Agricultural and Applied Economics*, 46(4):527, 2014.

[207] K. Woodford. Dairy price estimates are consistently wrong, 2016. URL `https://keithwoodford.wordpress.com/2016/05/31/dairy-price-estimates-are-consistently-wrong/`. [Online; accessed 2017-08-25].

[208] K. Woodford. Dairy prices are not predictable, 2016. URL `https://keithwoodford.wordpress.com/2016/04/20/dairy-prices-are-not-predictable/`. [Online; accessed 2017-08-25].

[209] L. Yates, J. Dotterer, and A. McDermott. The consequences of variability in pasture growth and milk price on dairy farm profitability. In *Proceedings of the 4th Australiasian Dairy Science Symposium*, pages 244–248, 2010.

[210] G. Zakeri, A. B. Philpott, and D. M. Ryan. Inexact Cuts in Benders Decomposition. *SIAM Journal on Optimization*, 10(3):643–657, 2000.

[211] A. Zanette, M. Fischetti, and E. Balas. Lexicography and degeneracy: Can a pure cutting plane algorithm work? *Mathematical Programming*, 130(1):153–176, 2011.

[212] W. Zhang, H. Rahimian, and G. Bayraksan. Decomposition Algorithms for Risk-Averse Multistage Stochastic Programs with Application to Water Allocation under Uncertainty. *INFORMS Journal on Computing*, 28(3):385–404, 2016.

[213] J. Zou, S. Ahmed, and X. A. Sun. Stochastic Dual Dynamic Integer Programming. *Optimization Online*, 2016. URL `http://www.optimization-online.org/DB_HTML/2016/05/5436.pdf`.

Cows, cows, everywhere.
Are calves sucking your fingers?
Calves, calves, everywhere.