University of Auckland

Department of Engineering Science

Part IV Project

Optimal Feeding Strategies for Dairy Cows

Author:
Oscar Dowson

Supervisors:
Dr. Anthony Downward
Prof. Andy Philpott

February 18, 2015

Abstract

This report documents the development, implementation and testing of MOO – the Milk Output Optimiser. In New Zealand, the dairy industry is the largest export earner. Therefore, increasing the efficiency and profitability of New Zealand dairy farms will have a large impact on New Zealand's economic performance. MOO is a Dual Dynamic Program that models a dairy farming year in weekly intervals. It calculates the optimal level of supplement to feed a cow at a weekly level, and decides how long the herd should be milked. The objective of this is to not only maximise profit, but also to ensure that the cow meets a target Body Condition Score at the end of the season. An additional feature of MOO is its ability to split the herd into an arbitrary number of groups and construct different policies for each. This allows a finer approximation of an overall herd and increases the flexibility in decision making. MOO shows promise as a useful tool to help farmers make intelligent decisions concerning supplementation, increasing both profitability and animal health.

Acknowledgments

The author acknowledges his principal supervisor, Dr. Anthony Downward, and his secondary supervisor, Professor Andy Philpott. Their suggestions and feedback were invaluable, despite the fact that at times neither they nor this author understood what was being discussed.

The author also acknowledges the Livestock Improvement Corporation for awarding him a Pat Shannon Scholarship in 2014.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Dairy Industry: In 2010, the New Zealand Institute of Economic Research completed a comprehensive review of the dairy sector's contribution to the New Zealand economy (Schilling, Zuccollo, and Nixon, 2010). They found that in 2009, the dairy industry in New Zealand directly accounted for 2.8% of G.D.P and provided 26% of New Zealand's total goods exports. In addition, the dairy sector employs around 35,000 people, excluding those who are self-employed (potentially another 10,000). It provides more jobs than the finance and accommodation sectors and 65% more than the sheep and beef farming sector. A $1 increase in the dairy payout delivers an additional $270 per year (2009 dollars) in additional income for every person in New Zealand .

Season: In New Zealand, the dairy farming year follows the seasons. Depending on location, cows begin to calve sometime in July. Through spring, cows produce maximum milk yield (Figure 1.1) and are impregnated using artificial insemination around October.

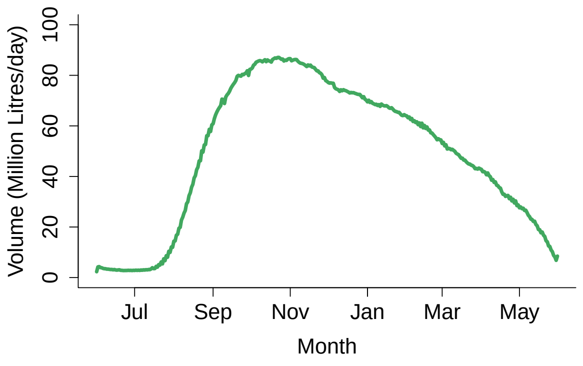


Figure 1.1: Volume of milk supplied to Fonterra in the 2013/14 Season (Fonterra, 2014b).

As summer approaches, milk production drops along with feed supply. By February, some farms are down to once-a-day milking instead of the usual twice-a-day. Depending on the level of feed, cows are dried off at the beginning of April. This allows them to recover and regain body condition prior to their next calving in July.

Body Condition: A body condition score (BCS) is a subjective score of the body fat reserves of the animal. In New Zealand a BCS scale of 1 to 10 is used. A cow with a BCS greater than 6 is considered overweight, while a BCS less than 3 is considered underweight (Dairy NZ, 2014d). A body condition score of 6.5 at calving has been found to maximise milk production in the following season. A BCS greater than this increases the risk of acidosis and other negative health outcomes (Roche et al., 2007).

Supplementation: A supplement is an energy source given to dairy cows over and above the energy that cows consume from grazing on pasture. Common supplements in New Zealand include hay, silage, molasses and maize (Dairy NZ, 2014a).

In the past, the majority of New Zealand dairy farms were purely pasture based (farm system 1: Dairy NZ, 2014c). In the last few decades however, with the improved availability of supplements such as PKE (palm kernal expeller) (Dairy NZ, 2008b), farms have been moving to higher supplementation systems (farm system 4-5).

In the last five years, the proportion of low input (farm system 1 or 2) farms decreased from 44% in the 2007/08 season (Dairy NZ, 2008a) to 35% in the 2012/13 season (Dairy NZ, 2013). This is one reason for the average kg milksolids per cow increasing from 259KgMS/Year in 1992/93 season to 346KgMS/Year in the 2012/13 season (Dairy NZ and LIC, 2013).

Revenue: The majority (87%) of farmers in New Zealand belong to the dairy co-operative Fonterra. Other companies include Open Country Dairy, Synlait and the Tatua Co-operative Dairy Company (Paterson, 2014). All companies collect milk from farmers in refrigerated tankers. The milk is analysed to determine milkfat and protein percentages. Farmers are then paid based on the total milk solids (MS; kg milkfat + protein) collected (Fonterra, 2014a). The price per kgMS varies with global markets (Global Dairy Trade, 2014) leading to uncertain revenue in the future.

Farmers also face high costs in order to maintain cow and land condition. This includes things like fertiliser, veterinarian visits, power, machinery and supplementary feed. Of these costs, feed is the largest component at 30.8% (Dairy NZ, 2013).

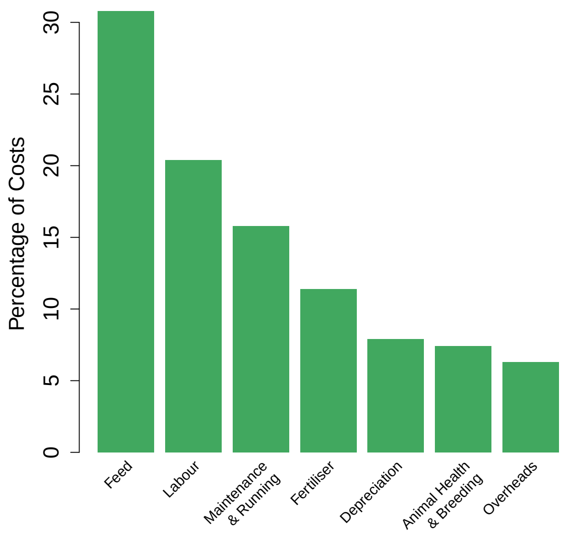


Figure 1.2: Breakdown of farm costs.

1.2 Literature

The literature contains a number of attempts to create models to predict dairy cow performance. We split these into two categories: simulation and optimisation.

1.2.1 Simulation

Various authors (SIMCOW: Kristensen, Sorensen, and Clausen, 1997; MOOSIM: Bryant, 2006; E-Cow: Baudracco et al., 2011) have modelled the biological response of dairy cattle to feed input. Predicted variables are usually feed intake (pasture and supplementation), body condition, liveweight and milk solids (MS) produced. This allows simulations to be conducted to measure the response of dairy cows to different conditions.

Both E-Cow and MOOSIM report on the accuracy of their models. They both use the concordance correlation coefficient (CCC: Lin, 1989) as a measure of the reproducibility of their predictions. A CCC of 0 represents no concordance between the actual and predicted values, while a CCC of 1 represents perfect concordance between the actual and predicted values. E-Cow incorporates many of the same ideas and principles as MOOSIM, and then extends them, so it is not surprising that it performs better (Table 1.1) .

	DMI	LW	MY
E-Cow	0.81	0.61	0.74
MOOSIM	0.27	0.16	0.59
E-Dairy*	0.93	0.69	0.93

Table 1.1: Reported Concordance Correlation Coefficients for predicted dry matter intake (DMI), liveweight change (LW) and milk yield (MY). *E-Dairy is a Whole Farm Model as opposed to a single cow model.

Instead of focusing on a single cow, Whole Farm Models (DAFOSIM: Rotz et al., 1999; DairyWise: Schills et al., 2007; e-Dairy: Baudracco et al., 2012) can be created to simulate entire herds. E-Dairy is the most recent of these and is composed of a 'herd' of correlated E-Cows. Prediction of herd level variables (such as total milk solids produced) is more accurate (Table 1.1) than that of a single cow as random errors are averaged out. Whole Farm Models are a good tool for farmers to use, but are more complicated than single cow models.

Other tools are available (Dairy NZ, 2014b) which allow farmers to calculate the break even cost of various supplements, however there are rough generalisations and do not integrate cow models.

1.2.2 Optimisation

There have been prior attempts (Bath and Bennett, 1980, Klein et al., 1986) to use linear programming in order to determine the optimal amount of supplement to feed at a given stage of lactation. They begin by calculating the minimum requirements for protein, energy and various vitamins and minerals at different stages of lactation, fitting regression models to a herd of Canadian dairy cows. This data is then used to solve the classical diet problem (Stigler, 1945)[1].

The author is not aware of any published literature that has attempted to optimise more recent simulation models in a mathematical programming framework.

Stochastic optimisation has been used in order to optimise drying-off policy in the face of drought (Mills, 2013). However, the model does not use an accurate animal model, which may limit the applicability of its policies to real world situations. However, it does set out a good foundation upon which other stochastic dairy optimisation models can be based.

1.3 Overview

Project Aim: We can draw two conclusions from the above information; the first is that the intelligent management of supplementary feed is likely to have a significant impact on the profitability of New Zealand dairy farms. In addition, it enables better management of cow condition, leading to improved health and productivity. Secondly, there exist biological models of dairy cows that have been developed and iterated for many years.

Therefore, the aim of this project was to take an existing simulation model of a dairy cow, and apply mathematical optimisation to it in order to develop a model which is able to help dairy farmers make intelligent decisions surrounding supplementary feed.

Report Overview: The resulting model, MOO (the Milk Output Optimiser), is a Dual Dynamic Program (DDP) that models a 52 week year, and calculates the optimum level of supplement to feed each week with the objective of maximising profit.

The next chapter presents an existing simulation model of a dairy cow (E-Cow), and uses that model to test simple supplementation policies. This model is non-linear, and therefore presents a challenge to

[1]Dantzig (1990) gives an entertaining account his attempt at using the diet problem for his personal use

optimise. Chapter 3 thus outlines how to construct a piecewise linear approximation of the simulation discussed in Chapter 2 so that it can be solved using mathematical programming techniques.

In Chapter 4 we discuss a class of problems known as Optimal Control Problems and give a general formulation of the problem. In Chapter 5 we outline three different solution methods for the piecewise linear optimal control problem developed over the last three chapters. Chapter 6 takes two of these methods and compares their solution times and accuracy. The solution methods tested were a time staged, mixed integer program (MIP), and a Dual Dynamic Program (DDP). We settle upon a Dual Dynamic Program as our preferred solution method as the MIP is to computationally expensive to solve.

In Chapter 7 we extend the DDP algorithm so that it can accommodate binary state variables. This allows a milking decision (to milk or not to milk) to be incorporated into the DDP, rather than prescribing a policy prior to solving the model. This requires both a modification to the Bender's cuts that approximate the value-to-go function, and a two phase solution method that takes advantage of certain properties of the model.

We then explore some preliminary results of this modified model in Chapter 8. In particular, we investigate how an initial body condition score influences the optimal policy.

We discuss possible extensions to the model in Chapter 9, before concluding in Chapter 10.

2 E-Cow simulation

In this chapter, we outline E-Cow, a model that was developed at Massey University (Baudracco et al., 2011). We then use it to explore simple supplementation policies to generate a basic understanding of how the model behaves, and the results we can expect.

2.1 Overview

E-Cow integrates three prior models; herbage intake (Baudracco et al., 2010), milk production (Vetharaniam et al., 2003) and body condition change (Friggens, Ingvartsen, and Emmans, 2004). It is a deterministic simulation with a daily time-step. It has since been implemented in a web interface (http://e-cow.net: Baudracco, Lopez-Villalobos, and Zamateo, 2008).

E-Cow is, in essence, an energy balance model of a cow. It first predicts the energy intake of a cow given factors such as pasture availability, pasture quality and stage of lactation. This energy is partitioned into required energy (such as that allocated to a growing fetus, and for maintenance), energy for milk production and energy for fat deposition. For a detailed explanation of the model see Baudracco (2011).

Note: In the United States, a 1-5 BCS scale is used instead of the 1-10 New Zealand scale. Equation 2.1 gives the appropriate conversion (Roche et al., 2009):

$$US = 1.5 + 0.32 \times NZ. \tag{2.1}$$

E-Cow uses the US BCS scale, so we will use this scale for the remainder of this paper.

2.2 Formulation

Although the original E-Cow model has a daily time-step, we reformulate it here with a weekly time-step and make some simplifications.

2.2.1 List of Terms

Constants in time: These are explicit functions of the week of lactation (although may require information such as liveweight at calving, and body condition score at calving). These functions can be found in Appendix A.

$pmy(t)$	maximum potential milk yield (MJ/week)
$lip(t)$	genetically driven change in lipid mass (kg/week)
$mep(t)$	energy required for pregnancy (MJ/week)
$sol(t)$	stage of lactation
$lwp(t)$	change in LW due to pregnancy (kg/week)

Constants: These are constants in the model.

A	Age of cow (Years)
B_1	Body condition score at calving
$BCSperLW$	Change in BCS per change in LW (BCS unit/kgLW)
C^M	Value of milk (\$ per kgMS)
C^S	Cost of supplement (\$ per Tonne)
H^A	herbage offered (kgDM/week)
H^E	Energy content of pasture (MJ/kgDM)
H^N	Neutral Detergent Fibre proportion of pasture
H^D	Digestibility of pasture
KM	Energy efficiency of lipid deposition
KL	Energy efficiency of milk production
S^E	Energy content of supplement (MJ/kgDM)
W_0	Liveweight at calving (kg)

Assumption We assume that the energy of lipid deposition is identical for gaining and losing body lipid. This is in order to avoid the need to split variables into positive and negative components.

Variables in model: All variables are indexed by a subscript t for each week of lactation.

B_t	Body Condition Score
H_t^I	Actual Herbage Intake (kgDM)
L_t	1 if cow is milking, 0 otherwise
W_t	Liveweight (kg)
E_t^l	Energy required for genetically driven BC change (MJ)
E_t^m	Energy required for maintenance (MJ)
E_t^{milk}	Energy allocated to milk production (MJ)
E_t^n	Net Energy before partitioning (MJ)
E_t^r	Total Energy Requirement (MJ)
E_t^t	Total Energy Intake (MJ)
H_t^p	Maximum potential Dry Matter Intake (kgDM)

2.2.2 Energy Requirements

All the equations in this, and the following sections are taken from the E-Cow model (Baudracco et al., 2011).

In a given week, a cow will require a certain amount of energy for maintenance, body condition change, pregnancy and milk production:

$$E_t^r = E_{t-1}^m + E_t^l + pmy(t) \times L_t + mep(t). \tag{2.2}$$

A cow will have some genetic drive to change body lipid percentage, even if there are no nutritional constraints (i.e. underfeeding). This can lead to a negative energy requirement ($E_t^l < 0$) if it is consuming stored fat. The energy required for lipid change is:

$$E_t^l = lip(t) \times (10.1 + 1.976 \times B_t)/KM. \tag{2.3}$$

The energy that the cow requires to maintain its present state is a non-linear function of liveweight, herbage intake and milk production

$$E^m_{t+1} = \gamma_0 W_t^{0.75} + \gamma_1 W_t \times H^I_t + \gamma_2 \times W_t + \gamma_3 \times E^{milk}_t. \tag{2.4}$$

See Appendix A for definitions of γ_0, γ_1, γ_2, γ_3.

The total herbage intake of a cow is governed by three things: the size of the rumen, the maximum energy requirement and the rate at which it can graze. Therefore, the maximum potential dry matter intake is the minimum of these three terms:

$$H^p_t = \min\{0.0375 \times sol(t) \times W_t, \quad 0.165 \times sol(t) \times W_t/H^N, \quad E^r_t/H^E\}. \tag{2.5}$$

We then calculate the harvesting efficiency of the cow when grazed on rye-grass as:

$$HE = 0.57676 \times \left(\frac{H^p_t}{H^A}\right)^{0.536}. \tag{2.6}$$

Then use the harvesting efficiency to calculate the actual herbage intake:

$$H^I_t = HE \times H^A \times \left[1 - 21 \times \left({}^{S_t}/_{W_t}\right)\right] + 0.18 \times S_t. \tag{2.7}$$

Thus, the total energy intake in a given week, is a combination of the total energy consumed from pasture, plus the total energy consumed from supplement:

$$E^t_t = H^E \times H^I_t + S^E \times S_t. \tag{2.8}$$

2.2.3 Energy Allocation

The net energy balance of the cow in any week is therefore the total energy intake, minus the required energy. This net energy is then partitioned into stored fat (ΔE^l_t) and milk production (ΔE^{milk}_t):

$$E^n_t = E^t_t - E^r_t = \Delta E^l_t + \Delta E^{milk}_t. \tag{2.9}$$

When calculating our required energy, we assume that the cow produces the maximum amount of milk possible. Therefore, when partitioning the net energy, this value can only be negative. When the cow is in positive energy balance, it partitions all available energy towards depositing fat. When a cow is in negative energy balance, it will reduce its milk production, and reduce the deposition of fat (such deposition may be negative, i.e. the cow is burning stored fat). If the cow is in a large negative energy balance, then it will cease producing milk entirely.

$$\Delta E^{milk}_t = \begin{cases} 0 & E^n_t > 0 \\ -pmy(t) & E^n_t < -\beta_1 \\ \beta_2 \times E^n_t & otherwise \end{cases}, \tag{2.10}$$

where

$$\beta_1 = pmy(t) + |E^l_t|, \tag{2.11}$$

$$\beta_2 = \frac{pmy(t)}{\beta_1}. \tag{2.12}$$

The total energy allocated to milk production is therefore the maximum potential milk yield plus any energy from partitioning when we choose to milk the cow, and zero in the case when we do not milk the cow

$$E^{milk}_t = \begin{cases} pmy(t) + \Delta E^{milk}_t & L_t = 1 \\ 0 & L_t = 0 \end{cases}. \tag{2.13}$$

Another constraint is that we cannot spend negative energy on milk production

$$E^{milk}_t \geq 0. \tag{2.14}$$

2.2.4 Liveweight

The liveweight of the cow in any week is a linear function of B_t plus the weight of the growing fetus

$$W_t = B_t/BCSperLW + lwp(t). \tag{2.15}$$

2.2.5 Milk Solids

Therefore, the total kilograms of milksolids produced in week t is a function of the week of lactation and the total energy devoted to milk production (E_t^{milk}):

$$kgMS_t = E_t^{milk} \times [fat(t) + prot(t)] \times [0.0376 \times fat(t) + 0.0209 \times prot(t) + 0.948]/KL. \tag{2.16}$$

2.2.6 Profit

We define a simple profit function as the total value of milk, less the cost of supplement:

$$P = \sum_{t=1}^{52} kgMS_t \times C^M - S_t \times C^S. \tag{2.17}$$

2.2.7 Update States

The body condition score of a cow in any week is the BCS in the previous week, plus some function of both the energy allocated towards fat deposition, and its current BCS

$$B_{t+1} = B_t + [(E_t^l + \Delta E_t^l) \times BCSperLW \times KM/(10.1 + 1.976 \times B_t)]. \tag{2.18}$$

2.3 Simulated Policies

In this section, we present the results from the *E-Cow* simulation under different supplementation policies. The models were run with the parameters listed in Table 2.1.

H^A	H^N	H^E	S^E	W_1	B_1	C^M	C^S	L_{41}
220	0.44	10.3	10.3	480	3.2	6.5	400	0

Table 2.1: Parameters for simulated policies.

Table 2.2 lists the policies, their net return (milk value - supplement cost), and final BCS. Policies 1 and 4 (No supplementation and 2kg/d while milking) failed to meet the target BCS. This was not penalised in this simulation. The best policy to choose from the five tried is to feed 5kg/d of supplement while the cow is milking. This is largely because feeding the cow extra during lactation allows it to create more milk. In contrast, feeding supplement when the cow is dry increases the final BCS, but generates no extra income. However, it may be desirable in an optimisation to feed supplement when the cow is dry if a cost on missing a BCS target is included.

Policy	Obj. ($)	Final BCS
No Supplementation	4,714.40	2.97
2kg/d	5,230.32	3.97
5kg/d	5,982.17	5.43
2kg/d while milking	5,304.99	3.06
5kg/d while milking	6,168.84	3.28

Table 2.2: Results of simulated supplementation policies.

2.3.1 Policy 1: No supplementation

Under this policy, no supplementary feed is provided. The cow fails to meet the final body condition target of 3.2. Assuming there is no penalty for missing this target, the farmer makes a profit of $4,714.40.

Figure 2.1 shows four plots of the cow's status over time. In all the plots, the x-axis represents the number of weeks since the cow last calved (Weeks in Milk).

The left plot shows BCS against weeks in milk. The cow starts with a BCS of 3.2, drops to a minimum BCS around week 15, then begins gaining condition.

The middle plot shows the feed intake (in kgDM/week) of both pasture (solid line) and supplement (dashed line). In this case, the supplement is zero due to the policy. The sudden drop in pasture intake after week 40 is due to the cow being dried off; it no longer has a large energy requirement and so desires less pasture.

The right graph plots milk solids against days in milk. The solid line is the actual milk solids produced, while the dashed line represents the biological maximum.

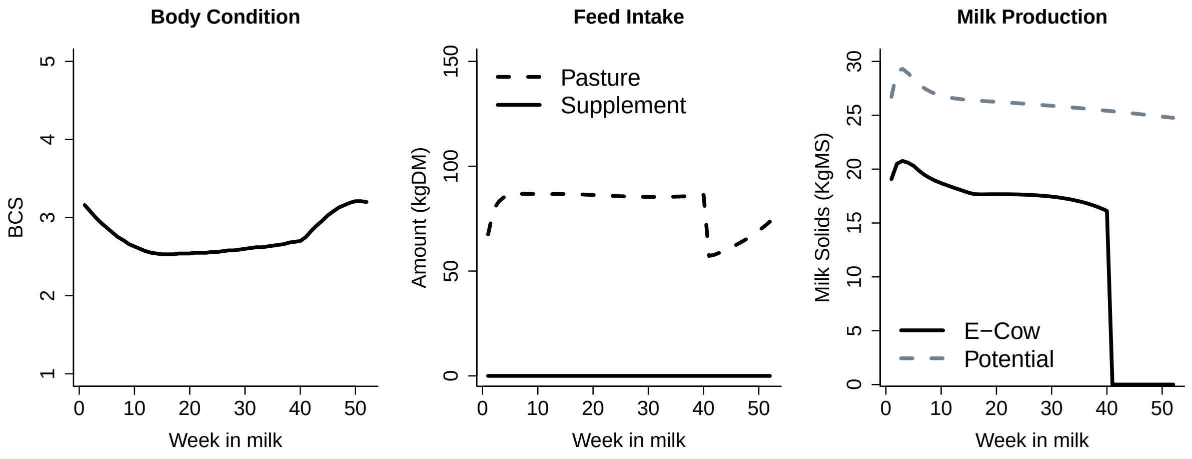


Figure 2.1: Predicted response under Policy 1.

Similar plots for the other four policies can be found in Appendix A.

2.3.2 Policy 2: 2kg/day

Under this policy, 2kgDM of supplement is provided daily. As shown in Figure A.1, the cow makes more milk during lactation than the cow fed no supplement, although this is still below the maximum possible. Once lactation stops, the cow partitions all of its net energy into fat deposition, which leads to a rapid increase in body condition. The cow meets the BCS target and the farmer makes $5230.32.

2.3.3 Policy 3: 5kg/day

Under this policy, 5kgDM of supplement is provided daily. The cow produces very close to the maximum amount of milk possible due to the high amount of energy intake during lactation (Figure A.2). Once dried off, the cow puts on body condition very quickly. The cow meets the BCS target and the farmer makes $5,982.17.

2.3.4 Policy 4: 2kg/day while milking

Under this policy, 2kgDM of supplement is provided while the cow is lactating. No supplement is fed once the cow is dried off. The cow produces the same amount of milk as policy 2 (Figure A.3), but eats less supplement so the farmer makes an extra $74.67 compared to policy 2. However, the cow fails to meet the BCS target by 0.14BCS units.

2.3.5 Policy 5: 5kg/day while milking

Under this policy, 5kgDM of supplement is provided while the cow is lactating. No supplement is fed once the cow is dried off. The cow produces the same amount of milk as policy 3, but eats less supplement (Figure 2.2). The farmer makes an extra $186.67 compared to policy 3. In addition, the cow meets the BCS target, but by less than policy 3.

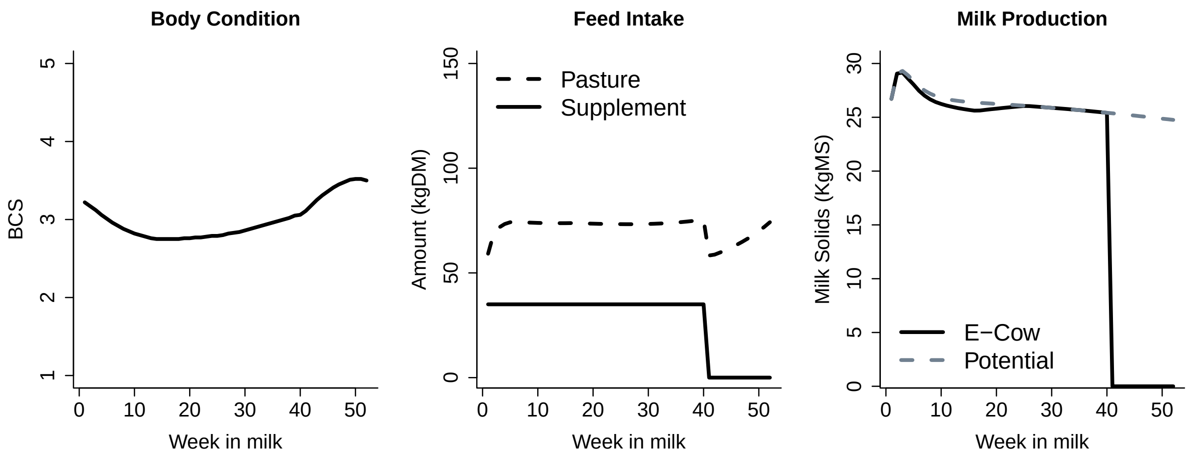


Figure 2.2: Policy 5: 5kg/cow/day while milking.

2.3.6 Conclusion

Based on this analysis, we expect that the optimal supplement policy should look similar to policy five. This is because it is making close to the maximum possible amount of milk, and meets the BCS target.

3 Linearisation

The model outlined in the previous chapter is non-linear, and therefore unable to be solved via standard linear programming techniques in its current form. To avoid this problem, we create a piecewise linear approximation of the original model.

3.1 Linearisation Methods

3.1.1 Special Ordered Sets of Type II

A Special Ordered Set of Type II (SOS2) is an ordered set of non-negative variables, of which at most two can be non-zero, and, if two are non-zero, then these must be consecutive in their ordering (Beale and Tomlin, 1970).

To approximate the function $f(z)$ we add the following constraints

$$\begin{aligned} \sum a_i\lambda_i &= z, \\ f(z) &= \sum f(a_i)\lambda_i, \\ \sum \lambda_i &= 1, \end{aligned} \tag{3.1}$$

where the set λ is a SOS2.

Figure 3.1 shows how SOS2 might be used to approximate $f(x) = x^2$.

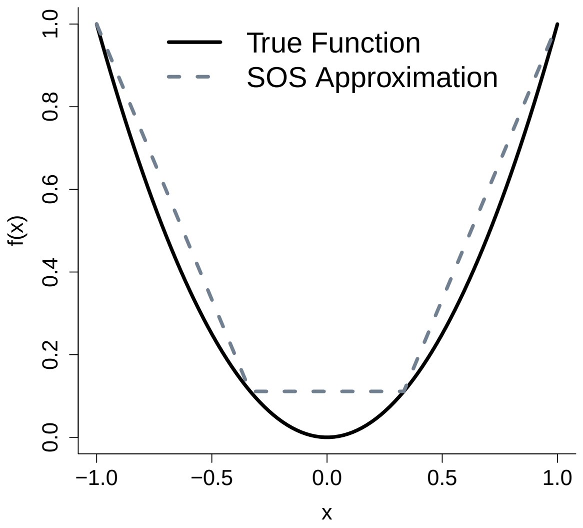


Figure 3.1: SOS2 approximation of $f(x) = x^2$ with three segments.

3.1.2 Bi-linear Term

We can rewrite a bilinear term as the following:

$$xy = \frac{(x+y)^2 - (x-y)^2}{4} \tag{3.2}$$

and then use SOS2 to fit a piecewise approximation to each quadratic term (Gabriel et al., 2006).

3.2 Modifications

3.2.1 Potential Dry Matter Intake

A greater potential dry matter intake allows more pasture to be eaten, and therefore more milk to be produced. This means that any optimal solution should seek to maximise the potential dry matter intake H_t^p. Therefore can replace the minimisation function with inequalities of the following form:

$$H_t^p \leq \alpha_i. \tag{3.3}$$

3.2.2 Herbage Intake

Plotting the surface (Figure 3.2) shows that it looks concave over the relevant domain. However, it is non-concave due to a bi-linear term.

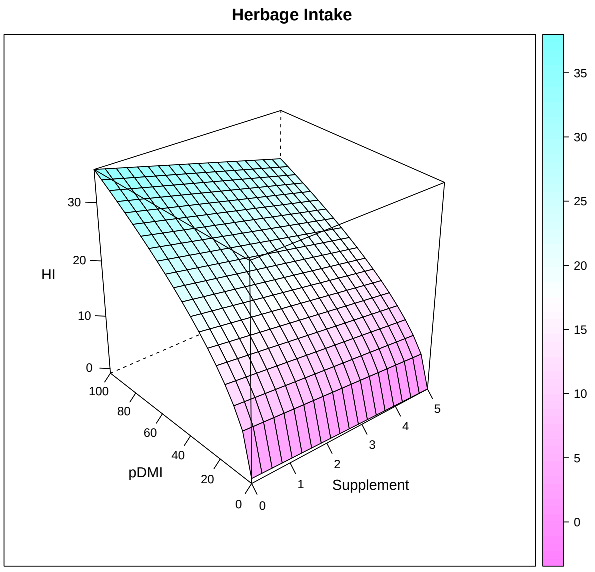


Figure 3.2: Non-Concave surface of H^I (daily values, $W = 480$).

However, we can break down the HI equation into three terms

$$H_t^I = \underbrace{HE \times H^A}_{\text{term 1}} - \underbrace{HE \times H^A \times 21 \times \left({}^{S_t}/_{W_t}\right)}_{\text{term 2}} - \underbrace{0.18 \times S_t}_{\text{term 3}}. \tag{3.4}$$

Term 1

$$term1_t = 0.57676 \times \left(H^A\right)^{0.464} \times \left(H_t^p\right)^{0.536}. \tag{3.5}$$

To deal with this we create a new variable

$$\hat{H}_t^p = \left(H_t^p\right)^{0.536}.$$

We can see that as $\hat{H}_t^p$ increases, the herbage intake also increases (since $term1 > term2$). This leads to increased energy intake, leading to higher milk production and less need for supplement. Therefore (so long as we are maximising profit), any optimal solution will have $\hat{H}_t^p$ at its maximum possible value. Thus we can approximate $\hat{H}_t^p$ by a series of linear cuts on its outer surface.

The linear cuts are calculated using the first order Taylor polynomial expansion

$$f(x) = f(\bar{x}) + f'(\bar{x})(x - \bar{x}). \tag{3.6}$$

This gives us a formula for the cuts:

$$\hat{H}_t^p \leq \left[0.464 \times \left(\overline{H_t^p}\right)^{0.536}\right] + \left[0.536 \times \left(\overline{H_t^p}\right)^{-1.536}\right] \times H_t^p. \tag{3.7}$$

Example cuts are shown in Figure 3.3.

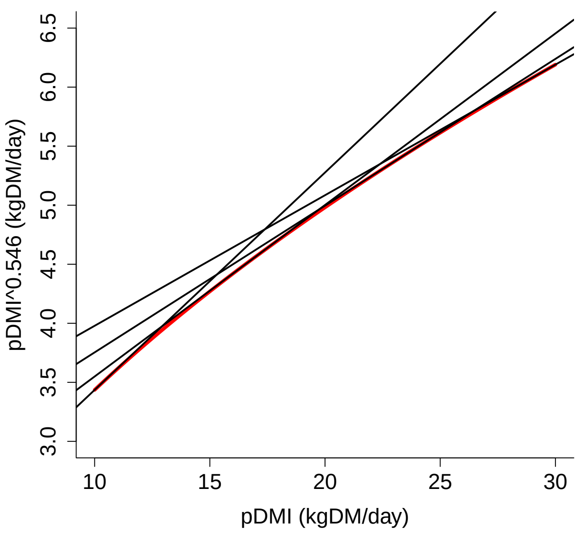


Figure 3.3: Linear cuts defining maximum pasture intake.

Term 2 This term contains three variables, H_t^p, S_t and LW_t:

$$term2_t = 0.57676 \times H^{A^{0.464}} \times H_t^{p^{0.536}} \times 21 \times S_t/W_t. \tag{3.8}$$

We can substitute our new variable $\hat{H}_t^p$ in

$$term2_t = 0.57676 \times H^{A^{0.464}} \times \hat{H}_t^p \times 21 \times S_t/W_t. \tag{3.9}$$

Of these three variables, W_t is the least dominant. Further, since we can expect the liveweight to decrease initially to below the liveweight at calving, then increase again as the cow regains body condition and is pregnant for longer, we can make an approximation by setting $W_t = W_1$ for all stages.

We now only have to worry about the bilinear term between $\hat{H}_t^p$ and S_t, however we can use the bilinear decomposition and two SOS2 to approximate the quadratic terms.

$$term2 = \left(0.57676 \times HA^{0.464} \times 21/W_1\right) \times 0.25 \times \left[\left(\hat{H}_t^p + S_t\right)^2 - \left(\hat{H}_t^p - S_t\right)^2\right]. \tag{3.10}$$

These two SOS2 approximations - $(\hat{H}_t^p + S_t)^2$ and $(\hat{H}_t^p - S_t)^2$ - shall be referred to as $sosHI^+$ and $sosHI^-$ respectively in the remainder of this report.

Term 3 Term 3 is linear and so needs no modification

New herbage intake equation This gives a new equation for herbage intake:

$$H_t^I = term1_t - term2_t - term3_t. \tag{3.11}$$

3.2.3 Maintenance

There are two non-linear terms in the original maintenance equation: a polynomial and a bilinear term.

Any optimal solution will seek to minimize maintenance energy as this is a sink of energy that does not return any benefit. Therefore, unlike H_t^p in the previous section, we cannot approximate $LW^{0.75}$ by linear cuts on the outer surface as it is concave rather than convex. This means that we need to approximate it by a SOS2.

The bilinear term involving W_t and H_t^I can be handled using the bilinear decomposition and two SOS2 for the quadratic terms.

Thus, the maintenance equation can rewritten as follows:

$$\begin{aligned} MEm(W_t, E_t^{milk}, H_t^I) =&\gamma_0 \times W_t^{0.75} + \gamma_1 \times \frac{1}{4}[(W_t + H_t^I)^2 - (W_t - H_t^I)^2] \\ &+ \gamma_2 \times W_t + \gamma_3 \times E_t^{milk}. \end{aligned} \tag{3.12}$$

This decomposition requires three SOSs: $W^{0.75}$, $(W_t + H_t^I)^2$ and $(W_t - H_t^I)^2$, referred to as $sosLW$, $sosMEm^+$ and $sosMEm^-$ respectively.

3.2.4 Net Energy to Milk Allocation

In the E-Cow simulation, partitioning energy to milk allocation requires a piecewise linear function. This can be achieved using SOS2.

However, there is an additional complication: one of the vertices's in the SOS2 is a function of BCS. Therefore when implementing the SOS2 there is a bilinear term. However, this can be approximated using the same bilinear decomposition and SOS2 approximation outlined previously.

In addition, we have to take into account the decision to milk or not (since when the cow is dried off $pmy(t) = 0$).

$\mathbf{\Delta E^{milk}}$ To deal with cows milking and dried off, we can decompose ΔE_t^{milk} into two variables, ΔE_t^{milk+} and ΔE_t^{milk-}. Then

$$\Delta E^{milk} = \Delta E_t^{milk+} + \Delta E_t^{milk-}, \tag{3.13}$$

$$0 \geq \Delta E_t^{milk} \geq -pmy(t) \times L_t, \tag{3.14}$$

$$0 \leq \Delta E_t^{milk+} \leq pmy(t)(1 - L_t). \tag{3.15}$$

sosMEmilk #1 This SOS2 is not used to approximate a non-linear function but to implement a piecewise linear function.

It has three sections, giving four nodes:

$$(a_i, f(a_i)) = \{(-800, -pmt(t)), (-pmy(t) - |E_t^l|, 0), (0, 0), (0, 200).\} \tag{3.16}$$

With the SOS2 constraints

$$\begin{array}{rcl} \sum a_i \lambda_i &=& E_t^n, \\ \Delta E_t^{milk-} &=& \sum f(a_i)\lambda_i, \\ \sum \lambda_i &=& 1. \end{array} \tag{3.17}$$

This SOS2 is referred to as sosMEmilk1.

sosMEmilk #2 When expanding sosMEmilk1, we end up with

$$E_t^n = \cdots - (pmy(t) + |E_t^l|) \times \lambda_2 + \ldots. \tag{3.18}$$

Expanding again gives

$$\begin{aligned} MEn_t = &\cdots - pmy(t)\lambda_2 - |lip(t)| \times 10.1/KM \times \lambda_2 \\ &- |lip(t)| \times 1.976/KM \times (B_{t-1} \times \lambda_2) \\ &+ \ldots \end{aligned} \tag{3.19}$$

which contains the bilinear term $B_{t-1} \times \lambda_2$.

The two SOS2 needed to approximate the quadratics, $(B_{t-1} + \lambda_2)^2$ and $(B_{t-1} - \lambda_2)^2$ are referred to as $sosMEn^+$ and $sosMEn^-$ respectively.

3.2.5 Change in BCS

Whilst this equation in its original form is not linear, we can linearise it by rewriting it as

$$B_t = \xi_0 B_{t-1} + \xi_1(E_t^l + \Delta E_t^l) \tag{3.20}$$

and use linear regression to obtain values for ξ_0 and ξ_1 (Table 3.1). However, this effectively negates the cross term between BCS and MElip. This results in small (< 0.01BCS/week) residuals relative to the actual BCS.

Important: Since this approximation is made 52 times, it is possible that the error may accumulate and become serious. This can be validated against the E-Cow simulation to check if the BCS diverges from the true value.

Coefficient	Value	p-value
ξ_0	1.00	$< 2 \times 10^{-16}$
ξ_1	5.979×10^{-4}	$< 2 \times 10^{-16}$
R^2	0.9999	

Table 3.1: Example Linear Regression parameter summary for BCS. $W_0 = 480$, $B_0 = 3.2$.

This linear regression needs to be re-calculated at the beginning of each solve (or whenever the B_0 or W_0 is changed).

3.3 Accuracy of Approximation

Assuming we are approximating the function $f(x)$ over the domain X and the value of the SOS2 approximation at the point x is $\bar{f}(x)$, we can define the maximum relative percentage error (MRPE) as

$$MRPE = \max_{x \in X} \left\{ \left| \frac{f(x) - \bar{f}(x)}{f_{max} - f_{min}} \right| \right\} \tag{3.21}$$

where f_{max} and f_{min} are the greatest and smallest values in the co-domain respectively.

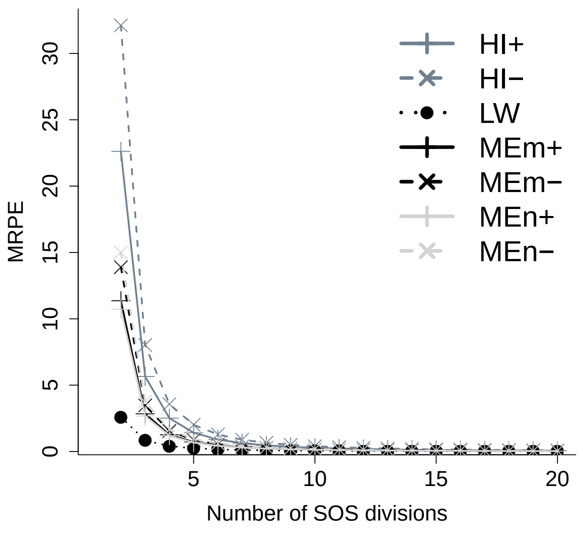


Figure 3.4: MRPE of SOS2 approximation for each of the SOS2 constraints against the number of divisions.

3.3.1 SOS2

When using SOS2 to approximate non-linear functions, the quality of the approximation is determined by the number of SOS divisions and the domain of the SOS2 (Table 3.2).

Plotting the MRPE against the number of SOS2 divisions (Figure 3.4) allows us to determine the number of divisions needed in each SOS2 to meet some maximum error. Table 3.2 gives the number of divisions needed for each SOS2 in order to meet varying upper limits on MRPE.

The trade off in this is that increasing the number of SOS2 divisions increases the complexity of the MIP, therefore increasing solution times. This is explored in Section 6.2.

	Domain		MRPE			
SOS	α_{min}	α_{max}	5%	1%	0.5%	0.1%
HI^+	0	20	3	5	7	16
HI^-	-5	15	3	6	9	18
LW	300	700	1	2	3	7
MEm^+	300	800	2	4	5	11
MEm^-	200	600	2	4	6	12
MEn^+	20	5	2	4	4	11
MEn^-	1	4	2	4	6	13

Table 3.2: Number of SOS2 divisions to achieve MRPE.

In all future solves (unless otherwise stated) we will use the number of divisions required to reach a MRPE of 0.1%.

3.3.2 Cutting Planes

We can conduct a similar analysis of the approximation of $\hat{H}_t^P$. In this case, the approximation is better than the other SOS2 approximations, as we only need five cutting planes to have a MRPE less than 0.1%.

4 Optimal Control Problems

In this chapter, we discuss what an optimal control problem is, and how E-Cow can be optimized using its ideas.

4.1 Overview

An optimal control problem is an optimisation problem that can be solved to find an optimal control policy for a dynamical system over time. The method was largely developed by Lev Pontryagin (Pontryagin, 1987) in the then USSR and Richard Bellman in the United States[1].

4.1.1 Formulation

The problem is split into state variables $\mathbf{x}$ and control variables $\mathbf{a}$. A state variable is a variable that defines the characteristics of the system. A control variable is a variable that can be modified to change the state variables.

Objective: It requires some cost function to minimize

$$\Pi = \sum_{t=1}^{N} f_t(\mathbf{x}_t, \mathbf{a}_t). \tag{4.1}$$

Change in state: As well as some law that governs the change in the state variable from one time period to another

$$\mathbf{x}_{t+1} = F_t(\mathbf{x}_t, \mathbf{a}_t). \tag{4.2}$$

Initial Conditions: And some initial conditions

$$\mathbf{x}_1 = \mathbf{b}. \tag{4.3}$$

4.2 Problem Definition

4.2.1 Stages

The original E-Cow model has a daily time-step. This would result in a model with a large number of stages. To reduce the number of stages, we aggregate the equations into a weekly time-step. Therefore, there are 52 stages, one for each week:

$$T := \{1, 2, ..., 52\}.$$

4.2.2 States

There are two state variables, body condition and energy required for maintenance

$$\mathbf{x_t} = \{B_t, E_t^m\}.$$

[1]see Pesch and Plail (2009) for an excellent overview of the simultaneous development of the Maximum Principle of Optimal Control in both the East and West

4.2.3 Simplification

In E-Cow, an input to each day is the amount of energy required for maintenance, E_t^m. When this is changed to a weekly time-step, the total energy required for maintenance during the week will be $\frac{1}{7}E_{t-1}^m + \frac{6}{7}E_t^m$. Therefore, the energy from the current week dominates the energy requirement from the previous week. In addition, the energy required for maintenance does not vary by a large amount week to week. In order to simplify our state space, we can move all of the maintenance energy into the current week. This simplification can be checked by validating the optimisation model against E-Cow with the daily time-step. If there are differences between the two with respect to E_t^m then this simplification can be re-examined. Therefore, we only have a single state variable:

$$\mathbf{x_t} = \{B_t\}.$$

4.2.4 Actions

In a given week, we have two decisions to make: how much supplement to feed the cow, and whether or not to milk it:

S_t	KgDM of supplement given to the cow in week t
L_t	$\begin{cases} 1 & \text{Milk the cow in week } t \\ 0 & \text{Don't milk the cow in week } t \end{cases}$

To simplify the model, we shall fix our milking policy prior to solving. This turns L_t from a variable into a parameter. We shall remove this simplification in Chapter 7.

4.2.5 Rewards

The reward we receive in a given week is the value of the milk produced in that week, minus the cost of supplement provided in that week.

$$R_t(\mathbf{x}_t, \mathbf{a}_t) = f(t) \times E_t^{milk} - g(t) \times S_t \tag{4.4}$$

where f(t) is the value of 1KJ of energy allocated towards milk production in week t, and g(t) is the value of 1Kg of supplement in week t.

4.2.6 Recursion

We then define the Bellman recursion function

$$V_t(\mathbf{x}_t) = \max_{\mathbf{a}_t}\{R_t(\mathbf{x}_t, \mathbf{a_t}) + V_{t+1}(F_t(\mathbf{x}_t, \mathbf{a}_t))\}. \tag{4.5}$$

4.3 Constraints

In addition, there is a number of constraints that govern the way in which the states are mapped from stage t to stage $t+1$ that are not present in the E-Cow simulation.

In order to prepare the cow for the next season, it needs to be above a certain body condition at the end of the season:

$$B_{53} \geq \zeta. \tag{4.6}$$

Once a cow has stopped being milked, it is unable to start milking until the next season (assuming it gets in-calf):

$$L_{t+1} \leq L_t. \tag{4.7}$$

Another constraint is that our herbage intake must be non-negative:

$$H_t^I \geq 0. \tag{4.8}$$

5 Solution Methods

5.1 Dynamic Program

A dynamic program (Bellman, 1954) requires a discrete state space. However, the body condition of a cow is continuous. Therefore it would have to be discretised. However, the BCS not an arbitrary number, but a physically constrained property of the cow. In the absence of nutritional constraints (such as a lack of feed), the BCS will follow a smooth trajectory. Large changes in our actions will not have large changes in the BCS, but instead nudge its trajectory slightly. Therefore, at a given stage, that instead of being anywhere between 2 and 4, the BCS is much more likely to between say, 2.7 and 2.9. This requires a very fine approximation of the state space around a given BCS 'trajectory' in order to accurately model the physical changes. However, since the optimal trajectory is not known ahead of time, this requires a fine discretisation of a large proportion of the state space.

When the model is extended to multiple mobs of cows, our state space increases exponentially as the discretisation would have to be repeated for each mob (the so called 'curse of dimensionality').

For these reasons, we have not considered solving this model as a dynamic program.

5.2 Time Staged Dynamic Program

Instead of discretising the BCS, we can solve the model as a time staged dynamic program. This allows the BCS to remain continuous. In this method, we 'chain' together each stage into a single Mixed Integer Linear Program (MIP).

This MIP will have the following form

$$\begin{array}{lrcll} \max & \sum\limits_{t\in T} R_t(\mathbf{x}_t, \mathbf{a}_t) & & & \\ \text{s.t.} & F_t\left(\mathbf{x}_t, \mathbf{a}_t\right) & = & \mathbf{x}_{t+1} & \forall t \in T \\ & B_{53} & \geq & \zeta & \\ & T = 1, 2, & \ldots & , 52 & \end{array} \tag{5.1}$$

where ζ is the target BCS in the final period.

The downside of this method is that it becomes a very large mixed integer program. In each stage there are eight SOS2. This means that in the final problem there are 416 SOS2. As many of SOS2's interact with each other, both in a single stage and across stages, many branching decisions will need to be made in order to solve the MIP.

5.3 Dual Dynamic Program

Dual Dynamic Programming (DDP; Pereira and Pinto, 1991) is an solution technique that removes the need to discretise the state space. This means that the method does not suffer the 'Curse of Dimensionality' and can be used to solve large scale problems. DDP uses Benders Decomposition (Benders, 1962) to break the problem into a series of sub-problems, and uses a piecewise linear approximation of the value-to-go function.

A DDP consists of two separate events: a forward pass and a backward pass.

5.3.1 Forward Pass

In the forward pass, we solve the following MIP at each stage, passing $\mathbf{x}_t$ forward to the next stage. θ_{t+1} represents the value-to-go in stage t.

$$\begin{aligned} max \quad & R_t(\mathbf{x}_t, \mathbf{a}_t) + \theta_{t+1} \\ st \quad & F_t\left(\mathbf{x}_t, \mathbf{a}_t\right) = \mathbf{x}_{t+1} \\ & \mathbf{x}_t = \bar{\mathbf{x}}_t \qquad\qquad (\pi_t). \end{aligned} \tag{5.2}$$

We store the optimal values of $\mathbf{x}_t$ for use on the backward pass.

5.3.2 Backward Pass

The backward pass consists of solving the stage problems in reverse order, and using the dual information to add a linear cut to the value-to-go function in stage $t-1$.

However, given the fact that the single stage MIP contains integer variables, we are unable to calculate the dual values as we would in a linear program.

Duality of an integer program To overcome this, we can fix the integer variables to the values that were solved on the forward pass, and then resolve the problem as a linear program, calculating the duals in the standard way. The slight complication is that we are dealing with SOS2 instead of integer variables. However, Gurobi fixes SOS2 by adding constraints to force all the zero elements in the set to remain at zero, leaving the two non-zero elements free.

The following linear cut can then be added to the t^{th} stage:

$$\theta_{t+1} \leq \bar{\theta}_{t+1} + \pi_{t+1}^{\mathrm{T}}(\mathbf{x}_{t+1} - \bar{\mathbf{x}}_{t+1}) \tag{5.3}$$

where $\bar{\theta}_{t+1}$ is the objective of the stage $t+1$ MIP, π_{t+1} are the dual variables calculated on the backward pass, and $\bar{\mathbf{x}}_t$ are the solved values of the state variables that were stored on the forward pass.

5.3.3 Convergence

This process of repeated forward simulation passes, and corrective backward passes continues until the problem has converged. Convergence occurs when the objective of the first stage problem (upper bound: z_{upper}) is close enough to the simulated value of the policy (lower bound: z_{lower}):

$$\frac{z_{upper} - z_{lower}}{z_{upper}} < \varepsilon.$$

Since our objective is purely an economic objective, we shall terminate the algorithm when our upper and lower bounds are less than 1¢ apart.

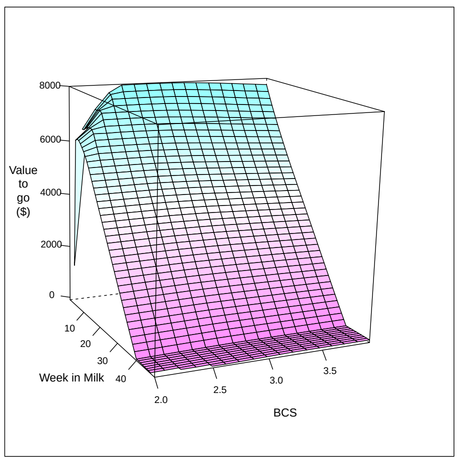
5.3.4 Requirements

Concave value-to-go function

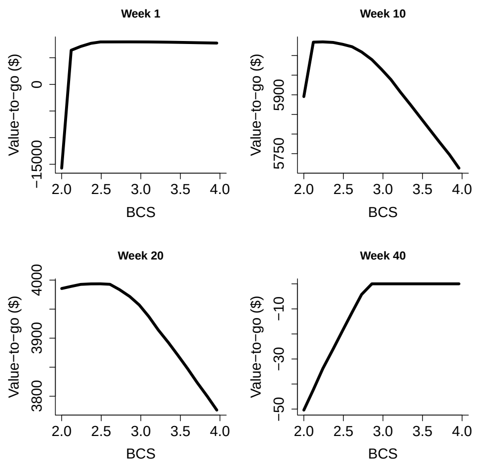
As the value-to-go function is approximated by a series of piecewise linear cuts, the DDP requires a concave value-to-go function. However, given the modelling approximations that we have made, and the introduction of the SOS2, it is no longer certain that this is the case. We can sketch the value-to-go surface by repeatedly running the MIP with different starting parameters at different points in time. It appears (Figure 5.1) that the value-to-go function at a given stage is concave since cross-sectional slices through the surface at different stages are all concave, and the surface appears well behaved. It also makes logical sense that there should be some optimum BCS in a given stage: a smaller BCS will make it more expensive to meet the target BCS in the final stage, while a greater BCS requires more energy to maintain itself that instead could have been turned into milk.

Global Feasibility

Another requirement of the DDP algorithm is global feasibility.



(a) Wireframe



(b) Cross section at selected weeks

Figure 5.1: Value-to-go function given a maximum supplement of 5kg/cow/day.

Final BCS: The constraint that forces the final body condition to be greater than some value can result in the model being infeasible if, in the 51st stage, the body condition is very low. Therefore, we relax this constraint by adding a penalty variable with a large negative objective cost.

This gives the constraint

$$B_{53} + z \geq \zeta \tag{5.4}$$

where z is a non-negative variable.

We create a new objective function for the 52nd stage where M is large:

$$R_{52}(\mathbf{x}_{52}, \mathbf{a}_{52})' = R_{52}(\mathbf{x}_{52}, \mathbf{a}_{52}) - Mz. \tag{5.5}$$

SOS2: SOS2 require the declaration of a set **a** such that any value being approximated can be made of a convex combination of two adjacent members of the set. If this is not true, then the model can become infeasible. Therefore, **a** should be initialised so that the domain of the SOS2 is larger than the domain of the variable being approximated.

5.3.5 Speed Improvements

Initial iterations of the DDP often consider solutions that are clearly sub-optimal. Such cases include high (> 4) and low (< 2) body condition scores. In the high BCS case, the cow is at risk of negative health outcomes such as acidosis, while a low BCS is associated with malnourished and poor fertility. Therefore, we can guide the DDP away from these solutions by adding relaxed upper and lower bounds on the BCS in every stage:

$$B_t + z_0 \geq 2 \qquad B_t - z_1 \leq 4. \tag{5.6}$$

With a new objective function for t^{th} stage where M is large

$$R_t(\mathbf{x}_t, \mathbf{a}_t)' = R_t(\mathbf{x}_t, \mathbf{a}_t) - M \times (z_0 + z_1). \tag{5.7}$$

This also helps the feasibility of the DDP as the DDP attempts to keep the BCS within a normal range throughout the season, rather than reaching an infeasible point late in the season. If it is apparent that an optimal solution is affected by these bounds (such as the BCS flat-lining along $BCS = 2$ before increasing again), they can be removed or altered.

6 Method Comparison

This chapter analyses the two different methods. The models were run with the parameters found in Table 6.1.

H^A	H^N	H^E	S^E	W_1	B_1	C^M	C^S	L_{41}
220	0.44	10.3	10.3	480	3.2	6.5	400	0

Table 6.1: Model Comparison Parameters.

6.1 Validation

The first step was to validate each method. This was done by running the optimisation, then taking the optimal policy and using that as the input to the non-linear simulation. Differences between the optimisation model and the simulation indicate that the optimisation model is a bad approximation of the non-linear simulation.

6.1.1 MIP

The MIP was solved with a SOS2 MRPE of 5% (Table 3.2). It is likely that increasing the number of SOS2 divisions will increase the accuracy of the MIP, however as outlined in Section 6.2, this makes the model hard to solve. Figure 6.1 shows a plot of the time staged MIP solution.

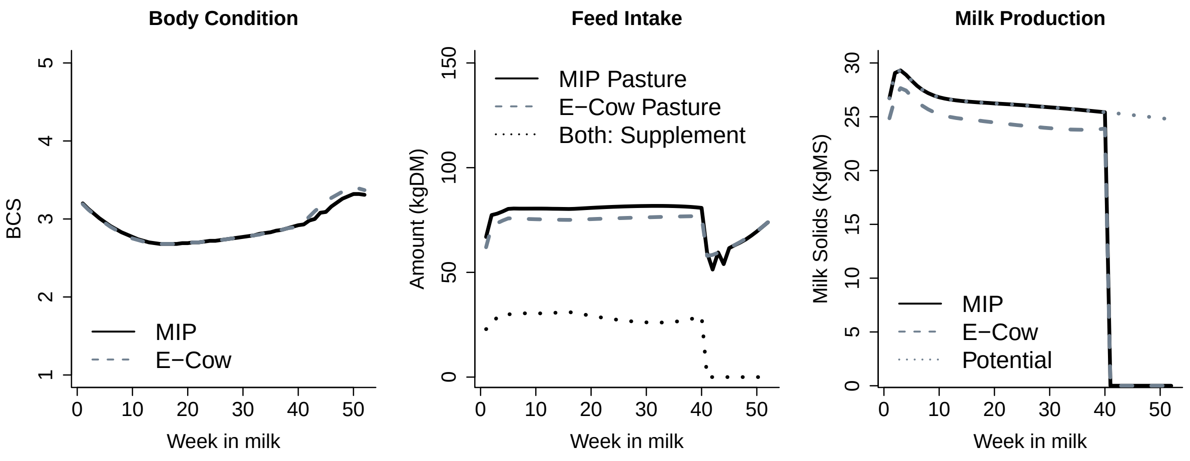


Figure 6.1: Solution obtained via Time Staged MIP.

The left graph, *Body Condition*, shows the body condition score against week in milk for both the MIP and E-Cow, assuming the optimal policy was followed. The middle graph, *Feed Intake*, displays the feed intake of the cow in KgDM per week. The solid line is the predicted pasture intake from the MIP and the dotted line shows the optimal amount of supplement to feed each week. The other line (dashed) is the predicted pasture intake from E-Cow, if the optimal policy was followed. The right graph, *Milk Production* displays the total milk solids produced (milkfat + protein) each week. The dotted line is the maximum potential production, the solid line is the predicted output from the MIP and the dashed line is the predicted output from the e-cow simulation, assuming the optimal actions were followed

There are a number of differences between the MIP and the E-Cow simulation, indicating that the MIP with a MRPE of 5% is not a good approximation of E-Cow. During the milking period, the MIP overestimates the amount of pasture that the cow will eat. This leads to overestimating the level of milk production. Therefore the MIP returns a policy that is sub-optimal. During lactation, the BCS of the MIP matches E-Cow closely. Once the cow has stopped milking, the MIP underestimates the BCS. It is likely that increasing the number of SOS2 divisions (i.e. reducing the MRPE) will make the MIP more accurate.

LP relaxation

Due to the SOS2 in the MIP, the model is computationally expensive to solve. By relaxing the SOS2 constraints, the MIP can be turned into the LP relaxation. This is much easier to solve. Therefore, if the LP relaxation gives a solution that is close to the MIP solution, then it may be advantageous to trade a slight decrease in accuracy for a large reduction in solution time.

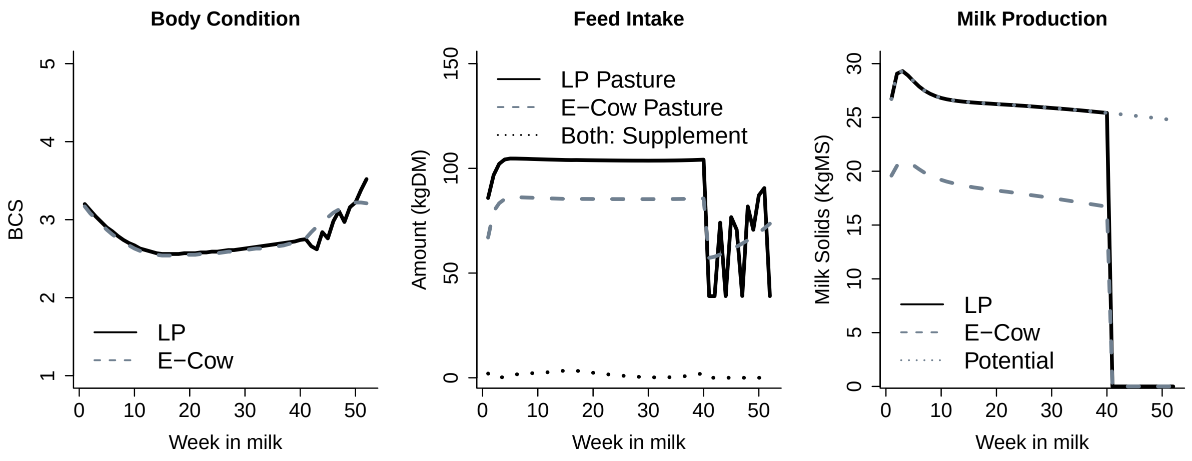


Figure 6.2: Solution obtained via Time Staged LP relaxation.

However, as Figure 6.2 shows, the LP relaxation is a poor approximation of the E-Cow simulation. Therefore, the LP relaxation should not be used to generate solutions.

6.1.2 DDP

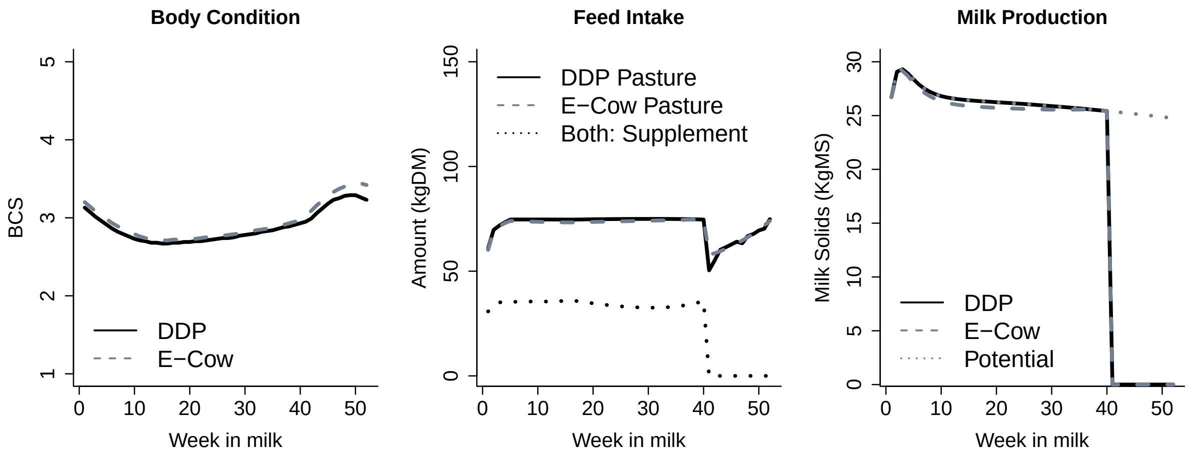


Figure 6.3: Solution obtained via DDP.

In contrast to the MIP, where all the stages are chained together, the DDP solves each stage independently. This allows a much greater number of divisions in all the SOS2, allowing a MRPE of 0.1% (increasing the number of divisions even further was not found to have a noticeable impact on the solution).

During lactation, the DDP slightly underestimates the BCS, and overestimates both the pasture intake, and milk production. Once the cow is dried off, the DDP underestimates both the BCS and pasture intake. This is likely because the cow will meet the BCS target with no supplement being fed. Therefore, the assumption that $\hat{H}_t^p$ will always maximise to be on the surface is incorrect.

However, this is not an issue as it will not cause our policy to change, or the final objective value. This is because, if we were feeding supplement, we could reduce the amount of supplement being fed. Since $\hat{H}_t^p$ is not on the surface, it can increase to replace the lost energy. This increases our objective and pushes $\hat{H}_t^p$ to the upper bound. If $\hat{H}_t^p$ is not on the surface during lactation, we must also be producing at the maximum level of milk production. Otherwise we could increase $\hat{H}_t^p$, leading to an increase in energy intake. This additional energy will be converted into milk production and our objective will increase. Therefore, if $\hat{H}_t^p$ is not on the surface, we must be producing at the maximum level of milk production and feeding zero supplement. The results of the DDP support this conclusion as $\hat{H}_t^p$ only pulls away from the surface once we have stopped milking, and no longer feed supplement.

6.1.3 Simplification Check

In Section 4.2.3, we simplified the model by removing E_t^m as a state variable. In Figure B.1, we see a plot of E_t^m against t. There is close agreement between the E-Cow values and the DDP values. This fact, combined with the general agreement in other variables validates this simplification.

6.2 Solution Time

Each model was solved using 32bit GurobiPy (Gurobi Optimization, 2014) on a Windows 7 i5 (quad-core) with 16GB of RAM (so only 4GB addressable). The total solution time (model creation + solve) was recorded. This was repeated four times, with the number of SOS divisions determined by the number required to achieve a given MRPE (Table 3.2). The models were solved with the parameters given in Table 6.1.

6.2.1 Results

Figure 6.4 shows a plot of solution time against MRPE for each of the methods.

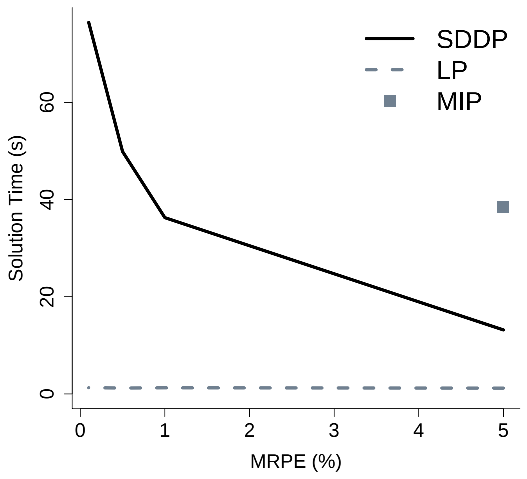


Figure 6.4: Solution time against MRPE.

DDP As the MRPE decreased, the DDP solution time increased exponentially. The DDP was able to solve the model with a MRPE of 1% in roughly the same time as the MIP took to solve the problem with a MRPE of 5%.

MIP The time staged MIP could only be solved at a MRPE of 5%. For smaller MRPEs, Gurobi was unable to solve the model with the RAM available.

LP The LP relaxation of the time staged MIP solved quickly ($\approx 2s$) across all MRPEs. There was little variation in solution time. This is expected as the LP relaxation discards the condition that the SOS2 set can contain at most two non-zero variables, and if it does, they must be adjacent.

6.3 Conclusion

The time staged MIP is unsuitable to use as a model as to solve the MIP in any reasonable amount of time, a large amount of accuracy must be sacrificed. In contrast, the DDP is able to solve with a low MRPE ($< 0.1\%$) in less than two minutes.

6.4 Comparison to Simulated Policies

In Section 2.3 we used the E-Cow to evaluate five simple policies. It was found that the best policy was to feed the cow 5kgDM of supplement per day, when we milked the cow. When the DDP is run with the same parameters, the optimal policy is similar (Figure 6.5).

The cow meets the target BCS in both cases, but by a smaller amount in the DDP (Table 6.2). In total, the simulated policy fed the cow 1400kgDM of supplement over the entire season. In comparison, the DDP policy fed 1370kgDM. This reduction in supplement purchased, and targeted feeding at different points in the season led to a $111.68 increase in our objective.

Policy	Obj. ($)	Final BCS
5kg/day while milking	6,168.84	3.28
Optimisation	6,280.52	3.23

Table 6.2: Comparison between most profitable simulated policy and optimal solution ($10,000/unit BCS penalty of missing final BCS target of 3.2).

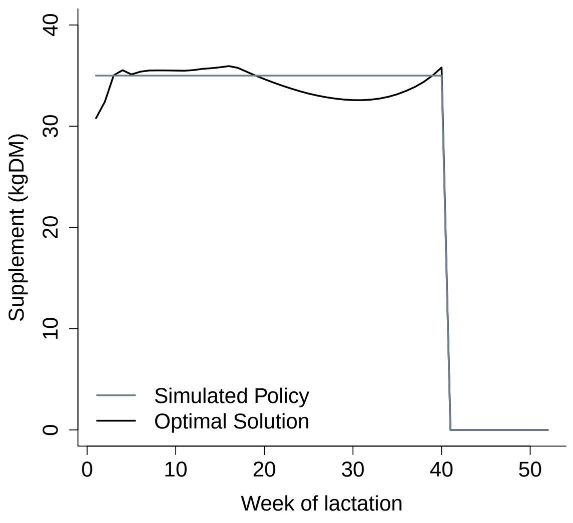


Figure 6.5: Simulated and optimal supplementation policies.

7 DDP Extension

7.1 Motivation

In the model discussed previously, we chose whether to milk the cow or not in each stage prior to solving the model. This meant that L_t was a fixed parameter in the DDP. However, in order to turn this into a variable in the DDP we run into a problem.

The DDP algorithm requires that the value-to-go function in each stage is concave over the the relevant domain. Therefore, in standard DDP, the inclusion of a binary variable as a *state* variable will cause the method to fail as this is no longer the case. If we relax the binary constraint on the milking variable, then we find that the optimal solution is to milk fractional amounts of the cow in each time step – something we clearly wish to avoid.

The chapter outlines a proposed method for adjusting the Benders cuts so that the DDP algorithm can be used with a binary state variable.

7.2 Literature

The author is aware of one other attempt to incorporate binary variables in a DDP (Newham, 2008). In that thesis, Newham introduces the idea of a *dynamic constraint.* This is a constraint that is updated each forward pass, so that it is not persistent from pass to pass. When this idea was implemented in MOO, the idea was found to be flawed.

If, in a given stage, the optimal solution is to dry off (due to a bad approximation of the value-to-go function), then a Bender's cut is added to the previous stage with the observed value to go, $\bar{\theta}$ less than or equal to zero. If the true optimal solution is to keep milking, then the true value-to-go function will be larger than this (since the milk can be sold for profit). Therefore, the Bender's cuts do not represent an outer approximation to the value-to-go function, and the model either converges at a sub-optimal solution, or fails to converge as the upper bound decreases below the lower bound (due to the incorrect value-to-go function).

However, despite the incorrect value-to-go function, if we change our convergence criteria to Newham's, the model converges to a solution that appears close to optimality. Newham's test for convergence is to check for a change in any of the state variables after each forward pass. If none of the state variables have changed value in any stage since the last forward pass, the algorithm is terminated.

We are unaware of the mechanism by which the method converged, and we are unsure if a similar phenomenon happened in Newham's case study (whereby invalid cuts are added to the model, but the model converges to the correct solution anyway). Although seemingly successful, this method was disregarded as a solution due the clear addition of invalid Bender's cuts and no mechanism by which to explain why it works.

7.3 The Problem

To make the proposed method easier to understand, we shall consider a slightly different stage problem:

$$\begin{array}{lrcll}
\text{maximise} & C_t(x_t, y_t, a_t) & + & \theta_{t+1} & \\
\text{subject to} & x_t & = & \hat{x}_t & [\pi_t^x] \\
 & y_t & = & \hat{y}_t & [\pi_t^y] \\
 & F_t\left(x_t, y_t, a_t\right) & = & x_{t+1} & \\
 & y_{t+1} & \leq & y_t & \\
 & x_{t+1} & \in & [x_{min}, x_{max}] & \\
 & y_{t+1} & \in & \{0, 1\}. &
\end{array} \tag{7.1}$$

In a given stage t, θ_{t+1} represents the value-to-go of all future stages. We also have some control variables a_t that change our state variables x_t. $\bar{x}_t$ and $\bar{y}_t$ are constants in the given stage problem and are the solved values of x_t and y_t from the forward pass. In the standard DDP algorithm, some large upper bound is declared such that $\theta_{t+1} \leq M$. There is also π_t which represents the rate of change in θ_{t+1} with respect to the incoming state variables.

7.4 Modifying the Bender's Cut

The standard Bender's cut would be

$$\theta_{t+1} \leq \bar{\theta}_{t+1} + \pi^x_{t+1}(x_{t+1} - \bar{x}_{t+1}) + \pi^y_{t+1}(y_{t+1} - \bar{y}_{t+1})$$

where $\bar{x}_{t+1}$ and $\bar{y}_{t+1}$ are the observed values of the state variables that were solved on the forward pass and $\bar{\theta}_{t+1}$ is the solved objective of stage $t+1$ in the backward pass.

However, in order to calculate the dual variables, we fix the integer variables in the model and resolve it as a linear program. This correctly calculates the π^x_t dual variables, but incorrectly calculates the π^y_t duals (as $y_{t+1} - \bar{y}_{t+1} = 0$ when y_{t+1} is fixed) and so the change in future profit is not computed. This leads to cuts being added to the problem that remove part of the inner region of the value-to-go function, causing the DDP to converge to a sub-optimal solution.

Therefore, we propose modifying π^y_{t+1} so that we can be confident of convergence to the global optimum in all cases.

7.4.1 Modified Cut

Assume we have some solution to the t^{th} stage problem $(x_{t+1}, y_{t+1}) = (\bar{x}_{t+1}, 0)$ from the forward pass, and values of $\bar{\theta}_{t+1}$ and π_{t+1} from stage $t+1$ of the backward pass. We now wish to calculate a value for π^y_{t+1} such that the cut is valid for all x_{t+1} when $y_{t+1} = 1$.

Case 1: If π^x_{t+1} is positive, we need to make sure that the plane intersects the point $(x_{min}, 1)$ at a value greater than M for all state variables. Therefore, the equation for the plane would be

$$\theta_{t+1} \leq \bar{\theta}_{t+1} + \pi^x_{t+1}(x_{t+1} - \bar{x}_{t+1}) + \left[M + max\left\{0, \pi_{t+1}(\bar{x}_{t+1} - x_{min})\right\} - \bar{\theta}_{t+1}\right] \times y_{t+1}.$$

Case 2: If π^x_{t+1} is negative, we need to make sure that the plane intersects the point $(x_{max}, 1)$ at a value greater than M. Therefore, the equation for the plane would be

$$\theta_{t+1} \leq \bar{\theta}_{t+1} + \pi^x_{t+1}(x_{t+1} - \bar{x}_{t+1}) + \left[M + max\left\{0, \pi_{t+1}(\bar{x}_{t+1} - x_{max})\right\} - \bar{\theta}_{t+1}\right] \times y_{t+1}.$$

Let us define a new variable

$$x'_{t+1} = \begin{cases} x_{min} & \text{if } \pi_{t+1} \geq 0 \\ x_{max} & \text{if } \pi_{t+1} < 0 \end{cases}.$$

The two cases can be combined into a single cut:

$$\theta_{t+1} \leq \bar{\theta}_{t+1} + \pi^x_{t+1}(x_{t+1} - \bar{x}_{t+1}) + \left[M + max\left\{0, \pi_{t+1}(\bar{x}_{t+1} - x'_{t+1})\right\} - \bar{\theta}_{t+1}\right] \times y_{t+1}.$$

We can then make a similar argument for the case when $\bar{y}_t = 1$ to get the following cut:

$$\theta_{t+1} \leq \bar{\theta}_{t+1} + \pi^x_{t+1}(x_{t+1} - \bar{x}_{t+1}) + \left[M + max\left\{0, \pi_{t+1}(\bar{x}_t - x'_t)\right\} - \bar{\theta}_{t+1}\right] \times (1 - y_{t+1}).$$

Final Cut: These can be combined into a single cut

$$\begin{aligned}\theta_{t+1} \leq\ & \bar{\theta}_{t+1} + \pi^x_{t+1}(x_{t+1} - \bar{x}_{t+1}) \\ & + \left[M + max\left\{0, \pi_{t+1}(\bar{x}_{t+1} - x'_{t+1})\right\} - \bar{\theta}_{t+1}\right]\left[\underbrace{y_{t+1}(1 - \bar{y}_{t+1})}_{\bar{y}_{t+1}=0} + \underbrace{(1 - y_{t+1})\bar{y}_{t+1}}_{\bar{y}_{t+1}=1}\right].\end{aligned} \qquad (7.2)$$

7.4.2 Reasoning

The dual variable corresponding to the binary variable can be modified in such a manner because the interior of the continuous domain ($y_{t+1} \in (0,1)$) is unimportant since the binary constraint in the stage problem will force the y_{t+1} variable to an extreme point. These cuts build up the outer approximation of the value-to-go function at each value of y_{t+1} while ensuring that the planes do not intersect the true value-to-go function at the opposite value of y_{t+1}.

7.4.3 Apparent Complication

Notice that we completely disregard all *true* dual information concerning the binary variable and force the 'fake' dual to be positive in the case that $y_{t+1} = 0$ and negative in the case that $y_{t+1} = 1$. This seems like it will cause the method to fail as, if we are at $\bar{y}_{t+1} = 0$ and if the true dual is negative, then in the next pass y_{t+1} will switch to 1, even though this is an incorrect thing to do.

We can show that this will still result in convergence to the global optimum with the following argument:

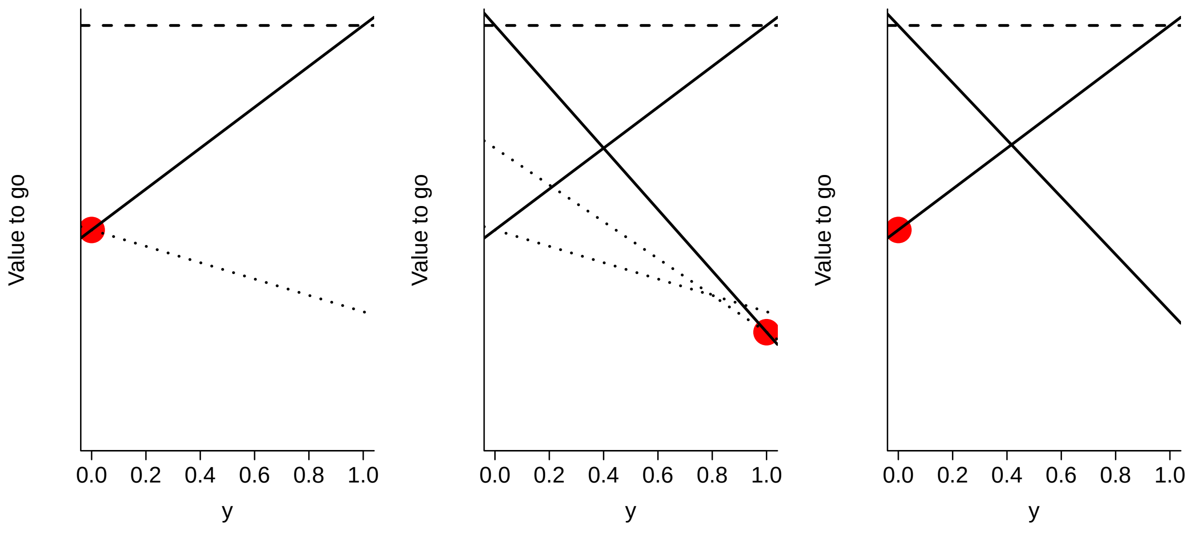


Figure 7.1: DDP Modification. Dotted lines represent the cuts that would have been added if we knew the 'true' duals.

Assume we find a solution $\bar{\theta}^1_{t+1}$ at $(\bar{x}^1_{t+1}, 0)$, and that no cuts have been added yet. If the *true* value of the dual was positive, then in the next pass the binary would switch to 1. As such adding the cut will have no negative effect on the decision. This is because the 'fake' dual will be greater than the true dual. So the cut is still an upper bound on the value-to-go function. However, if the *true* value of the dual was negative, we still add a cut with a large positive *fake* dual. The next pass, the binary variable switches to 1. But, because a cut that used the *true* dual would represent an outer approximation to the true value-to-go function, $\theta^{\bar{2}}_{t+1}$ will be less than $\theta^{\bar{1}}_{t+1}$. The modified cut that gets added will cause the binary to switch back to the value 0 in the next pass. This requires more passes of the DDP algorithm to converge, but will not affect the converged solution. The same argument is true for the case when $\bar{y}^1_{t+1} = 1$. This sequence of cuts is shown in Figure 7.1.

7.5 Implementation

We have shown in the section above how to modify the Bender's cuts so that we can build up an approximation of the value-to-go function in each case of the binary variable so that they do not interfere with each other.

We can now use these modified cuts, and the monotonically decreasing property of L_t to create a two-phase solution method that will solve this model.

7.5.1 Phase I

Consider the case in which the cow is dried off initially. Given the monotonic property of L_t, this forces the cow to remain dried off throughout the season. Therefore, the binary variable is effectively transformed into a parameter. This means that, as before, the DDP can be run until convergence, constructing a good approximation of the value-to-go function for the dried off case. We also note that the value-to-go function in the dry case can never be greater than zero, since we are not producing any milk to sell.

7.5.2 Phase II

Now allow the cow to begin milking. On the first pass, the cow will be milking in every stage, since no cuts have been added to the milking case (so the value-to-go function has a large upper bound), and the value-to-go of the dried off case must be less than, or equal to, zero.

On the first backward pass, cuts will only be added to the milking state, and are upper bounds on the true value-to-go function. On the next forward pass, if, at any stage, the upper bound of the milking value-to-go function is less that the converged dried off value-to-go function, then the optimal decision must be to stop milking. On future forward passes, the value of the binary variable will transfer permanently to the dried off state at this point (if it has not already). This then repeats until convergence.

7.5.3 WJustification

In the second set of passes, the DDP will never sample a stage in the dried off state if the optimal decision is to remain milking.

This is because the value-to-go function of the dried off case has been evaluated in the first set of passes, and so if the upper bound of the milking state is less than the converged value-to-go function, then the true value of the milking state must also be less than it. Therefore, the optimum decision is to minimise the loss and dry off.

This feature prevents invalid cuts being added to the model that might compromise convergence.

Converged plots of selected stages with the Bender's cuts added can be found in Appendix C.

8 Modified DDP Results

In this chapter we present some initial results aimed at showing how MOO with a variable dry off week can be used, and the behaviour that can be expected. Unless otherwise stated, the following parameters were used in the model:

H^A	H^N	H^E	S^E	W_1	B_1	C^M	C^S
220	0.44	10.3	10.3	480	3.2	6.5	400

Table 8.1: Parameters used in modified DDP results.

Assumptions The herd could be fed, on average, no more than 14kg of supplement each week. This assumption was made so that there are differences in the lactation lengths of different groups; higher levels of supplementation are able to sustain milking cows to the end of the season. In contrast, limited supplementation will cause the optimal dry off policy to be different for different initial body conditions.

8.1 Two Mobs

In this section we compare the optimal lactation length for a herd consisting of two mobs, given different starting conditions. Group 2 was dried off either at the same time as Group 1, or earlier.

8.1.1 Lactation Length

Figure 8.1 shows the lactation length (in weeks) of each mob given a range of different starting BCSs. Mob 2, which was constrained to be dried off first, has a monotonic trend between lactation length and initial BCS; as the BCS of a given mob increased, the optimal lactation length either increased, or stayed the same.

The same is not true of Mob 1. Instead, the relationship depends on the lactation length of Mob 2 as well as the starting BCS of each mob. For a given lactation length of Mob 2 (i.e. 42 weeks), the lactation length of Mob 1 is monotonically increasing with increasing BCS of either group. If the lactation length of Mob 2 changes with a given change in BCS (such as increasing the BCS of Mob 1 from 3 to 3.1 while holding the initial BCS of Mob 2 constant at 2.8) then the lactation length of Mob 1 may decrease. This is because there is a greater potential for milk earlier in the season. It is cost effective to reduce the lacation length of Mob 1 by two weeks (from 46 to 44), in order to divert supplement to Mob 2, allowing it to milk an extra week.

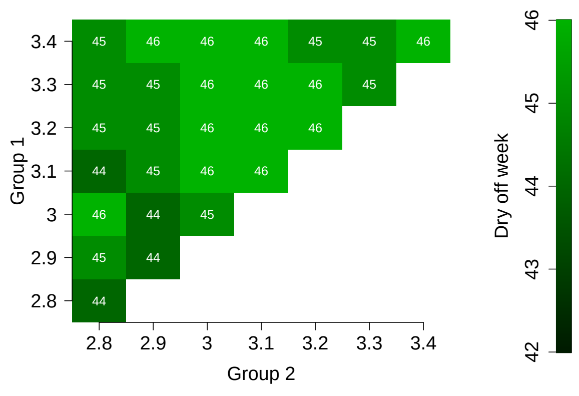
8.1.2 Objective

Figure 8.2 shows a plot of the objective against the starting BCS for each group. The least money is made when both groups have a starting BCS of 2.8 (skinny). If the starting BCS of Group 1 is fixed, then the objective increases monotonically with increasing starting BCS of Group 2. The same is true if the starting BCS of Group 2 is fixed and the starting BCS of Group 1 is varied.

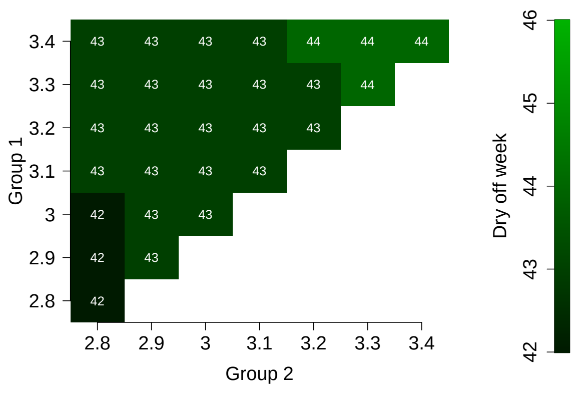
This result is a good validation that the modified DDP correctly handles the dry of decision variables correctly.

8.2 Multiple Mobs

In this section, the herd was divided into different numbers of identical groups to investigate if having the ability to construct a different policy for each group is advantageous.



(a) Group1



(b) Group2

Figure 8.1: Optimum length of lactation assuming Group 1 is milked longer than Group 2.

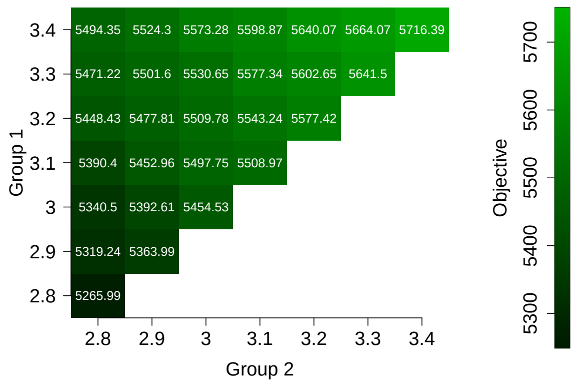


Figure 8.2: Maximum return given different starting BCS and assuming that Group 1 is milked longer than Group 2.

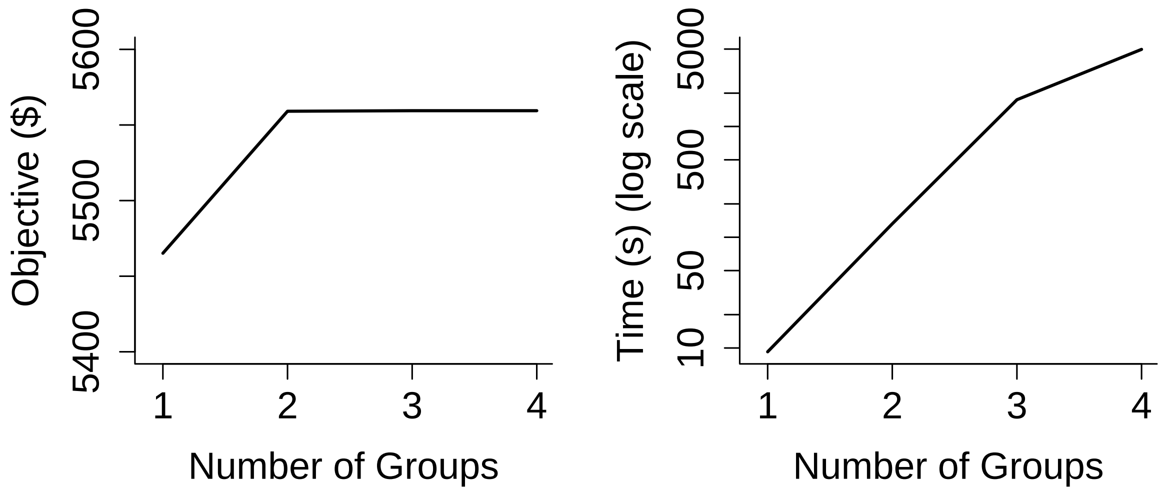
As the number of groups increases, the objective also increases (Figure 8.3a). This is because in the first case (a single group), there is only only one decision to make, However, as the number of groups increases to two, a better policy is to dry off one mob earlier, in order to keep milking the other mob longer. There is no extra benefit in dividing the herd into three or four mobs, compared to dividing it into two mobs.

As the number of groups increases, the model takes longer to solve (Figure 8.3b). For less than four groups, the trend is exponential (linear relationship in semi-log y graph). However, adding a fourth group does not continue the trend. The reason for this is likely explained by the objective results. As the number of groups increases, more complicated decisions can be made surrounding the binary lactation variable. However, there is no added benefit in splitting the herd into four mobs, when compared to three. Therefore, the problem is still harder than the three mob case (demonstrated by the extra solution time), but not as hard as potentially expected, should the trend have continued.

It may be the case that as the number of mobs tends towards the number of cows (i.e. each cow has an independent milking decision), it is possible to increase the objective even further. However, the extra computation required outweighs any minimal increase.

8.3 Comments

One factor that should be taken into consideration when interpreting these results is the weekly timestep. If we consider the dry off decisions for Mob 2 (Section 8.1.1), there is very little variation (42 - 44 weeks). It may be the case, that if we reformulated the model with a daily timestep, the gradient would be



(a) Objective. (b) Solution Time.

Figure 8.3: Objective and Solution Time for varying numbers of groups.

more gradual. This would allow a finer balance between the two groups, and there may not be the non-monotonic behaviour in the dry off decision of Group 1. Evidence for this can also be seen in Section 8.2. When dealing with a herd consisting of two identical mobs, the optimal solution is to treat them differently. This is likely because the optimal solution in continuous time is to dry them off mid-way through a week. Therefore, half dry off earlier, and half dry off later.

9 Future Work

9.1 Stochasticity

9.1.1 Pasture

Mills (2013), adds uncertainty to his model by modelling rainfall as a markov process, and linking rainfall to pasture growth. Although he models only two states (wet and dry), this could be extended to any number of pasture growth states (i.e. wet, normal and dry). The DDP can then easily be extended to a stochastic dual dynamic program (SDDP).

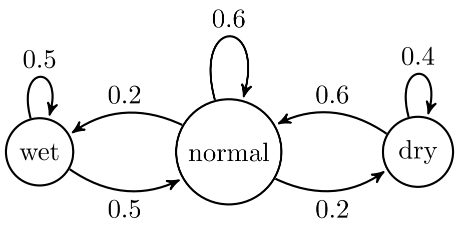


Figure 9.1: Potential Markov chain representing different weather states.

The Markov process can also be stage dependent, so that there is a higher probability of transitioning into the dry state during summer than during spring. This allows a finer approximation of typical weather patterns than stage independent transition probabilities.

9.1.2 Price of supplement

Instead of the current model in which supplement is purchased on-demand at a spot price, an option pricing could be introduced. This is what the majority of farmers do; they purchase a large option cheaply at the start of the season, and only purchase extra on the spot market if they run out. This could be modelled by introducing a state variable representing the amount of supplement currently in storage. A decision in each state chooses how much additional supplement to purchase at that given time. The maximum amount of supplement that can be fed is then defined by the amount in storage at the beginning of the week. Again, this is a closer approximation of what really happens, as in its current form, MOO essentially buys from the spot market, and has it delivered instantaneously. In the real world, there is a delay to delivery, and farmers have to plan ahead.

9.2 Stocking rate

The stocking rate (SR) is defined as the number of cows grazed per hectare. A low SR reduces pressure on resources and the environment, allowing each cow more grass to graze. In contrast, a high SR increases the pressure on resources, reducing the amount of grass each cow can graze. The trade off is that for a given farm, as the number of cows increases, the potential milk production of the herd increases, but the actual milk production of each cow decreases. Therefore, there must be some optimum SR which maximises the profit of the farm. Studies to measure the optimal SR have been conducted in farmlet trials (Macdonald et al., 2001, Macdonald et al., 2008, Macdonald et al., 2011). All found that the efficiency of pasture utilisation and kgMS/Ha increased as SR increased, but that the kgMS/cow decreased. There is general agreement that a SR around 3.2-3.3 was the most profitable SR. MOO is an ideal tool to explore the effect of different SR without needing to undertake costly, muti-year trials.

10 Conclusions

This report presents the development and testing of MOO (the Milk Output Optimiser). MOO is a DDP based optimisation program developed in the Gurobi python modelling language.

MOO is based on E-Cow, a non-linear simulation developed at Massey University in New Zealand. The equations in E-Cow were linearised using techniques such as Special Ordered Sets of Type II, linear regression and outer approximation by tangent plane. These equations are aggregated into weekly time-steps. These approximations were validated against the non-linear simulation with a daily time-step and found to be good approximations.

Extensions were made to the DDP algorithm, and the structure of the problem exploited, in order to incorporate binary decision variables into a DDP. This was done by modifying the Bender's cuts so that outer approximations of the value-to-go function could be made in each state of the binary variable, without a cut in one state causing an invalid approximation in the other. Secondly, the monotonically decreasing property of the binary variable was used to converge the dry state value-to-go function before any cuts are made to the milking state. Therefore, forward passes of the DDP never sample a state which would result in an invalid cut being added to the model.

In conclusion, MOO shows promise as a useful tool for practical farm management of both supplementary feed, and lactation length.

References

Bath, D. and L. Bennett (1980). “Development of a Dairy Feeding Model for Maximizing Income Above Feed Cost with Access by Remote Computer Terminals”. In: *Journal of Dairy Science* 63 (8), pp. 1379–1389.

Baudracco, J. (2011). “Effects of feeding level and genetic merit on the efficiency of pasture-based dairy systems: field and modelling studies”. PhD thesis. Palmerston North, New Zealand: Massey University.

Baudracco, J., N. Lopez-Villalobos, and M. Zamateo (2008). *E-Cow: Simulation of a dairy cow’s response.* [Online; accessed 2014-07-22]. URL: `http://e-cow.net`.

Baudracco, J. et al. (2010). “Prediction of herbage dry matter intake for dairy cows grazing ryegrass-based pastures”. In: *Proceedings of the New Zealand society of Animal Production* 70, pp. 80–85.

Baudracco, J. et al. (2011). “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”. In: *Animal* 6 (6), pp. 980–993.

— (2012). “e-Dairy: a dynamic and stochastic whole-farm model that predicts biophysical and economic performance of grazing dairy systems”. In: *Animal* 7 (5), pp. 870–878.

Beale, E.M.L. and J.A. Tomlin (1970). “Special Facilities in a General Mathematical Programming System for Non-convex problems using ordered sets of variables”. In: *Proceeding of the Fifth International Conference on Operational Research*, pp. 447–454.

Bellman, R. (1954). “The Theory of Dynamic Programming”. In: *Bulletin of the American Mathematical Society* 60 (6), pp. 503–515.

Benders, J.F. (1962). “Partitioning procedures for solving mixed-variables programming problems”. In: *Numerische Mathematik* 4, pp. 238–252.

Bryant, J. (2006). “Quantifying genetic variation in environmental sensitivity of New Zealand dairy cattle to apply in the development of a dairy cattle simulation mdoel for pastoral systems”. PhD thesis. Palmerston North, New Zealand: Massey University.

Dairy NZ (2008a). *DairyNZ Economic Survey 2007-08.* Dairy NZ, p. 34.

— (2008b). *Palm Kernel Extract.* Farm fact 1-71. Dairy NZ.

— (2013). *DairyNZ Economic Survey 2012-13.* Dairy NZ, p. 44.

— (2014a). *Common Feed Supplements.* [Online; accessed 2014-07-24]. URL: `http://www.dairynz.co.nz/feed/supplements/common-feed-supplements/`.

— (2014b). *Supplement Price Calculator.* [Online; accessed 2014-07-22]. URL: `http://www.dairynz.co.nz/feed/feed-management-tools/supplement-price-calculator/`.

— (2014c). *The 5 Production Systems.* [Online; accessed 2014-04-02]. URL: `http://www.dairynz.co.nz/farm/farm-systems/the-5-production-systems/`.

— (2014d). *What is BCS?* [Online; accessed 2014-04-02]. URL: `http://www.dairynz.co.nz/animal/herd-management/body-condition-scoring/what-is-bcs/`.

Dairy NZ and LIC (2013). *New Zealand Dairy Statistics 2012-13.* Dairy NZ, p. 8.

Dantzig, G. (1990). “The Diet Problem”. In: *The Practice of Mathematical Programming* 20 (4), pp. 43–47.

Fonterra (2014a). *Farmgate Milk Prices.* [Online; accessed 2014-07-24]. URL: `http://www.fonterra.com/nz/en/Financial/Farmgate+Milk+Price`.

— (2014b). *Global Dairy Update: July 2014.* Global Dairy Update 22. Fonterra.

Friggens, N.C., K.L. Ingvartsen, and G.C. Emmans (2004). “Prediction of Body Lipid Change in Pregnancy and Lactation”. In: *Journal of Dairy Science* 87 (4), pp. 988–1000.

Gabriel, S.A. et al. (2006). “A practical approach to approximate bilinear functions in mathematical programming problems by using Schur’s decomposition and SOS type 2 variables”. In: *Journal of the Operational Research Society* 57, pp. 995–1004.

Global Dairy Trade (2014). *Product Results: Whole Milk Powder.* [Online; accessed 2014-07-24]. URL: `http://www.globaldairytrade.info/en/product-results/whole-milk-powder/`.

Gurobi Optimization (2014). *Gurobi Optimizer 5.6 Overview.* [Online; accessed 2014-07-22]. URL: `http://www.gurobi.com/products/gurobi-optimizer/gurobi-overview`.

Klein, K.K. et al. (1986). "Profit-Maximizing Linear Program Model for Dairy Cows". In: *Journal of Dairy Science* 69 (4), pp. 1070–1080.

Kristensen, T., J.T. Sorensen, and S Clausen (1997). "Simulated effect on dairy cow and herd production of different grazing intensities". In: *Agricultral Systems* 55, pp. 123–138.

Lin, L. (1989). "A concordance correlation coefficient to evaluate reproducibility". In: *Biometrics* 33, pp. 255–268.

Macdonald, K.A. et al. (2001). "Farm Systems - Impact of stocking rate on dairy farm efficiency". In: *Proceedings of the New Zealand Grassland Association* 63, pp. 223–227.

Macdonald, K.A. et al. (2008). "Effect of Stocking Rate on Pasture Production, Milk Production and Reproduction of Dairy Cows in Pasture-Based Systems". In: *Journal of Dairy Science* 91 (5), pp. 2151–2163.

Macdonald, K.A. et al. (2011). "Effect of Stocking Rate on the economics of pasture-based dairy farms". In: *Journal of Dairy Science* 94 (5), pp. 2581–2586.

Mills, P. (2013). *Optimal Dairy Farming Strategies During Drought.* Honours Thesis. Auckland, New Zealand: University of Auckland.

Newham, N. (2008). "Power System Investment Planning using Stochastic Dual Dynamic Programming". PhD thesis. Christchurch, New Zealand: University of Canterbury.

Paterson, B. (2014). *Why is Fonterra losing grip of markey share in the dairy industry.* [Online; accessed 2014-07-26]. URL: `http://www.infometrics.co.nz/Forecasting/ForecastArticle.aspx?id=31`.

Pereira, M.V.F. and L.M.V.G. Pinto (1991). "Multi-stage stochastic optimization applied to energy planning". In: *Mathematical Programming* 52, pp. 359–375.

Pesch, H.J. and M. Plail (2009). "The Maximum Principle of Optimal Control: A history of ingenious ideas and missed opportunities". In: *Control and Cybernetics* 38 (4A), pp. 975–995.

Pontryagin, L.S. (1987). *Mathematical Theory of Optimal Processes.* CRC Press.

Roche, J.R. et al. (2007). "Relationships among body condition score, body weight and milk production variables in pasture-based dairy cows". In: *Journal of Dairy Science* 90 (8), pp. 3802–3815.

Roche, J.R. et al. (2009). "Body condition score ad its association with dairy cow productivity, health and welfare". In: *Journal of Dairy Science* 96 (12), pp. 5769–5801.

Rotz, C.A et al. (1999). "A Dairy Herd Model for Use in Whole Farm Simulations". In: *Journal of Dairy Science* 82 (12), pp. 2826–2840.

Schilling, C., J. Zuccollo, and C. Nixon (2010). *Dairy's role in sustaining New Zealand.* NZIER.

Schills, R.L.M. et al. (2007). "DairyWise, A Whole-Farm Dairy Model". In: *Journal of Dairy Science* 90 (11), pp. 5334–5346.

Stigler, G. (1945). "The cost of subsistence". In: *Journal of Farm Economics* 27 (2), pp. 303–314.

Vetharaniam, L. et al. (2003). "Modelling the Effect of Energy Status on Mammary Gland Growth and Lactation". In: *Journal of Dairy Science* 86 (10), pp. 3148–3156.

A E-Cow

A.1 Equations

Selected equations. Others can be found in Baudracco (2011). All equations given here are for a daily time step. These were aggregated into a weekly time-step by either averaging their values to get a weekly average, or summing to get a weekly total.

Potential Milk Yield

$$pmy(t) = S \times (a_1 e^{b_1 t} + a_2 e^{b_3 t} + a_2 e^{b_3 t})/kl \tag{A.1}$$

Efficiency of utilisation of ME for milk synthesis

$$KL = 0.02MEdiet + 0.4 \tag{A.2}$$

Efficiency of utilisation of ME for fat deposition

$$KM = 0.02MEdiet + 0.5 \tag{A.3}$$

Efficiency of utilisation of ME for fat deposition

$$BCSperLW = (5 + B_0)/W_0 \tag{A.4}$$

Wilmink function for percentage milkfat and percentage protein

$$fat(t) = a + be^{-0.05t} + ct \tag{A.5}$$

$$prot(t) = a + be^{-0.05t} + ct \tag{A.6}$$

Stage of lactation

$$sol(t) = 0.67 + 0.0972 \times (4.0401 \times log_{10}(t) - 0.095 \times (t + 0.095)) \tag{A.7}$$

Energy Required for Maintenance

$$\begin{array}{rcl} \gamma_0 & = & \frac{1.4 \times 0.28 \times e^{-0.03 \times A}}{KM} \\ \gamma_1 & = & \frac{0.0025 \times (0.9 - H^D)}{KM} \\ \gamma_2 & = & \frac{0.05 \times 1}{(3+1.5) \times KM} \\ \gamma_3 & = & 0.1 \end{array} \tag{A.8}$$

Energy required for pregnancy Energy content gain of the gravid uterus (MJ/day)

$$mep(dp) = 0.025 \times CBW^2 \times 0.0201 \times e^{-5.76 \times 10^{-5} dp} \times 10^{151.665 - 151.64 \times e^{-5.75 \times 10^{-5} dp}}/(0.13 * 40) \tag{A.9}$$

where dp is the number of days pregnant

Change in LW due to pregnancy Weight gain of the gravid uterus (kg/day)

$$lwp(dp) = CBW \times (18.28 \times 0.02 - 2.86 \times 10^{-5} dp) \times e^{0.02dp - 1.43 \times 10^{-5} dp^2} \tag{A.10}$$

where dp is the number of days pregnant

Note: there is a typo in Baudracco, 2011, page 154. The formula for LWp is for grams liveweight gain per day, rather than kg/day as stated.

A.2 Simulated Policies

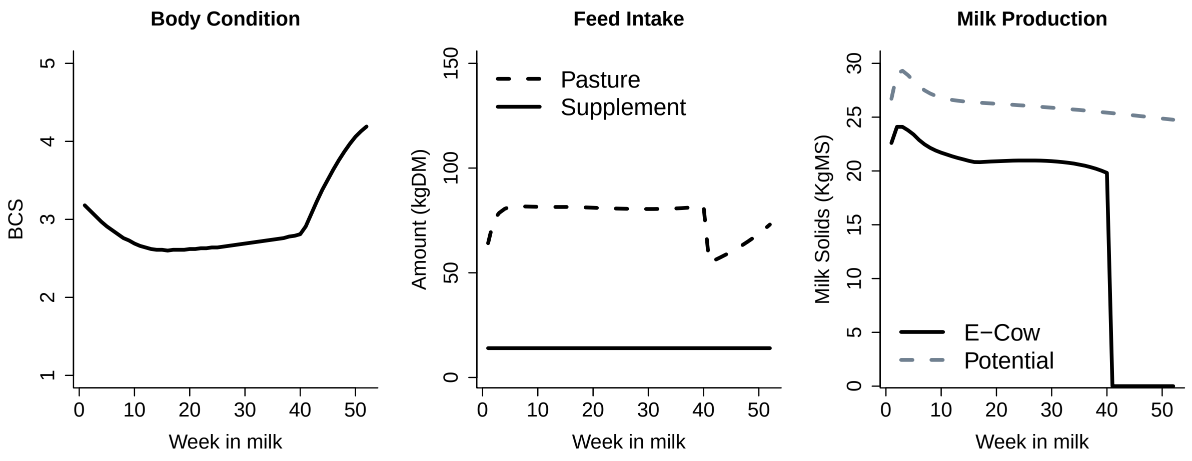


Figure A.1: Policy 2: 2kg/cow/day.

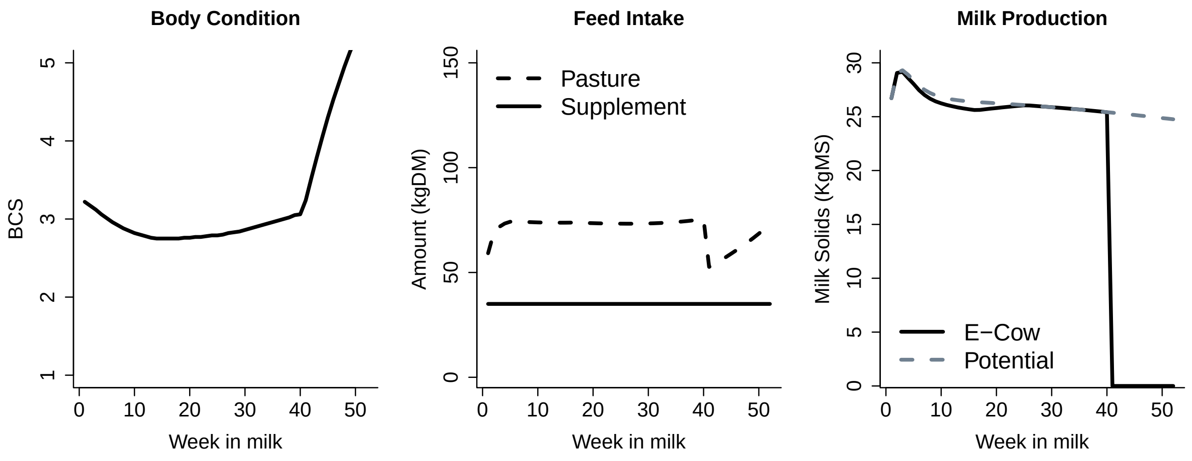


Figure A.2: Policy 3: 5kg/cow/day.

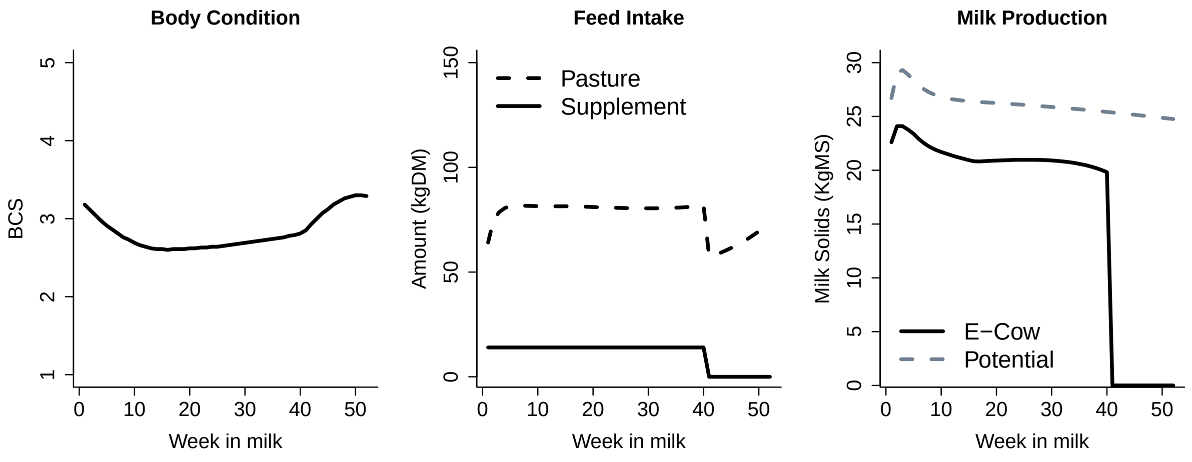


Figure A.3: Policy 4: 2kg/cow/day while milking.

B Energy required for maintenance

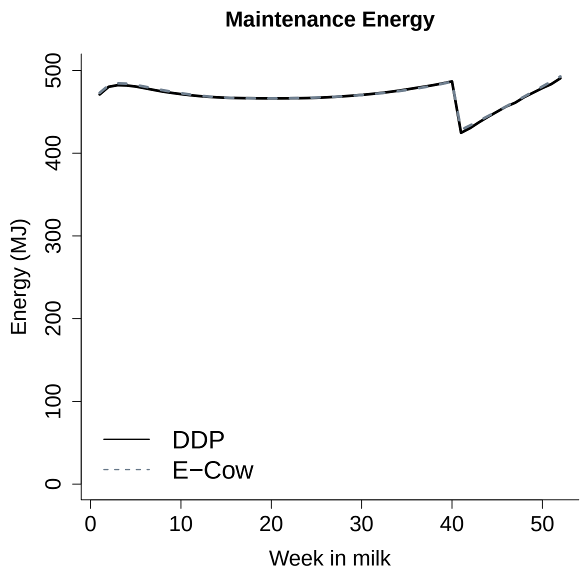


Figure B.1: Energy required for maintenance.

C DDP Convergence

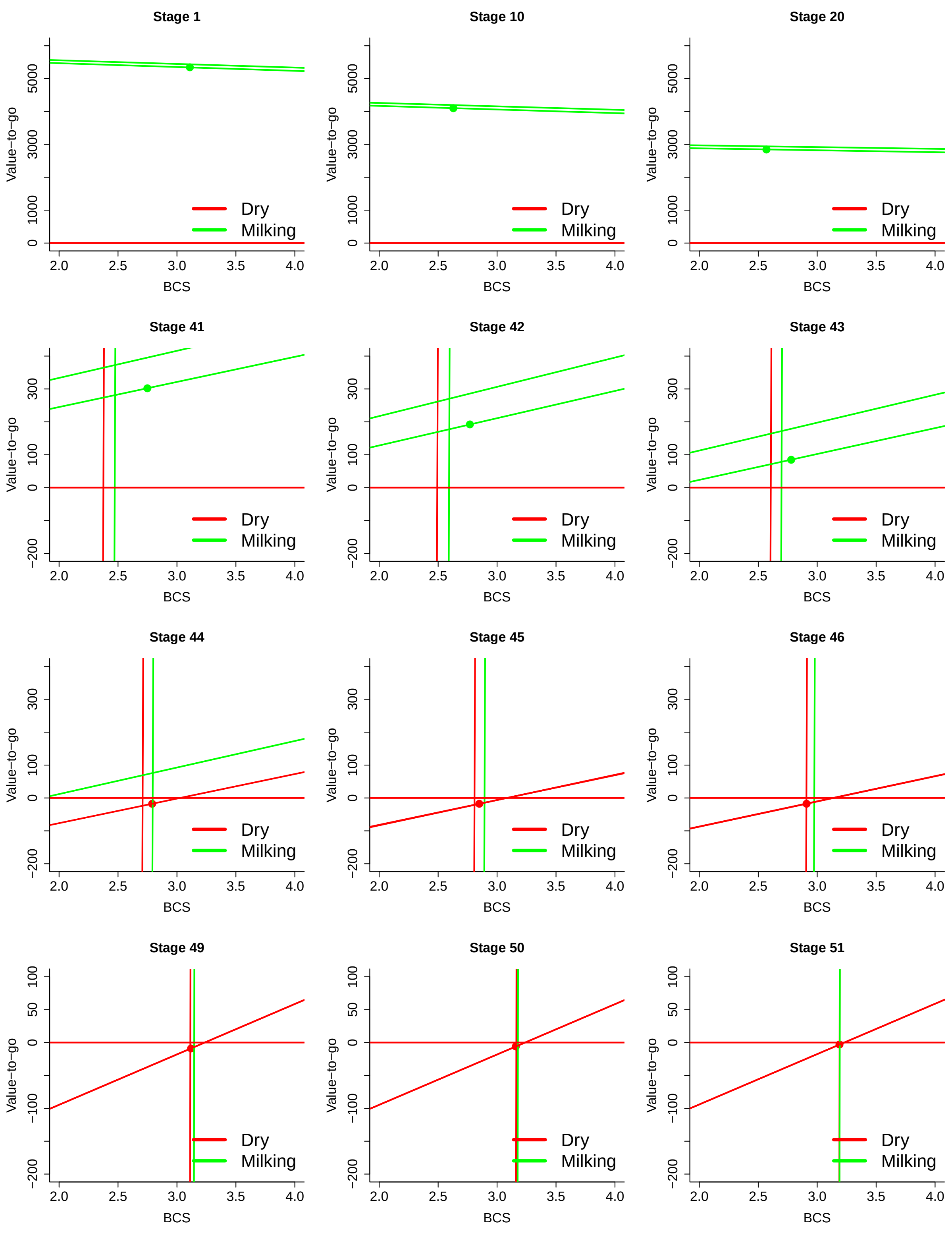


Figure C.1: Actual cuts added to one run of MOO. DDP took five passes to converge.

D Code

This project generated in excess of 3000 lines of code. For the sake of brevity, only the python code for the Dual Dynamic Program is included in this appendix.

```
from gurobipy import *
from model_inputs import *
from math import exp, log10
import numpy as np
import matplotlib.pyplot as plt
import sim_ecow


# ================================================================================
#       DDP functions
# ================================================================================
def createStage(t, x, DLDT, reg, is_mip=True, t_max=52):
    # This function creates a single stage problem given
    #   t        = week
    #   x        = dictionary of stage variables x['state']
    #   dldt     = dictionary of change in lipid (kg) for each mob
    #   reg      = dictionary of linear regression constants reg[mob][0,1]
    #   is_mip   = boolean if integer version
    #   t_max    = final week

    # Get the potential milk yield
    MEPMY = MEpmy(t)

    # Create the stage model
    m = Model("Stage_%d" % t)

    # Turn off display
    m.setParam('OutputFlag', False)

    # Initialise variables
    old_bcs, old_lw, bcs, lw, mem = {}, {}, {}, {}, {}
    lb_bcs_penalty, ub_bcs_penalty, fin_bcs_penalty = {}, {}, {}
    sup, e_milk, e_l, e_n, e_r, e_t = {}, {}, {}, {}, {}, {}
    herb_intake, pot_dmi, pot_dmi_pow = {}, {}, {}
    net_bcs, net_milk, net_milk_n, net_milk_p = {}, {}, {}, {}
    s_e_split, s_bcs_p, s_bcs_n, s_p_dmi_p = {}, {}, {}, {}
    s_p_dmi_n, s_e_m_p, s_e_m_n, sos_lw = {}, {}, {}, {}
    lac, new_lac = {}, {}
    LW_PER_BCS = {}

    # Duplicate variables for each mob
    for mob in MOBS:
        # Lactating variable
        new_lac[mob] = m.addVar(lb=0, ub=1, vtype=GRB.BINARY,
                                name='lac_%s' % mob)
```

```
lac[mob] = m.addVar(lb=0, ub=1,
                    name='currentlac_%s' % mob)
# Incoming BCS
old_bcs[mob] = m.addVar(lb=0, ub=10, name='old_bcs_%s' % mob)
# Incoming LW
old_lw[mob] = m.addVar(lb=0, ub=1e3, name='old_lw_%s' % mob)

# New bcs
bcs[mob] = m.addVar(lb=0, ub=10, name='bcs_%s' % mob)
# New lw
lw[mob] = m.addVar(lb=0, ub=1e3, name='lw_%s' % mob)
# New maintenance
mem[mob] = m.addVar(lb=-INF, ub=INF, name='mem_%s' % mob)

# Penalty variables to penalise low and herb_intakegh BCS
lb_bcs_penalty[mob] = m.addVar(lb=0, name='lb_bcs_penalty%s' % mob)
ub_bcs_penalty[mob] = m.addVar(lb=0, name='ub_bcs_penalty%s' % mob)

if t == t_max:
    # fin_bcs_penalty variable to allow
    #  final BCS constraint to be violated
    fin_bcs_penalty[mob] = m.addVar(
        lb=0, name='fin_bcs_penalty_%s' % mob)

# Amount of suppleee_nt to be fed
sup[mob] = m.addVar(lb=0, ub=MOBS[mob]['MAX_SUP'],
                    name='supplement_%s' % mob)

# Energy devoted to milk production
e_milk[mob] = m.addVar(lb=0, ub=MEPMY, name='e_milk_%s' % mob)
# Energy for Lipid change
e_l[mob] = m.addVar(lb=-INF, ub=INF, name='e_l_%s' % mob)
# Net energy balance
e_n[mob] = m.addVar(lb=-INF, ub=INF, name='e_n_%s' % mob)
# Required Energy
e_r[mob] = m.addVar(lb=-INF, ub=INF, name='e_r_%s' % mob)
# Total Energy Intake
e_t[mob] = m.addVar(lb=0, ub=INF, name='e_t_%s' % mob)

# Herbage Intake
herb_intake[mob] = m.addVar(lb=0, name='herb_intake_%s' % mob)
# Potential Dry Matter Intake
pot_dmi[mob] = m.addVar(lb=0, ub=INF, name='pot_dmi_%s' % mob)
# Potential Dry Matter Intake pow 0.5636
pot_dmi_pow[mob] = m.addVar(lb=0, name='pot_dmi_pow_%s' % mob)

# Additional energy to BCS change
net_bcs[mob] = m.addVar(lb=-INF, ub=INF, name='net_bcs_%s' % mob)
# Additional energy to milk production
net_milk[mob] = m.addVar(lb=-MEPMY, ub=0, name='net_milk_%s' % mob)
# Positive slack variable
net_milk_p[mob] = m.addVar(lb=0, ub=INF, name='net_milk_p%s' % mob)
# Negative slack variable
```

```
net_milk_n[mob] = m.addVar(lb=-INF, ub=0, name='net_milk_n_%s' % mob)

# ================================================================
# SOS2 to partition net energy
s_e_split[mob] = list()
for i in range(3):
    s_e_split[mob].append(
        m.addVar(lb=0, ub=1, name='vs_e_split%d_%s' % (i, mob)))

# ================================================================
# SOS2 to deal with bilinear BCS*s_e_split
# Positive quadratic
s_bcs_p[mob] = {'var': m.addVar(lb=0, name='vs_bcs_p_%s' % mob),
                'sos': list(),
                'range': linspace(0, 10, N_SOS_MILK_p)}
for i in range(N_SOS_MILK_p):
    s_bcs_p[mob]['sos'].append(
        m.addVar(lb=0, ub=1, name='ls_bcs_p_%d_%s' % (i, mob)))

# Negative quadratic
s_bcs_n[mob] = {'var': m.addVar(lb=0, name='vs_bcs_n_%s' % mob),
                'sos': list(),
                'range': linspace(0, 10, N_SOS_MILK_n)}
for i in range(N_SOS_MILK_n):
    s_bcs_n[mob]['sos'].append(
        m.addVar(lb=0, ub=1, name='ls_bcs_n_%d_%s' % (i, mob)))

# ================================================================
# SOS2 to deal with bilinear p_dmi^0.75 * sup
# Positive quadratic
s_p_dmi_p[mob] = {'var': m.addVar(lb=0, name='vs_p_dmi_p_%s' % mob),
                  'sos': list(),
                  'range': linspace(0, 30, N_SOS_PDMI_p)}
for i in range(N_SOS_PDMI_p):
    s_p_dmi_p[mob]['sos'].append(
        m.addVar(lb=0, ub=1, name='ls_p_dmi_p_%d_%s' % (i, mob)))

# Negative quadratic
s_p_dmi_n[mob] = {'var': m.addVar(lb=0, name='vs_p_dmi_n_%s' % mob),
                  'sos': list(),
                  'range': linspace(-5, 30, N_SOS_PDMI_n)}
for i in range(N_SOS_PDMI_n):
    s_p_dmi_n[mob]['sos'].append(
        m.addVar(lb=0, ub=1, name='ls_p_dmi_n_%d_%s' % (i, mob)))

# ================================================================
# SOS2 to deal with bilinear HI*LW in the maintenance eqn.
# Positive quadratic
s_e_m_p[mob] = {'var': m.addVar(lb=0, name='vs_e_m_p_%s' % mob),
                'sos': list(),
                'range': linspace(SOS_MEP_MIN,
                                  SOS_MEP_MAX, N_SOS_MEm_p)}
for i in range(N_SOS_MEm_p):
```

```
          s_e_m_p[mob]['sos'].append(
              m.addVar(lb=0, ub=1, name='ls_e_m_p_%d_%s' % (i, mob)))

      # Negative quadratic
      s_e_m_n[mob] = {'var': m.addVar(lb=0, name='vs_e_m_n_%s' % mob),
                      'sos': list(),
                      'range': linspace(SOS_MEN_MIN, SOS_MEN_MAX,
                                        N_SOS_MEm_n)}
      for i in range(N_SOS_MEm_n):
          s_e_m_n[mob]['sos'].append(
              m.addVar(lb=0, ub=1, name='ls_e_m_p_%d_%s' % (i, mob)))

      # ======================================================================
      # SOS2 to deal with lw^0.75
      sos_lw[mob] = {'var': m.addVar(lb=0, ub=700, name='sos_lw_%s' % mob),
                     'sos': list(),
                     'range': linspace(SOS_MEA_MIN, SOS_MEA_MAX, N_SOS_LW)}
      for i in range(N_SOS_LW):
          sos_lw[mob]['sos'].append(m.addVar(lb=0, ub=1,
                                    name='soslw%d_%s' % (i, mob)))

  # ========================================================================
  #    Value to go
  if t < t_max:
      theta = m.addVar(lb=-1e10, ub=THETA_MAX, name='theta')

  # ========================================================================
  m.modelSense = GRB.MAXIMIZE
  m.update()

  # ========================================================================
  #    Objective
  if t == t_max:
      # No value to go and final penalty
      m.setObjective(quicksum(MOBS[mob]['PROPORTION'] * (
          MILK_PRICE * e_milk[mob] / wMilkMEperKG(t)
          - SUPCOST[t] / SUPP_DM_CONTENT * sup[mob]
          - 1e5 * fin_bcs_penalty[mob])
          for mob in MOBS))
  else:
      # Penalties for being above or below bcs bounds
      m.setObjective(quicksum(MOBS[mob]['PROPORTION'] * (
          MILK_PRICE * e_milk[mob] / wMilkMEperKG(t)
          - SUPCOST[t] / SUPP_DM_CONTENT * sup[mob]
          - 0 * lb_bcs_penalty[mob]
          - 0 * ub_bcs_penalty[mob])
          for mob in MOBS) + theta)

  # ========================================================================
  #    Constraints
  for mob in MOBS:
      LW_PER_BCS[mob] = MOBS[mob]['LW_CALVING'] \
          / (5 + MOBS[mob]['BCS_CALVING'])
```

```
# Dummy constraint to get dual variable
m.addConstr(old_bcs[mob] == x['bcs_%s' % mob], 'pi_bcs_%s' % mob)

# Convert old_bcs into old_lw
m.addConstr(old_lw[mob] == LW_PER_BCS[mob] * (old_bcs[mob] + 5),
            'cold_lw_%s' % mob)

# Energy for milk
m.addConstr(e_milk[mob] <= MEPMY * lac[mob] + net_milk[mob],
            'maxmilk_%s' % mob)

# Min bcs penalty
m.addConstr(bcs[mob] + lb_bcs_penalty[mob] >= MIN_LACTATING_BCS,
            'minbcs_%s' % mob)

# Max bcs penalty
m.addConstr(bcs[mob] - ub_bcs_penalty[mob] <= 4, 'maxbcs_%s' % mob)

# Required energy
m.addConstr(e_r[mob] == mem[mob] + e_l[mob] +
           MEp(t) + MEPMY * lac[mob], 'cEnergyReq_%s' % mob)

# Lipid change energy
m.addConstr(e_l[mob] == DLDT[mob] *
            (10.1 + 1.976 * old_bcs[mob]) / KM, 'cLipid_%s' % mob)

# Maintenance energy
m.addConstr(mem[mob] == MEm(sos_lw[mob]['var'], s_e_m_p[mob]['var'],
           s_e_m_n[mob]['var'], lw[mob], e_milk[mob]),
            'cMEm_%s' % mob)

# Potential dry matter intake
m.addConstr(pot_dmi[mob] <= DMIr(t, old_lw[mob]), 'cDMIr_%s' % mob)
m.addConstr(pot_dmi[mob] <= DMIe(e_r[mob]), 'cDMIe_%s' % mob)
m.addConstr(pot_dmi[mob] <= DMIg(t, old_lw[mob]), 'cDMIg_%s' % mob)

# Total energy intake
m.addConstr(e_t[mob] == herb_intake[mob] * PAST_ME +
           sup[mob] * SUPP_ME, 'ce_t_%s' % mob)

# Net energy
m.addConstr(e_n[mob] == e_t[mob] - e_r[mob], 'cNet_%s' % mob)
m.addConstr(net_bcs[mob] + net_milk[mob] == e_n[mob], 'netEq_%s' % mob)

# net milk constraints
m.addConstr(net_milk[mob] >= -MEPMY * lac[mob], 'cnet_milk_%s' % mob)
m.addConstr(net_milk[mob] == net_milk_p[mob] +
            net_milk_n[mob], 'cnet_milkSum_%s' % mob)
m.addConstr(net_milk_p[mob] <= 2 * MEPMY * (1 - lac[mob]),
            'cnet_milk_pUB_%s' % mob)

# new bcs
```

```
m.addConstr(bcs[mob] == reg[mob][0] * old_bcs[mob] +
            reg[mob][1] * (e_l[mob] + net_bcs[mob]), 'cBCS_%s' % mob)

# new lw
m.addConstr(lw[mob] == LW_PER_BCS[mob] * (bcs[mob] + 5) +
            sum([LWp(i) for i in range(1, t + 1)]), 'cLW_%s' % mob)

if t == t_max:
    # Target bcs
    m.addConstr(bcs[mob] >= TARGET_BCS - fin_bcs_penalty[mob],
                'cFinalBCS_%s' % mob)

#  SOS Type II constraints
if is_mip:
    m.addSOS(GRB.SOS_TYPE2, s_e_split[mob])
    m.addSOS(GRB.SOS_TYPE2, s_e_m_p[mob]['sos'])
    m.addSOS(GRB.SOS_TYPE2, s_e_m_n[mob]['sos'])
    m.addSOS(GRB.SOS_TYPE2, s_bcs_p[mob]['sos'])
    m.addSOS(GRB.SOS_TYPE2, s_bcs_n[mob]['sos'])
    m.addSOS(GRB.SOS_TYPE2, s_p_dmi_p[mob]['sos'])
    m.addSOS(GRB.SOS_TYPE2, s_p_dmi_n[mob]['sos'])
    m.addSOS(GRB.SOS_TYPE2, sos_lw[mob]['sos'])

# Convexity constraints
m.addConstr(quicksum(i for i in s_e_split[mob]) == 1,
            'convexity_s_e_split_%s' % mob)
m.addConstr(quicksum(i for i in s_e_m_p[mob]['sos']) == 1,
            'convexity_s_e_m_p_%s' % mob)
m.addConstr(quicksum(i for i in s_e_m_n[mob]['sos']) == 1,
            'convexity_s_e_m_n_%s' % mob)
m.addConstr(quicksum(i for i in sos_lw[mob]['sos']) == 1,
            'convexity_sos_lw_%s' % mob)
m.addConstr(quicksum(i for i in s_bcs_p[mob]['sos']) == 1,
            'convexity_s_bcs_p_%s' % mob)
m.addConstr(quicksum(i for i in s_bcs_n[mob]['sos']) == 1,
            'convexity_s_bcs_n_%s' % mob)
m.addConstr(quicksum(i for i in s_p_dmi_p[mob]['sos']) == 1,
            'convexity_s_p_dmi_p_%s' % mob)
m.addConstr(quicksum(i for i in s_p_dmi_n[mob]['sos']) == 1,
            'convexity_s_p_dmi_n_%s' % mob)

# ==========================================================================
# SOS2 to deal with net energy split
m.addConstr(e_n[mob] == - s_e_split[mob][0] * 2 *
            (MEPMY + abs(DLDT[mob]) * 10.1 / KM)
            - 2 * abs(DLDT[mob]) * 1.976 / KM * 0.25 *
            (s_bcs_p[mob]['var'] - s_bcs_n[mob]['var'])
            + s_e_split[mob][2] * (500), 'xs_e_split_%s' % mob)
m.addConstr(net_milk_n[mob] ==
            s_e_split[mob][0] * (-2 * MEPMY), 'ys_e_split_%s' % mob)

# ==========================================================================
# SOS2 to deal with bilinear lw*hi
# Positive quadratic
m.addConstr(lw[mob] + herb_intake[mob] / 7.0 == quicksum(
            [s_e_m_p[mob]['sos'][i] * s_e_m_p[mob]['range'][i]
             for i in range(N_SOS_MEm_p)]), 'xs_e_m_p_%s' % mob)
m.addConstr(s_e_m_p[mob]['var'] == quicksum(
            [s_e_m_p[mob]['sos'][i] *
             pow(s_e_m_p[mob]['range'][i], 2)
             for i in range(N_SOS_MEm_p)]), 'ys_e_m_p_%s' % mob)

# Negative quadratic
m.addConstr(lw[mob] - herb_intake[mob] / 7.0 == quicksum(
            s_e_m_n[mob]['sos'][i] * s_e_m_n[mob]['range'][i]
            for i in range(N_SOS_MEm_n)), 'xs_e_m_n_%s' % mob)
m.addConstr(s_e_m_n[mob]['var'] == quicksum(
            [s_e_m_n[mob]['sos'][i] *
             pow(s_e_m_n[mob]['range'][i], 2)
             for i in range(N_SOS_MEm_n)]), 'ys_e_m_n_%s' % mob)

# ===========================================================================
# SOS2 to deal with lw^0.75
# Positive quadratic
m.addConstr(lw[mob] == quicksum([sos_lw[mob]['sos'][i] *
            sos_lw[mob]['range'][i]
            for i in range(N_SOS_LW)]), 'xs_lw_%s' % mob)
m.addConstr(sos_lw[mob]['var'] == quicksum(
            [sos_lw[mob]['sos'][i] * pow(sos_lw[mob]['range'][i], 0.75)
             for i in range(N_SOS_LW)]), 'ys_lw_%s' % mob)

# ===========================================================================
# SOS2 to deal with bilinear BCS*s_e_split
# Positive quadratic
m.addConstr(bcs[mob] + s_e_split[mob][0] == quicksum(
            s_bcs_p[mob]['sos'][i] * s_bcs_p[mob]['range'][i]
            for i in range(N_SOS_MILK_p)), 'xs_bcs_p_%s' % mob)
m.addConstr(s_bcs_p[mob]['var'] == quicksum(
            s_bcs_p[mob]['sos'][i] * pow(s_bcs_p[mob]['range'][i], 2)
            for i in range(N_SOS_MILK_p)),
            'ys_bcs_p_%s' % mob)

# Negative quadratic
m.addConstr(bcs[mob] - s_e_split[mob][0] == quicksum(
            s_bcs_n[mob]['sos'][i] * s_bcs_n[mob]['range'][i]
            for i in range(N_SOS_MILK_n)), 'xs_bcs_n_%s' % mob)
m.addConstr(s_bcs_n[mob]['var'] == quicksum(
            s_bcs_n[mob]['sos'][i] * pow(s_bcs_n[mob]['range'][i], 2)
            for i in range(N_SOS_MILK_n)), 'ys_bcs_n_%s' % mob)

# ===========================================================================
# SOS2 to deal with bilinear sup*pdmi^0.75
# Positive quadratic
m.addConstr(pot_dmi_pow[mob] + sup[mob] / 7.0 == quicksum(
            s_p_dmi_p[mob]['sos'][i] * s_p_dmi_p[mob]['range'][i]
            for i in range(N_SOS_PDMI_p)), 'xs_p_dmi_p_%s' % mob)
```

```
        m.addConstr(s_p_dmi_p[mob]['var'] == quicksum(
                    s_p_dmi_p[mob]['sos'][i] *
                    pow(s_p_dmi_p[mob]['range'][i], 2)
                    for i in range(N_SOS_PDMI_p)),
                    'ys_p_dmi_p_%s' % mob)

        # Negative quadratic
        m.addConstr(pot_dmi_pow[mob] - sup[mob] / 7.0 == quicksum(
                    s_p_dmi_n[mob]['sos'][i] * s_p_dmi_n[mob]['range'][i]
                    for i in range(N_SOS_PDMI_n)), 'xs_p_dmi_n_%s' % mob)
        m.addConstr(s_p_dmi_n[mob]['var'] == quicksum(
                    s_p_dmi_n[mob]['sos'][i] *
                    pow(s_p_dmi_n[mob]['range'][i], 2)
                    for i in range(N_SOS_PDMI_n)), 'ys_p_dmi_n_%s' % mob)

        # ============================================================
        # Herbage intake
        m.addConstr(herb_intake[mob] ==
                    7 * (0.57676 * pow(PASTURE[t] / 7.0, 0.464) *
                        pot_dmi_pow[mob]
                        - 0.57676 * pow(PASTURE[t] / 7.0, 0.464) * 21
                        / MOBS[mob]['LW_CALVING'] * 0.25
                        * (s_p_dmi_p[mob]['var'] -
                        s_p_dmi_n[mob]['var'])
                        )
                    + 0.18 * sup[mob], 'cherb_intake_%s' % mob)

        pot_dmi_range = linspace(TAN_PDMI_MIN, TAN_PDMI_MAX, N_TAN_PDMI)
        for p in pot_dmi_range:
            # Add cuts
            m.addConstr(pot_dmi_pow[mob] <= pow(p, 0.536) +
                        0.536 * pow(p, -0.464) * (pot_dmi[mob] / 7.0 - p),
                        'cpot_dmi_pow_cut_p%d_%s' % (p, mob))

    for mob in MOBS:
        m.addConstr(lac[mob] == x['lac_%s' % mob], 'pi_lac_%s' % mob)
        m.addConstr(new_lac[mob] <= lac[mob], 'cCurrent_lac_%s' % mob)
        if t >= MOBS[mob]['DRY_OFF_WEEK']:
            # Max lactation length
            m.addConstr(new_lac[mob] <= 0.0, 'dried_off_%s' % mob)
    ordered_groups = MOBS.keys()
    ordered_groups.sort()
    for i, mob in enumerate(ordered_groups[1:]):
        m.addConstr(new_lac[mob] <= new_lac[ordered_groups[i]],
                    'ordered_groups')

    # ======================
    #   Linking constraint
    m.addConstr(
        quicksum([MOBS[mob]['PROPORTION'] * sup[mob] for mob in MOBS])
        <= MAX_SUP, name='max_supplement')

    # ======================
    #   Ensure everything is up-to-date
    m.update()
    return(m)


def get_max_pi(s, thetaBar, xBar, pi):
    m = 0
    for k in xBar:
        if k == s:
            continue
        if 'lac' in k:
            m_temp = THETA_MAX - thetaBar
        else:
            if pi[k] > 0:
                m_temp = THETA_MAX + xBar[k] * pi[k]
            else:
                m_temp = THETA_MAX + (xBar[k] - 10) * pi[k]
        if m_temp > m:
            m = m_temp
    return m + THETA_MAX


def addCut(m, thetaBar, xBar, pi):
    # This function adds a cut to the model m, given
    #   thetaBar    = Observed value-to-go in next stage
    #   xBar        = Observed states in current stage
    #   pi          = dual variables of states
    try:
        # Get value to go
        theta = m.getVarByName('theta')

        x = {}
        for s in pi:
            # get all the dual variables
            if 'lac' in s:
                # If BCS is large, then it has a dual of
                # zero. But if also not milking, then the
                # cut chops off the objective.
                #
                # Doesn't seem to realise that if the BCS
                # dropped it could milk and make more money
                #
                # To fix (hack) make the duals much steeper
                # pi[s] *= 1e5
                pi[s] = get_max_pi(s, thetaBar, xBar, pi)
            x[s] = m.getVarByName(s)

        m.addConstr(theta <= thetaBar + quicksum([pi[s] * (x[s] - xBar[s])
                    for s in pi if 'lac' not in s])
                    + quicksum([pi[s]
                             * (x[s] - 2 * x[s] * xBar[s] + xBar[s])
                             for s in pi if 'lac' in s]),
                    'cut%s%d' % (m.ModelName, m.getAttr('NumConstrs')))
        # with open('cuts.csv', 'a') as f:
        #     f.write('%s, %.2f, %.2f, %.2f, %.2f, %.2f\n' % (m.ModelName, thetaBar, pi

            # f.write('%s\nTheta:\t%s\nPi:\t%s\nX:\t%s\n\n' %
            #         (m.ModelName, str(thetaBar), str(pi), str(xBar)))
            # f.write(line + '\n')

        # update model
        m.update()
    except:
        # write the cut to file
        with open('Output\\failedCut%s.txt' % m.ModelName, 'w') as f:
            f.write('Theta:\t%s\nPi:\t%s\nX:\t%s' %
                    (str(thetaBar), str(pi), str(xBar)))
        raise Exception(
            'Unable to add cut to %s\nSee \'Output\\failedCut%s.txt\'' %
            (m.ModelName, m.ModelName))
    return(m)


def updateStartX(m, xb):
    # This function updates the values of the state variables xb in model m
    try:
        for key in xb:
            c = m.getConstrByName('pi_%s' % (key))
            c.setAttr('rhs', xb[key])
        m.update()
    except:
        raise Exception('Unable to update starting values of stage %d' % t)
    return(m)


def solveStage(m):
    # This function solves the model m

    # Solve stage
    m.optimize()

    # Not solved to optimality
    if m.status != GRB.status.OPTIMAL:
        # Irreducible inconsistent subsystem
        m.computeIIS()
        # Write out for inspection
        m.write('.\\Output\\%s.ilp' % m.ModelName)
        # Halt because SDDP requires feasibility
        raise Exception('%s not solved to optimality' % m.ModelName)

    # Fix integers to find duals
    f = m.fixed()

    # Solve fixed problem
    f.optimize()
    x, pi, out = {}, {}, {}
    for mob in MOBS:
        vName = 'bcs_%s' % mob
        lName = 'lac_%s' % mob

        # Find the variable values of original prob
        x[vName] = m.getVarByName(vName).x
        x[lName] = m.getVarByName(lName).x

        # Get dual variables
        pi[vName] = f.getConstrByName('pi_%s' % vName).pi
        pi[lName] = f.getConstrByName('pi_%s' % lName).pi

        out[mob] = {'herb_intake': m.getVarByName('herb_intake_%s' % mob).x,
                    'mem': m.getVarByName('mem_%s' % mob).x,
                    'sup': m.getVarByName('supplement_%s' % mob).x,
                    'e_milk': m.getVarByName('e_milk_%s' % mob).x,
                    'lac': x[lName]
                    }

    try:
        theta = m.getVarByName('theta').x
    except:
        theta = 0
    # Struct. to return
    d = {'m': m, 'obj': m.objVal, 'theta': theta, 'x': x, 'pi': pi, 'out': out}
    return(d)


def createStages(stages, start_bcs=None):
    x0 = {}
    DLDT = {}
    for mob in MOBS:
        if start_bcs is None:
            bcs0 = MOBS[mob]['BCS_CALVING']
        else:
            bcs0 = start_bcs
        DLDT[mob] = dLdT(mob)
        x0['bcs_%s' % mob] = bcs0
        x0['lac_%s' % mob] = 1
    reg = fitBCS()

    revStages = list(stages)
    revStages.reverse()

    model_list = {'maxErr': None}
    model_list[stages[0] - 1] = {'x': x0}
    for t in stages:
        # try:
        dl, lac = {}, {}
        for mob in MOBS:
            dl[mob] = DLDT[mob][t]
        m = createStage(t, model_list[t - 1]['x'], dl, reg)
    # except:
        #     raise Exception('Failed to create stage %d' % t)
        model_list[t] = solveStage(m)
    return(model_list)


def backwardPass(stages, model_list):
    # Backwards pass of SDDP

    revStages = list(stages)
    revStages.reverse()

    # Don't add cut to last stage
    for t in revStages[1:]:
        # Get model for therb_intakes stage
        m = model_list[t]['m']

        # Observed states
        xBar = model_list[t]['x']

        # Ovserved value to go
        thetaBar = model_list[t + 1]['obj']

        # Dual variables on stages in t+1
        pi = model_list[t + 1]['pi']

        # Add cut
        model_list[t]['m'] = addCut(m, thetaBar, xBar, pi)

        # Resolve with new cut
        model_list[t] = solveStage(model_list[t]['m'])

    return(model_list)


def forwardPass(stages, model_list, changeTol=1e-3):
    # Forward pass of the SDDP

    # Maximum change in theta between iterations
    bigDelta = None

    for t in stages:
        # Remember old objective
        oldObj = model_list[t]['obj']

        # Solve the stage
        model_list[t] = solveStage(model_list[t]['m'])

        # If stage to pass forward to
        if t < max(stages):
            # Pass forward states
            model_list[t + 1]['m'] = updateStartX(
                model_list[t + 1]['m'], model_list[t]['x'])

        # Change in objective
        delta = abs(model_list[t]['obj'] - oldObj)

        # Bigger change?
        if delta > bigDelta:
            bigDelta = delta

    # Store maximum change. SDDP stops when therb_intakes converges
    model_list['maxErr'] = bigDelta
    return(model_list, bigDelta > changeTol)


# ==================================================
#       Equations
# ==================================================
KM = 0.02 * MEdiet + 0.5


# Wilmink fat function
def fat(t):
    f = 0
    for i in range(7):
        f += FA + FB * exp(-0.05 * (7 * t - i)) + FC * (7 * t - i)
    return(f / 700.0)


# Wilmink protein function
def prot(t):
    f = 0
    for i in range(7):
        f += PA + PB * exp(-0.05 * (7 * t - i)) + PC * (7 * t - i)
    return(f / 700.0)


# Milk potential
def getMEpmy():
    out = [0] * 53
    for t in range(1, 53):
        out[t] = MEpmy(t)
    return(out)


def MEpmy(week):
    mepmy = 0
    for tbar in range(7 * (week - 1) + 1, 7 * week + 1):
        for i in range(0, 3):
            mepmy += MB[i] * exp(MC[i] * tbar)
    mepmy *= MA
    return(mepmy)
```

```
def wMilkMEperKG(t):
    return((0.0376 * fat(t) * 100 + 0.0209 * prot(t) * 100 + 0.948)
           / (0.02 * MEdiet + 0.4) / (fat(t) + prot(t)))


def dLdT(mob):
    emptyweight = MOBS[mob]['LW_CALVING'] * \
        (1 - 0.1113 / (1 - 0.129 * MOBS[mob]['BCS_CALVING']))
    lcalv = 0.12 * (MOBS[mob]['BCS_CALVING'] - 0.36) * emptyweight
    lprime = 0.26 * emptyweight
    lnext = 0.12 * (3.2 - 0.36) * emptyweight
    dlt = 2 * (112 - DAYS_TILL_CONCEPTION) * (lnext - lprime) / pow(284, 2)
    ltarget = lprime + \
        (lnext - lprime) * pow(112 - DAYS_TILL_CONCEPTION, 2) / pow(284, 2)
    dLcalv = 2 * (ltarget - lcalv) / 112 - dlt
    dldt = [0] * 366

    for t in range(1, 366):
        if t < 112:
            dldt[t] = dLcalv - t / 112.0 * (dLcalv - dlt)
        else:
            dldt[t] = 2 * (lnext - lprime) * \
                (t - DAYS_TILL_CONCEPTION) / pow(284, 2)

    DLDT = [0] * 53
    for week in range(1, 53):
        DLDT[week] = sum(dldt[7 * (week - 1) + 1: 7 * (week) + 1])
    return(DLDT)


def MEp(t):
    # Energy required for pregnancy in week t
    dp = DP(t)
    me = 0
    if dp > 0:
        for t in range(max(1, int(dp - 6)), int(dp + 1)):
            me += mep(dp)
    return(me)


def mep(dp):
    # Energy required for pregnancy on day of pregnancy dp
    term1 = 0.025 * pow(CBW, 2)
    term2 = pow(10, 151.665 - 151.64 * exp(-5.76e-5 * dp))
    term3 = 0.0201 * exp(-5.76e-5 * dp)
    mep = term1 * term2 * term3 / (0.13 * 40)
    return(mep)


def DP(t):
    # Number of Days pregnant in week t
    if (7 * t < DAYS_TILL_CONCEPTION):
        return(0)
    return(7 * t - DAYS_TILL_CONCEPTION)


def MEm(term1, mem_plus, mem_neg, LW, MEmilk):
    # Energy required for Maintenance
    b0 = 1.4 * 0.28 * exp(-0.03 * AGE) / KM
    b1 = 0.0025 * (0.9 - PAST_D) / KM
    b2 = 0.05 * TER / ((3 + GF) * KM)
    return(7 * (b0 * term1 + b1 * 0.25 * (mem_plus - mem_neg) + b2 * LW)
           + 0.1 * MEmilk)


def SOL(week):
    # Stage of Lactation
    s = 0.0
    for t in range(7 * (week - 1) + 1, 7 * week + 1):
        s += 0.67 + 0.0972 * \
            (4.0401 * log10(t / 7.0) - 0.095 * (t / 7.0) + 0.095)
    return(s)


def DMIg(week, LW):
    # Maximum kgDM of pasture cow can eat due to grazing
    return(0.0375 * LW * SOL(week))


def DMIe(MEreq):
    # Maximum kgDM of pasture cow can eat due to energy
    return(MEreq / PAST_ME)


def DMIr(week, LW):
    # Maximum kgDM of pasture cow can eat due to rumen size
    return(0.0165 * LW * SOL(week) / PAST_NDF)


def LWp(t):
    # Liveweight change due to pregnancy in week t
    dp = DP(t)
    lw = 0
    if dp > 0:
        for t in range(max(1, int(dp - 6)), int(dp + 1)):
            lw += lwp(dp)
    return(lw)


def lwp(dp):
    # liveweeight change of uterus on day pregnant dp
    return(CBW * (18.28 * 0.02 - 0.0000286 * dp)
           * exp(0.02 * dp - 0.0000143 * pow(dp, 2)) / 1000)


def linspace(a, b, n):
    # Implementation of the linspace function
    if n == 1:
        return b
    h = (b - a) / (n - 1.0)
    out = list()
    for i in range(n):
        out.append(a + h * i)
    return out


# ==========================================================================
#        Regression
# ==========================================================================
def MEperMElip(BCS):
    return((10.1 + 1.976 * BCS) / KM)


def newBCS(ME, BCS, mob):
    BCSperMElip = (5 + MOBS[mob]['BCS_CALVING']) / MOBS[mob]['LW_CALVING']
    return(BCS + ME * BCSperMElip / MEperMElip(BCS))


def fitBCS():
    bcs_range = linspace(2, 4, 10)
    melip_range = range(-200, 200, 5)
    n_rows = len(bcs_range) * len(melip_range)
    reg = {}
    for mob in MOBS:
        A = np.zeros([n_rows, 2])
        b = np.zeros([n_rows])
        r = 0
        for bcs in bcs_range:
            for me in melip_range:
                A[r][0] = bcs
                A[r][1] = me
                b[r] = newBCS(me, bcs, mob)
                r += 1
        lr = np.linalg.lstsq(A, b)[0]
        reg[mob] = list(lr)
    return(reg)


def ExtractValues(stages, model_list):
    # This function extracts the data from the solved models
    # It returns a dictionary values[mob][variable]

    stages.sort()
    values = {}
    for mob in MOBS:
        values[mob] = {}
        for k in model_list[stages[0]]['out'][mob]:
            values[mob][k] = [model_list[t]['out'][mob][k]
                              for t in stages]
        values[mob]['bcs'] = [model_list[t]['x']['bcs_%s' % mob]
                              for t in stages]
    values['obj'] = model_list[stages[0]]['obj']
    return(values)


def CalculateLowerBound(model_list, stages):
    lb = 0
    for t in stages:
        lb += model_list[t]['obj'] - model_list[t]['theta']
    return(lb)


# ==========================================================================
#       Plotting
# ==========================================================================
def plotFeed(stages, hi, sup, ecowHI=None):
    gray = 3 * [0.3]
    light_gray = 3 * [0.5]
    pastPlot, = plt.plot(stages, hi, color=gray, linestyle='--', linewidth='2')
    plt.hold(True)
    supPlot, = plt.plot(stages, sup, color=light_gray, linewidth='2')
    if ecowHI is not None:
        ecowPlot, = plt.plot(
            stages, ecowHI, color=gray, linestyle='-.', linewidth='2')
    plt.xlim((0, 52))
    plt.xlabel('Weeks_in_Milk')
    plt.ylim((0, 120))
    plt.ylabel('Feed_Intake_(KgDM/Week)')
    plt.title('Feed_Intake')
    if ecowHI is None:
        plt.legend([pastPlot, supPlot], ['Pasture', 'Supplement'], loc=1)
    else:
        plt.legend([pastPlot, ecowPlot, supPlot], [
                   'DDP_Pasture', 'E-Cow_Pasture', 'Supplement'], loc=1)


def plotMilk(stages, values, ecow=None):
    gray = 3 * [0.3]
    light_gray = 3 * [0.5]
    amilk, = plt.plot(stages, [v / wMilkMEperKG(t + 1)
                      for t, v in enumerate(values)],
                      color=gray, linewidth='2')
    plt.hold(True)
    pmilk, = plt.plot(stages, [MEpmy(t) / wMilkMEperKG(t)
                      for t in stages],
                      color=light_gray, linestyle='--', linewidth='2')
    if ecow is not None:
        ecowPlot, = plt.plot(stages, [v / wMilkMEperKG(t + 1)
                             for t, v in enumerate(ecow)],
                             color=gray, linestyle='-.', linewidth='2')
    plt.xlabel('Weeks_in_Milk')
    plt.ylim((0, 30))
    plt.xlim((0, 52))
    plt.ylabel('KgMS')
    plt.title('Milk Solids Produced')
    if ecow is None:
        plt.legend([amilk, pmilk], ['Actual', 'Potential'], loc=3)
    else:
        plt.legend([amilk, ecowPlot, pmilk], [
                   'DDP', 'E-Cow', 'Potential'], loc=3)

    return


def plotBCS(stages, values, ecow=None):
    gray = 3 * [0.3]
    plot, = plt.plot(stages, values, color=gray, linewidth='2')
    if ecow is not None:
        plt.hold(True)
        ecowPlot, = plt.plot(
            stages, ecow, color=gray, linestyle='-.', linewidth='2')
        plt.legend([plot, ecowPlot], ['DDP', 'E-Cow'], loc=3)
    plt.xlabel('Weeks in Milk')
    plt.ylim((0, 5))
    plt.xlim((0, 52))
    plt.ylabel('BCS')
    plt.title('Body Condition Score')
    return(plot)


def plotMEM(stages, values, ecow=None):
    gray = 3 * [0.3]
    plot, = plt.plot(stages, values, color=gray, linewidth='2')
    if ecow is not None:
        plt.hold(True)
        ecowPlot, = plt.plot(
            stages, ecow, color=gray, linestyle='-.', linewidth='2')
        plt.legend([plot, ecowPlot], ['DDP', 'E-Cow'], loc=3)
    plt.xlabel('Weeks in Milk')
    plt.ylim((0, max(values) + 25))
    plt.xlim((0, 52))
    plt.ylabel('Energy (MJ)')
    plt.title('Energy Maintenance')
    return(plot)


def PlotStageProblem(mob_values, stages, PlotECow=False, i=''):
    # This function plots the solutions
    for mob in MOBS:
        # Get the values relating to mob
        values = mob_values[mob]

        # New figure
        fig1 = plt.figure(1, figsize=(12, 11))
        if PlotECow:
            # Run E-Cow simulation using computed supplementation policy
            ecow_output = sim_ecow.runECow(values['sup'],
                                           MOBS[mob]['DRY_OFF_WEEK'])

            # Plot herbage intake
            plt.subplot(2, 2, 1)
            plotFeed(stages, values['herb_intake'],
                     values['sup'], ecow_output['HI'][1:])

            # Plot milk production
            plt.subplot(2, 2, 2)
            plotMilk(stages, values['e_milk'], ecow_output['MEmilk'][1:])

            # Plot body condition
            plt.subplot(2, 2, 3)
            plotBCS(stages, values['bcs'], ecow_output['BCS'][1:])

            # Plot maintenance
            plt.subplot(2, 2, 4)
            plotMEM(stages, values['mem'], ecow_output['MEM'][1:])
        else:
            # Plot herbage intake
            plt.subplot(2, 2, 1)
            plotFeed(stages, values['herb_intake'], values['sup'])

            # Plot milk production
            plt.subplot(2, 2, 2)
            plotMilk(stages, values['e_milk'])

            # Plot bcs
            plt.subplot(2, 2, 3)
            plotBCS(stages, values['bcs'])

            # Plot maintenance
            plt.subplot(2, 2, 4)
            plotMEM(stages, values['mem'])

        # Save figure
        if i == '':
            fig1.savefig('Output\\%sSDDP_solution_%s.png' % (i, mob),
                         bbox_inches='tight', pad_inches=0.5)
        else:
            fig1.savefig('Output\\solution\\%sSDDP_solution_%s.png' % (i, mob),
                         bbox_inches='tight', pad_inches=0.5)
        # Close figure
        plt.close(fig1)
    return
```