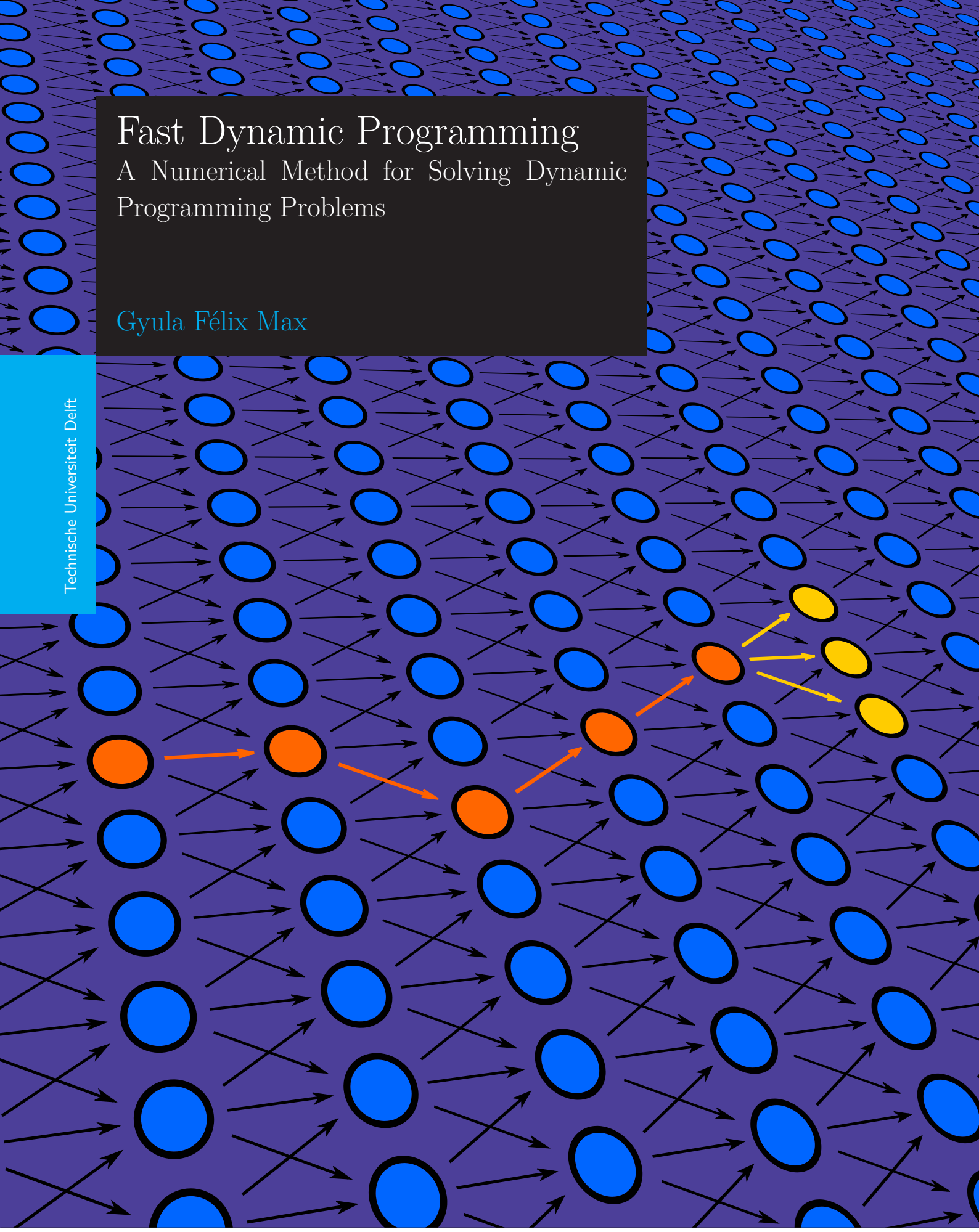
Fast Dynamic Programming

A Numerical Method for Solving Dynamic Programming Problems

Gyula Félix Max

Technische Universiteit Delft



TU Delft Delft University of Technology

Challenge the future

Fast Dynamic Programming

A Numerical Method for Solving Dynamic Programming Problems

by

Gyula Félix Max

in partial fulfillment of the requirements for the degree of

Master of Science
in Electrical Engineering

at the Delft University of Technology,
to be defended publicly on Wednesday November 27, 2019 at 14:00 AM.

Student number:	G.F. Max	4321553
Project duration:	March 1, 2019 – November 27, 2019	
Thesis committee:	Prof. dr. ir. A. van der Veen,	TU Delft
	Dr. ir. T. Keviczky,	TU Delft
	Dr. P. Mohajerin Esfahani,	TU Delft
	A. Sharifi Kolarijani,	TU Delft

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

TU Delft Delft University of Technology

Preface

The thesis delves into solving dynamic programming (DP) problems. The approach to this topic is analogous to, and was inspired by (Fast) Fourier Transform and how it is often more convenient to do computations/analysis in a dual domain. Under assumptions common in optimal control problems, these DP problems can be reformulated using convex conjugates. Further, we show that there is a potential computational gain by operating in the dual domain as opposed to direct optimization in the primal domain.

Both theoretical and practical approaches are presented to solve the problem in an unconventional way. A technique to compute solutions to DP problems, aptly named Fast Dynamic Programming, is developed which offers a numerical solution in linear time.

I would like to thank Amin Sharifi Kolarijani for his discussions on preliminary reports and his insightful feedback on the structure and the content of the thesis; and Peyman Mohajerin Esfahani for his guidence throughout the thesis.

Gyula Félix Max
Delft, November 12, 2019

Contents

1

Introduction

The objective of the thesis is to compute optimal cost functions, $J(x) : \mathcal{X} \to \mathbb{R}$, of the following form:

$$J(x) = \min_{u \in \mathcal{U}} g(x, u) + J^+(f(x, u)), \qquad \forall x \in \mathcal{X} \tag{1.1}$$

where $x \in \mathcal{X}$ denotes the state, $u \in \mathcal{U}$ is the action, $g : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the running cost and $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ models the dynamics of the underlying system. The function $J^+(x)$ refers to the cost function in the following step, in other words, the cost associated with the state where the system will end up in after taking action u. Problems having descriptions similar to Eq. (1.1) are known as dynamic programming (DP) problems. More formal definitions are given later in Chapter 2.

Many problems may be modelled and/or solved through dynamic programming: economical problems [1], inventory control [2], Hamilton–Jacobi–Bellman equation [3], shortest path algorithms [4], optimal control [5, 6], and many more. While being able to model the problem perfectly, computing its solution numerically grows exponentially with the number of dimensions. This is known as the "curse-of-dimensionality", a term originated from Bellman [7]. To mitigate this problem as well as address issues of possibly lacking the model of the underlying process, various methods were developed that fall under the category of approximate dynamic programming [8, 9] and neuro-dynamic programming [10] – or, with a term used more commonly, reinforcement learning.

To fully convey the message of the thesis we indicate its inspiration. In linear system theory, the Laplace transform often helps translating the problem into a dual domain where operations can be performed more conveniently [11]. Not only mathematical analysis simplifies after the transformation, but also numerical computations of solutions frequently require less time; even including the expense of the (inverse) transformation. Related to this, advances in signal processing have been immensely put forward starting from the mid-20th century due to development of efficient algorithms [12, 13], e.g. the Fast Fourier Transform (FFT), that can quickly compute signals' dual counterpart.

One major difference between system theory and dynamic programming, however, is that the solution operator of the latter is no longer linear in the conventional sense. We mean by the solution operator the mathematical transformation of the "input", i.e. $x(t)$ and $J^+(x)$ for system theory and DP respectively, to the "output" which is usually denoted $y(t)$ and $J(x)$ for systems and DP respectively. This nonlinearity makes handling such problems cumbersome.

Fortunately, there are two similar and related workarounds to avoid nonlinearity. First, when one changes algebras to the so-called max-plus algebra, which replaces addition ($+$) with maximization (max or "$\oplus$") and multiplication ($\times$) with traditional addition ($+$ or "$\otimes$"), the solution operator to dynamic programming problem does become linear in the max-plus sense [14]. Max-plus algebra is a powerful tool that has been extensively used for discrete event systems [15], (stochastic) optimal control problems [16–18], communication networks [19].

Second, there exists a transformation, the Legendre–Fenchel transform (LFT) or convex conjugate, that plays a role in the context of dynamic programming similar to the Laplace transform of linear systems [20]. The convex conjugate of a function $f : \mathcal{H} \to \mathbb{R}$ can be written as:

$$f^*(s) = \sup_{x \in \mathcal{H}} \langle x, s \rangle - f(x), \qquad \forall s \in \mathcal{H} \tag{1.2}$$

for some Hilbert space $\mathcal{H}$. Additionally, in max-plus algebra the LFT both symbolically and contextually resembles a form of convolution [21] which further strengthens the similarity between LFT and Laplace transform. An interesting property of conjugation is that inf-convolution of two functions becomes a simple addition in the conjugate domain. For appropriate functions f and g their inf-convolution is:

$$(f\square g)(x) = \inf_{y\in\mathcal{H}} f(y) + g(x-y) \qquad \xrightarrow{*} \qquad (f\square g)^*(s) = f^*(s) + g^*(s). \tag{1.3}$$

Notice how Eq. (1.1) and the left side of Eq. (1.3) are notationally similar, foreshadowing our method to be developed in this thesis.

Moreover, the convex conjugate can be computed with a *linear* time algorithm [22], called Linear-time Legendre transform (LLT), that "beats" the traditional $O(N \log N)$ complexity of FFT as well as complexity of previous LFT implementations [23, 24]. Besides dynamic programming, the convex conjugate emerged in other related fields, such as distance transform in computer science [25], computer vision [26], communication networks [27]. The reader is referred to [28] for an extensive survey on the applications of the conjugate.

To our best knowledge, so far no comprehensive algorithm has been proposed in the literature for solving DP problems that combines theoretical properties of conjugation with the power of the LLT algorithm and its numerical complexity. The main contribution of the thesis concerns expressing the optimal cost function in a deterministic case and is, in a sense, similar to the works [29] and [30]. However, our approach extends to a more general case with respect to [30], where a special combination of linear dynamics, with nonnegative monotone A, and cost function only dependent on the action ($g(u)$) was used. Further, it is possible to "chain" our method and solve DP problems with longer time-horizon but due to the nature of the problems we are aiming to solve there is little computational gain opposed to gain [29] and [30]. A concrete (step-by-step) algorithm is developed with numerical examples. It is worth noting that the approach presented in this thesis *does* suffer from curse-of-dimensionality. However, the computation time scales linearly with the cardinality of the state and action space as it will be explained later.

The thesis is structured as follows. The necessary mathematical preliminaries are to be found in Chapter 2. Chapter 3 shows how the DP problem may be translated by using conjugates with minor attention to implementation. In Chapter 4 numerical approaches are derived using the previous chapter as foundation. Computations of various examples together with discussion thereof are given in Chapter 5. The thesis is concluded in Chapter 6.

2

Preliminaries

The main topic revolves around convex functions and dynamic programming. To be able to convey the message of the thesis some definitions, notations and preliminary knowledge of these subjects have to be explained. There are countless of introductory materials to the topic and the reader is referred to [31] and [32] for more information about convex analysis, and [33] about dynamic programming. Further, it is assumed that the reader possesses knowledge about linear algebra and basic properties of Hilbert spaces.

2.1. General definitions

We denote the extended real line with $[-\infty, +\infty] = \mathbb{R} \cup \{-\infty\} \cup \{+\infty\}$ with the ordering $-\infty < x < \infty$ for all $x \in \mathbb{R}$. Similarly, we define $]-\infty, +\infty] = \mathbb{R} \cup \{+\infty\}$.

Definition 2.1 (Linear bounded operator [34])**.** Let V and W be normed linear spaces. A linear transformation $T : V \to W$ is said to be a continuous linear transformation, a continuous linear operator, or linear bounded operator, if it is continuous as a map between the topological spaces V and W (endowed with their norm topologies).

We denote the set of linear bounded operators by $\mathcal{B}(V, W)$.

Definition 2.2 (Separable sum of functions [32])**.** The *Hilbert direct sum* of a totally ordered family of real Hilbert spaces $(\mathcal{H}_i, \|\cdot\|_i)_{i\in I}$ is the real Hilbert space

$$\bigoplus_{i\in I} \mathcal{H}_i = \left\{ \boldsymbol{x} = (x_i)_{i\in I} \in \bigtimes_{i\in I} \mathcal{H}_i \;\middle|\; \sum_{i\in I} \|x_i\|_i^2 < +\infty \right\}, \tag{2.1}$$

equipped with the addition $(\boldsymbol{x}, \boldsymbol{y}) \mapsto (x_i + y_i)_{i\in I}$, the scalar multiplication $(\alpha, \boldsymbol{x}) \mapsto (\alpha x_i)_{i\in I}$ for scalar α, and the scalar product

$$(\boldsymbol{x}, \boldsymbol{y}) \mapsto \sum_{i\in I} \langle x_i, y_i \rangle_i. \tag{2.2}$$

Now suppose that, for every $i \in I$, $f_i : \mathcal{H}_i \to]-\infty, +\infty]$, and that if I is infinite, $\inf_{i\in I} f_i \geq 0$. Then the *sum of separable functions* is

$$\bigoplus_{i\in I} f_i : \bigtimes_{i\in I} \mathcal{H}_i \to]-\infty, +\infty] : (x_i)_{i\in I} \mapsto \sum_{i\in I} f_i(x_i). \tag{2.3}$$

We also abuse notation and write $\mathcal{H}^2 = \mathcal{H} \times \mathcal{H}$.

2.1.1. Convex sets, functions

Definition 2.3 (Convex set)**.** A set $C \subseteq \mathcal{H}$ is *convex* if for every element x and y in C, $\alpha x + (1-\alpha)y$ is also an element of C for all $0 \leq \alpha \leq 1$.

Definition 2.4 (Convex function)**.** A function $f : \mathcal{H} \to [-\infty, +\infty]$ is said to be *convex*, if its epigraph $\operatorname{epi} f = \{(x, \xi) \in \mathcal{H} \times \mathbb{R} \mid f(x) \leq \xi\}$ is a convex subset of $\mathcal{H} \times \mathbb{R}$. A function f is said to be *concave* if $-f$ is convex.

Definition 2.5 (Lower semicontinuous function [32])**.** Let $\mathcal{X}$ be a Hausdorff space. A function $f : \mathcal{X} \to [-\infty, \infty]$ is *lower semicontinuous function*, or lsc for short, if for every $x \in \mathcal{X}$ and for every $\xi \in]-\infty, f(x)]$ there exists some neighbourhood C of x such that $f(C) \subset]\xi, +\infty]$. Equivalently, f is lsc if its epigraph $\operatorname{epi} f = \{(x, \xi) \in \mathcal{H} \times \mathbb{R} \mid f(x) \leq \xi\}$ is closed.

Definition 2.6 (Domain of function)**.** The (effective) *domain* of function $f : \mathcal{X} \to [-\infty, +\infty]$ is defined as:

$$\operatorname{dom} f = \{x \in \mathcal{X} \mid f(x) < +\infty\}. \tag{2.4}$$

Let $\mathcal{X} \subset \mathbb{R}^n$ and $f : \mathcal{X} \to [-\infty, +\infty]$. We assign the value $+\infty$ to $f(y)$ where $y \notin \mathcal{X}$, that is:

$$f_{\mathcal{X}} : \mathbb{R}^n \to [-\infty, +\infty], \tag{2.5}$$

$$x \mapsto \begin{cases} f(x) & \text{if } x \in \mathcal{X}, \\ +\infty & \text{otherwise.} \end{cases} \tag{2.6}$$

We drop the $(\cdot)_{\mathcal{X}}$ notation whenever possible. Unless stated otherwise, throughout the thesis we extend the functions outside of their domain as mentioned above.

Definition 2.7 (Proper function [32])**.** A function $f : \mathcal{X} \to [-\infty, +\infty]$ is *proper* if $-\infty \notin f(\mathcal{X})$ and $f(\mathcal{X})$ is not identically $+\infty$ (the domain of f is non-empty).

Definition 2.8 (Subgradient [32])**.** Let $f : \mathcal{X} \to]-\infty, +\infty]$ be a proper function. Then $v \in \mathcal{X}$ is called a *subgradient* of f at point $y \in \mathcal{X}$ if the following holds for all $x \in \mathcal{X}$:

$$f(x) \geq \langle v \mid x - y \rangle + f(y). \tag{2.7}$$

The set of all subgradients at point y is called the *subdifferential* of f at y, and it is denoted by $\partial f(y)$.

2.2. Conjugate or Legendre–Fenchel Transform

Conjugate (or Legendre–Fenchel Transform, LFT for short) forms an essential part of our work.

Definition 2.9 (Convex conjugate)**.** Given a function $f : \mathcal{H} \to [-\infty, +\infty]$ on a Hilbert-space $\mathcal{H}$, the *convex conjugate* is defined as:

$$f^*(s) = \sup_{x \in \mathcal{H}} \big(\langle x \mid s \rangle - f(x) \big), \qquad \forall s \in \mathcal{H}, \tag{2.8}$$

where $\langle x \mid s \rangle$ is the inner product defined on $\mathcal{H}$. The *biconjugate* of f is obtained by performing the conjugation twice: $f^{**} = (f^*)^*$.

Let the set of convex functions on $\mathcal{H}$ be $\mathscr{C}(\mathcal{H})$, the set of convex lower semicontinuous functions on $\mathcal{H}$ be $\Gamma(\mathcal{H})$, and the set of proper convex lower semicontinuous functions on $\mathcal{H}$ be $\Gamma_0(\mathcal{H})$.

Proposition 2.10. *The conjugation operator $(\cdot)^*$ maps the set of functions $f : \mathcal{H} \to [-\infty, +\infty]$ to $\Gamma(\mathcal{H})$.*

Proof. See [32, Proposition 13.11]. □

Corollary 2.10.1. *The conjugation operator $(\cdot)^*$ maps the set of proper convex lower semicontinuous functions acting on $\mathcal{H}$ onto itself, i.e. $(\cdot)^* : \Gamma_0(\mathcal{H}) \to \Gamma_0(\mathcal{H})$.*

Proof. Proposition 2.10 together with [32, Proposition 13.9(ii)]. □

Theorem 2.11 (Fenchel–Moreau theorem [32])**.** *Let $f : \mathcal{H} \to]-\infty, +\infty]$ be proper. Then f is lower semicontinuous and convex if and only if $f = f^{**}$. In this case, f^* is proper as well.*

Definition 2.12. Let $\mathcal{H}$ be a Hilbert space and $\mathcal{K}$ be a real Hilbert space. The *infimal postcomposition* is defined as:

$$L \rhd f : \mathcal{K} \to [-\infty, +\infty] : y \mapsto \inf f(L^{-1}\{y\}) = \inf_{\substack{x \in \mathcal{H} \\ Lx = y}} f(x), \tag{2.9}$$

for an operator $L : \mathcal{H} \to \mathcal{K}$ and function $f : \mathcal{H} \to [-\infty, +\infty]$. The infimal postcomposition is *exact* if for all $y \in \mathcal{H}$ the infimum is reached, i.e. $(L \rhd f)(y) = \min_{x \in L^{-1}\{y\}} f(x)$

Definition 2.13. If $\mathcal{H}$ is a Hilbert space and $\mathcal{K}$ is a real Hilbert space and $T : \mathcal{H} \to \mathcal{K}$ is a bounded linear operator then the *adjoint* of T is the unique operator $T^{adj} : \mathcal{K} \to \mathcal{H}$ that satisfies $\langle Tx \mid y \rangle = \langle x \mid T^{adj} y \rangle$ for all $x \in \mathcal{H}$ and all $y \in \mathcal{K}$.

Remark 2.14. We deliberately avoid the usage of the symbols $(\cdot)^\top$ and $(\cdot)^*$ to express the adjoint of an operator, as it is otherwise common in literature. It is hoped that less confusion is generated this way.

In the future it will be extremely useful to have these lemmas.

Lemma 2.15. *Let $f : \mathcal{H} \to]-\infty, +\infty]$, and $L \in \mathcal{B}(\mathcal{H}, \mathcal{H})$ bijective, i.e. L^{-1} exists and unique. Then $(f \circ L)^* = f^* \circ (L^{-1})^{adj}$.*

Proof. Let $s \in \mathcal{H}$. Then:

$$\begin{aligned}
(f \circ L)^*(s) &= \sup_{x \in \mathcal{H}} \{\langle s \mid x \rangle - (f \circ L)(x)\} && (2.10)\\
&= \sup_{x \in \mathcal{H}} \left\{\langle (L^{-1})^{adj} s \mid Lx \rangle - f(Lx)\right\} && (2.11)\\
&= \sup_{x' \in \mathcal{H}} \left\{\langle (L^{-1})^{adj} s \mid x' \rangle - f(x')\right\} && (2.12)\\
&= f^*((L^{-1})^{adj} s). && (2.13)
\end{aligned}$$

Since $L \in \mathcal{B}(\mathcal{H}, \mathcal{H})$ we have $x' = Lx \in \mathcal{H}$ for all $x \in \mathcal{H}$. □

Lemma 2.16 (From [32])**.** *Let $f : \mathcal{H} \to]-\infty, +\infty]$, and $L \in \mathcal{B}(\mathcal{H}, \mathcal{K})$ with $\mathcal{K}$ real Hilbert space. Then $(L \rhd f)^* = f^* \circ L^{adj}$.*

Proof. The proof follows [32, Proposition 13.21(iv)]. Let $s \in \mathcal{K}$. Then:

$$\begin{aligned}
(L \rhd f)^*(s) &= \sup_{y \in \mathcal{K}} \left\{ \langle s \mid y \rangle - \inf_{\substack{x \in \mathcal{H} \\ Lx = y}} f(x) \right\} && (2.14)\\
&= \sup_{y \in \mathcal{K}} \sup_{\substack{x \in \mathcal{H} \\ Lx = y}} \{\langle s \mid y \rangle - f(x)\} && (2.15)\\
&= \sup_{y \in \mathcal{K}} \sup_{x \in L^{-1}\{y\}} \{\langle s \mid Lx \rangle - f(x)\} && (2.16)\\
&= \sup_{x \in \mathcal{H}} \left\{\langle L^{adj} s \mid x \rangle - f(x)\right\} && (2.17)\\
&= f^*(L^{adj} s). && (2.18)
\end{aligned}$$

We mean by $x \in L^{-1}\{y\}$ the preimage of L, that is $x \in \mathcal{X}' = \{x \in \mathcal{H} \mid Lx = y\}$. Because L maps from $\mathcal{H}$ to $\mathcal{K}$, and all elements of $\mathcal{K}$ are covered by the operator L, the two supremums can be merged together. □

Remark 2.17. The adjoint of an operator defined by matrix $A \in \mathbb{R}^{n \times m}$ is $A^{adj} = A^\top \in \mathbb{R}^{m \times n}$. Thus Lemma 2.15 reads $(f \circ A)^* = f^* \circ A^{-\top}$, and Lemma 2.16 yields $(A \rhd f)^* = f^* \circ A^\top$.

Lemma 2.18 (Conjugate of Hilbert sum)**.** *Let $(\mathcal{H}_i)_{i\in I}$ be a totally ordered finite family of real Hilbert spaces and, for every $i \in I$, let $f_i : \mathcal{H}_i \to]-\infty, +\infty]$. Then:*

$$\left(\bigoplus_{i\in I} f_i\right)^* = \bigoplus_{i\in I} f_i^*. \tag{2.19}$$

Proof. See [32, Proposition 13.27]. □

Corollary 2.18.1. *We have $(f_1 \oplus f_2)^*(s_1, s_2) = f_1^*(s_1) \oplus f_2^*(s_2)$ for $f_i : \mathcal{H}_i \to]-\infty, +\infty]$ where $i \in \{1, 2\}$.*

2.3. Dynamic programming

Throughout the thesis we will talk about a system, be it a collection of real objects or some abstract construction, which can be characterized by its *states*. These states define the particular properties of the system and are denoted by x_t. The subscript t hints that these states may change with respect to time, and, in general, this change may be influenced by some *action* (u_t) that we have full control over. This relationship is captured by the dynamics of the system:

$$x_{t+1} = f_t(x_t, u_t), \tag{2.20}$$

for $t = 0, 1, \ldots, T-1$ with state $x_t \in \mathcal{X}_t$, control $u_t \in \mathcal{U}_t$ and disturbance $w_t \in \mathcal{W}_t$. In this work, we restrict ourselves to the deterministic case where given a state of and an input to the system we can perfectly determine what the next state is. In general, the dynamics may be extended to incorporate some external disturbances/noise (e.g. $f(x, u, w)$ for disturbance w) but this lies outside of the scope of this thesis.

The last pieces on the board are running cost (g_t) and terminal cost (J_T). The former represents the costs of being in state x_t and applying action u_t as $g_t(x_t, u_t) : \mathcal{X}_t \times \mathcal{U}_t \to \mathbb{R}$. The latter stands for costs of being in state x_T at the very end of the time horizon, and is $J_T(x_T) : \mathcal{X}_T \to \mathbb{R}$. With these definitions the overall costs can be defined as:

$$\hat{J}_t(\boldsymbol{x}_t, \boldsymbol{u}_t) = \sum_{\tau=t}^{T-1} g_\tau(x_\tau, u_\tau) + J_T(x_T), \tag{2.21}$$

subject to (2.20) where $\boldsymbol{x}_t = [x_t^\top \; x_{t+1}^\top \; \cdots \; x_T^\top]^\top$ and, likewise, $\boldsymbol{u}_t = [u_t^\top \; u_{t+1}^\top \; \cdots \; u_T^\top]^\top$. We are interested in finding the best control in the following sense:

$$J_t(x_t) = \min_{u_\tau \in \mathcal{U}_\tau} \left\{ \sum_{\tau=t}^{T-1} g_\tau(x_\tau, u_\tau) + J_T(x_T), \right\} \tag{2.22}$$

subject to Eq. (2.20) which is referred to as cost function. It is well-known that Eq. (2.22) satisfies the Bellman equation. Roughly speaking, if we would have perfect knowledge of the cost function in the future, i.e. onwards $t+1$, we could evaluate our current cost function as:

$$J_t(x_t) = \min_{u_t \in \mathcal{U}_t} \{g_t(x_t, u_t) + J_{t+1}(f_t(x_t, u_t))\}, \tag{2.23}$$

or, more compactly,

$$J_t(x_t) \stackrel{\triangle}{=} \mathsf{DP}_t[J_{t+1}](x_t), \tag{2.24}$$

with non-linear operator DP_t. Note that this DP_t operator encompasses 3 main things, all with respect to a particular time instant: the dynamics of the system, the running cost, and the "best" action (minimization). Repeated application of this operator in a backward fashion for $t = T-1, \ldots, 0$ is known as the backwards Dynamic Programming Algorithm (DPA). In particular, if the dynamics and running costs are time-invariant then $\mathsf{DP}_t = \mathsf{DP}$ is also time-invariant and we have:

$$J_t(x_t) = \underbrace{\mathsf{DP}[\mathsf{DP}[\cdots \mathsf{DP}[J_T] \cdots]]}_{T-t \text{ times}}(x_t) = \mathsf{DP}^{T-t}[J_T](x_t). \tag{2.25}$$

We will abuse notation and drop subscripts t whenever possible. Furthermore, given the repetitive nature of the DP operator our main focus will be reformulating a single step to gain computational advantage.

3

Theory – Continuous spaces

The main objective in this chapter is to reformulate the DP problem using conjugates. Our motivation is twofold: the similarity between Legendre–Fenchel transform in DP and Laplace transform in linear system theory, as well as the computational ease of conjugation. Throughout this chapter we use $x, \tilde{x}$ for primal and $s, \tilde{s}$ for dual variables in the sense of conjugation. Figure 3.1 shows an overview of the presented methods. First we set up the scene with introducing perturbed DP operator. Then equivalence is shown between direct computation of the DP operator and biconjugation. Afterwards, we focus on expressing DP operator in the conjugate domain under different assumptions, so that biconjugation can be readily applied. We will show many variants to DP problem starting from a more general one to more specific.

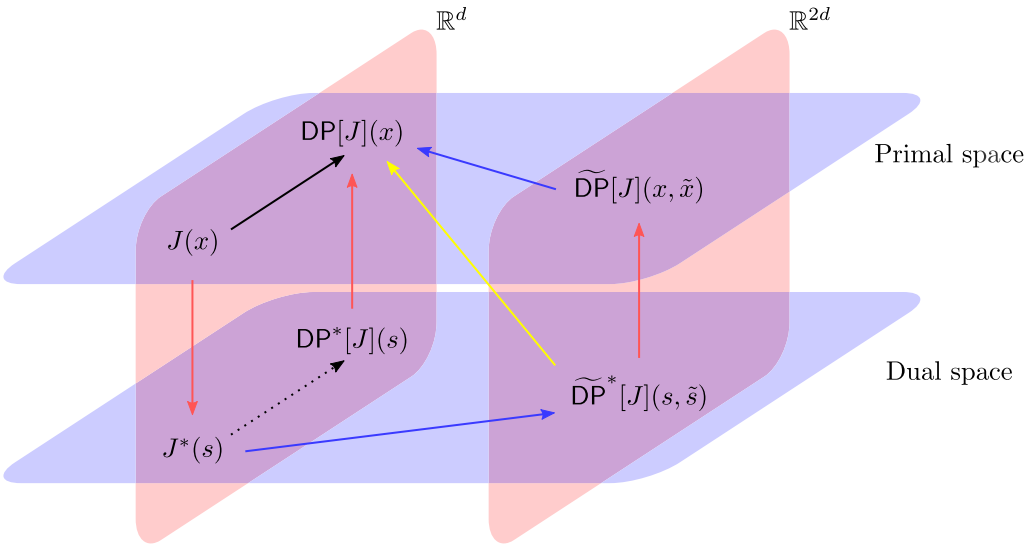


Figure 3.1: Summary of our approach. Our ultimate goal is to compute $\mathsf{DP}[J](x)$ from $J(x)$ indicated with black arrow. Going "sideways" (blue arrows) accounts to simple linear operations (additions and linear transformations) while going vertically (red arrows) accounts to Legendre–Fenchel Transform. There is a shortcut (yellow arrow), which is a composite of LFT and linear transformation, that may be used to skip a step.

3.1. Setup

Unless stated otherwise, we assume the following throughout the chapter:

Assumption $\mathcal{A}$.1.

$$\begin{cases} \mathcal{X} \subseteq \mathbb{R}^n & \text{Hilbert space} \\ \mathcal{U} \subseteq \mathbb{R}^m & \text{convex, compact} \\ f(x,u) \in \mathcal{B}(\mathcal{X} \times \mathcal{U}, \mathcal{X}) & \\ J(x) : \mathbb{R}^n \to]-\infty, +\infty] & J \in \Gamma_0(\mathcal{X}) \text{ with } \operatorname{dom} J \subseteq \mathcal{X} \\ g(x,u) : \mathbb{R}^n \times \mathbb{R}^m \to]-\infty, +\infty] & g \in \Gamma_0(\mathcal{X} \times \mathcal{U}) \text{ with } \operatorname{dom} g \subseteq \mathcal{X} \times \mathcal{U}, \end{cases}$$

where $\mathcal{B}(\mathcal{X} \times \mathcal{U}, \mathcal{X})$ represents the set of linear bounded operators acting on $\mathcal{X} \times \mathcal{U}$ to $\mathcal{X}$. We also assume that there exists $x \in \mathcal{X}$ and $u \in \mathcal{U}$ such that $f(x,u) \in \operatorname{dom} J$.

We repeat here the definition of the Dynamic Programming (2.23) for convenience.

Definition 3.1. The Dynamic Programming operator, DP, is defined as:

$$\mathsf{DP}[J](x) = \min_{u \in \mathcal{U}} \{g(x,u) + J(f(x,u))\}, \qquad \forall x \in \mathcal{X}, \tag{3.1}$$

with given running cost $g(x,u) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ and given dynamics $f(x,u) : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$.

We refer to Equation (3.1) as a single step in DP. Note that the very last assumption in ($\mathcal{A}$.1) basically implies that the minimum of a single step is real-valued at least once, thus making $\mathsf{DP}[J] \not\equiv +\infty$. This assumption is necessary from a mathematical point of view – as we will see later – however, any real system should satisfy this condition automatically. Without it, there would be no control/action that would lead to a state with *finite* cost.

In the following it will be useful to define a *perturbed* Dynamic Programming operator.

Definition 3.2. The *perturbed* Dynamic Programming operator, $\widetilde{\mathsf{DP}}$, is defined as:

$$\widetilde{\mathsf{DP}}[J](x,\tilde{x}) = \min_{u \in \mathcal{U}} \{g(x,u) + J(f(x,u) + \tilde{x})\}, \qquad \forall x \in \mathcal{X}, \tag{3.2}$$

with given running cost $g(x,u) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ and given dynamics $f(x,u) : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$, similarly to before.

From these definitions it is clear that $\mathsf{DP}[J](x) = \widetilde{\mathsf{DP}}[J](x,0)$. We show that $\widetilde{\mathsf{DP}}$ can expressed with the Hilbert direct sum and infimal postcomposition.

Lemma 3.3. *The perturbed DP operator $\widetilde{\mathsf{DP}}$ can be expressed as:*

$$\widetilde{\mathsf{DP}}[J](x,\tilde{x}) = \min_{y \in \mathcal{Y}} \left\{ (g \oplus J)(y) : L_f y = \begin{bmatrix} x \\ \tilde{x} \end{bmatrix} \right\} = (L_f \rhd (g \oplus J))(x,\tilde{x}), \tag{3.3}$$

where

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x \\ u \\ f(x,u) + \tilde{x} \end{bmatrix} \in \mathcal{X} \times \mathcal{U} \times \mathcal{X} =: \mathcal{Y} \tag{3.4}$$

and

$$L_f : \mathcal{Y} \to \mathcal{X} \times \mathcal{X} : y \mapsto \begin{bmatrix} y_1 \\ y_3 - f(y_1, y_2) \end{bmatrix}. \tag{3.5}$$

Proof. Notice that $x = [I \;\; 0][y_1^\top \;\; y_2^\top]^\top$ and $\tilde{x} = y_3 - f(y_1, y_2)$, with which we can write L_f as follows:

$$\begin{bmatrix} x \\ \tilde{x} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_3 - f(y_1, y_2) \end{bmatrix} = L_f y. \tag{3.6}$$

With these definitions Equation (3.2) becomes:

$$\begin{aligned}\widetilde{\mathsf{DP}}[J](x,\tilde{x}) &= \min_{u\in\mathcal{U}}\{g(x,u)+J(f(x,u)+\tilde{x})\}\\ &= \min_{\substack{y\in\mathcal{Y}\\ [I\ \ 0][y_1^\top\ \ y_2^\top]^\top=x\\ y_3-f(y_1,y_2)=\tilde{x}}}\{g(y_1,y_2)+J(y_3)\}\\ &= \min_{\substack{y\in\mathcal{Y}\\ L_f y=[x^\top\ \ \tilde{x}^\top]^\top}}\{(g\oplus J)(y)\},\end{aligned} \tag{3.7}$$

which concludes the proof. □

Proposition 3.4. *Under assumptions* ($\mathcal{A}$.1) *and* $J\in\mathscr{C}(\mathcal{X})$ *the DP operators preserve convexity, i.e. we have* $\mathsf{DP}:\mathscr{C}(\mathcal{X})\to\mathscr{C}(\mathcal{X})$ *and* $\widetilde{\mathsf{DP}}:\mathscr{C}(\mathcal{X})\to\mathscr{C}(\mathcal{X}^2)$.

Proof. The proof follows [31, Chapter 3.2.5]. We work from inside out and we start with DP. From the assumptions $g+J\circ f$ is convex in both x and u since it is the sum of two convex functions. Consequently, $\operatorname{epi}(g+J\circ f)$ is convex. Taking the minimum with respect to u accounts to a projection of a convex set on some of its components, which is in turn convex. Hence, $\min_u g(x,u)+J(f(x,u))$ is convex in x.

The proof is similar for $\widetilde{\mathsf{DP}}$ with the addition that $J(f(x,u)+\tilde{x})$ is convex in x,u and $\tilde{x}$. Fix $\tilde{x}\in\mathcal{X}$, then $f(x,u)+\tilde{x}$ is just an affine composition which does not alter convexity of $J(f(x,u)+\tilde{x})$. This holds for all $\tilde{x}\in\mathcal{X}$, thus, following the same steps as before, we get $\min_u g(x,u)+J(f(x,u)+\tilde{x})$ is convex.

Another, perhaps more intuitive way of proving the statement is that the partial infimum of a jointly convex function with respect to one of its arguments is also convex (see [35]). □

Notice that remaining within the realms of convex functions after applying DP has remarkable advantages in the context of conjugation.

Proposition 3.5. *Under assumptions* ($\mathcal{A}$.1) *we have*

$$\mathsf{DP}^{**}[J]=\mathsf{DP}[J] \qquad \textit{and} \qquad \widetilde{\mathsf{DP}}^{**}[J]=\widetilde{\mathsf{DP}}[J]. \tag{3.8}$$

Proof. According to the Fenchel–Moreau theorem (Theorem 2.11), it suffices to prove that $\mathsf{DP}[J]$ is proper, convex and lsc. Convexity is proven in Proposition 3.4. We only have to show properness and lower semicontinuity of $\mathsf{DP}[J]$.

Properness follows directly from the assumptions: $g>-\infty$ and $J\circ f>-\infty$ then $g+J\circ f>-\infty$. Therefore, $\min g+J\circ f>-\infty$. To show that $\min g+J\circ f\not\equiv+\infty$ it suffices to have an $x\in\mathcal{X}$ and $u\in\mathcal{U}$ such that $f(x,u)\in\operatorname{dom}J$, which is included in the assumptions as well.

To prove lower semicontinuity we invoke two lemmas from [32]. Lemma 1.27 of [32] says that the (positive weighted) sum of lower semicontinuous functions is lower semicontinuous. In our case $g\in\Gamma_0$ is lsc by definition, and $J\circ f$ is lsc by of [32, Proposition 9.5], leading to $g+J\circ f$ being lower semicontinuous as well. Lemma 1.29 of [32] states that the marginal function defined as $x\mapsto\inf_{c\in C}\phi(x,c)=\min_{c\in C}\phi(x,c)$ for lower semicontinuous ϕ is lower semicontinuous when $C\subseteq\mathbb{R}^k$ is compact space.[1] In our case $\phi(x,u)=g(x,u)+J(f(x,u))$, resp. $\phi(x,\tilde{x},u)=g(x,u)+J(f(x,u)+\tilde{x})$, and $\mathcal{U}\subseteq\mathbb{R}^m$ is compact by assumption therefore $\mathsf{DP}[J]$, resp. $\widetilde{\mathsf{DP}}[J]$, is lower semicontinuous. □

Remark 3.6. Convexity is necessary to ensure equality of $\mathsf{DP}[J]$ and $\mathsf{DP}^{**}[J]$ (and their perturbed version) *at all* x. In general, $f^{**}\le f$, for some proper function f, with equality only on points which contribute to the convex hull of f if f has a continuous affine minorant (see [32, Proposition 13.39]).

Let us to emphasize the importance of Proposition 3.5 from a computational point of view. The proposition suggests that $\mathsf{DP}[J]$ can be *directly* computed from $\mathsf{DP}^*[J]$, and similarly their perturbed versions. Therefore, if $\mathsf{DP}^*[J]$ is obtained more easily, it becomes $\mathsf{DP}[J]$ at the cost of conjugation. This

[1] It is sufficient for C to be a compact Hausdorff space in order to guarantee lower semicontinuity of the marginal function. This may lead to a more relaxed condition on $\mathcal{U}$. In our case $C=\mathcal{U}\subseteq\mathbb{R}^m$, which is Hausdorff.

is also hinted on Figure 3.1 by staying in $\mathbb{R}^n$. Unfortunately, the conjugate of $\mathsf{DP}[J]$ results in a similar optimization problem as in Equation (3.1), but instead of minimization there is maximization:

$$\begin{aligned}\mathsf{DP}^*[J](s) &= \sup_{x\in\mathcal{X}}\{\langle s \mid x\rangle - \mathsf{DP}[J](x)\}\\ &= \sup_{x\in\mathcal{X}}\left\{\langle s \mid x\rangle - \min_{u\in\mathcal{U}}\{g(x,u)+J(f(x,u))\}\right\}\\ &= \sup_{x\in\mathcal{X}}\max_{u\in\mathcal{U}}\{\langle s \mid x\rangle - g(x,u) - J(f(x,u))\}\\ &= \sup_{p\in\mathcal{X}\times\mathcal{U}}\{\langle (s,0) \mid p\rangle - g(p) - J(f(p))\}\,. \end{aligned} \tag{3.9}$$

Computing $\mathsf{DP}^*[J]$ does not leverage computational complexity. The conjugate of the perturbed operator, however, becomes more tangible as we will see in the next section.

3.2. Expressing DP in the conjugate domain

With these preliminaries behind we are finally ready to derive what is the cornerstone of the thesis.

Theorem 3.7. *The conjugate of the perturbed DP operator* $\widetilde{\mathsf{DP}}^*$ *can be expressed as:*

$$\widetilde{\mathsf{DP}}^*[J](s,\tilde{s}) := \left(\widetilde{\mathsf{DP}}[J]\right)^*(s,\tilde{s}) = \left((g\oplus J)^*\circ L_f^{adj}\right)(s,\tilde{s}) \tag{3.10}$$

$$= g^*\left((s,0) - f^{adj}(\tilde{s})\right) + J^*(\tilde{s}) \tag{3.11}$$

for $s,\tilde{s}\in\mathcal{S}\subseteq\mathbb{R}^n$*, where* $\mathcal{S}$ *is the dual domain of* $\mathcal{X}$*,* $L_f^{adj}:\mathcal{X}\times\mathcal{X}\to\mathcal{X}\times\mathcal{U}\times\mathcal{X}$ *is the adjoint of* L_f *given in* (3.6)*, and* $f^{adj}:\mathcal{X}\to\mathcal{X}\times\mathcal{U}$ *is the adjoint of* f*.*

Proof. Let $s,\tilde{s}\in\mathcal{S}\subseteq\mathbb{R}^n$. Using Lemma 2.16 and Lemma 2.16 the convex conjugate of $\widetilde{\mathsf{DP}}$ can be written as:

$$\begin{aligned}\widetilde{\mathsf{DP}}^*[J](s,\tilde{s}) &= (L_f \rhd (g\oplus J))^*(s,\tilde{s})\\ &= (g\oplus J)^*(L_f^{adj}(s,\tilde{s}))\\ &= g^*(L_f^{adj}(s,\tilde{s})) \oplus J^*(L_f^{adj}(s,\tilde{s})).\end{aligned} \tag{3.12}$$

The same result can be achieved with ordinary notations:

$$\begin{aligned}\widetilde{\mathsf{DP}}^*[J](s,\tilde{s}) &= \left(\min_{\substack{y\in\mathcal{Y}\\ L_f y=[x^\top\ \tilde{x}^\top]^\top}}\{(g\oplus J)(y)\}\right)^*(s,\tilde{s})\\ &= \sup_{(w,\tilde{w})\in\mathcal{X}\times\mathcal{X}}\left\{\langle (w,\tilde{w}) \mid (s,\tilde{s})\rangle - \min_{\substack{y\in\mathcal{Y}\\ L_f y=[w^\top\ \tilde{w}^\top]^\top}}\{(g\oplus J)(y)\}\right\}\\ &= \sup_{(w,\tilde{w})\in\mathcal{X}\times\mathcal{X}}\ \max_{\substack{y\in\mathcal{Y}\\ L_f y=[w^\top\ \tilde{w}^\top]^\top}}\left\{\langle (w,\tilde{w}) \mid (s,\tilde{s})\rangle - (g\oplus J)(y)\right\}\\ &= \sup_{(w,\tilde{w})\in\mathcal{X}\times\mathcal{X}}\ \max_{\substack{y\in\mathcal{Y}\\ L_f y=[w^\top\ \tilde{w}^\top]^\top}}\left\{\langle L_f y \mid (s,\tilde{s})\rangle - (g\oplus J)(y)\right\}\\ &= \sup_{(w,\tilde{w})\in\mathcal{X}\times\mathcal{X}}\ \max_{y\in L_f^{-1}[w^\top\ \tilde{w}^\top]^\top}\left\{\langle L_f y \mid (s,\tilde{s})\rangle - (g\oplus J)(y)\right\}\end{aligned} \tag{3.13}$$

$$\begin{aligned}&= \sup_{y\in\mathcal{Y}}\left\{\langle y \mid L_f^{adj}(s,\tilde{s})\rangle - (g\oplus J)(y)\right\}\\ &= (g\oplus J)^*(L_f^{adj}(s,\tilde{s}))\\ &= g^*(L_f^{adj}(s,\tilde{s})) \oplus J^*(L_f^{adj}(s,\tilde{s})).\end{aligned} \tag{3.14}$$

We mean by $y = L_f^{-1}[w^\top \ \ \tilde{w}^\top]^\top$ in Equation (3.13) the pre-image of L_f, that is $y \in \mathcal{Y}' = \{y \in \mathcal{Y} \mid L_f y = [w^\top \ \ \tilde{w}^\top]^\top\}$. Now only Equation (3.11) needs to be shown. For that to hold the following needs to be verified:

$$L_f^{adj}(s, \tilde{s}) = \begin{bmatrix} \begin{bmatrix} s \\ 0 \end{bmatrix} - f^{adj}(\tilde{s}) \\ \tilde{s} \end{bmatrix}. \tag{3.15}$$

Indeed, this follows from the definition of L_f and that of the adjoint operator:

$$\begin{aligned} \langle L_f y \mid (s, \tilde{s}) \rangle &= \langle y_1 \mid s \rangle + \langle y_3 \mid \tilde{s} \rangle - \langle f(y_1, y_2) \mid \tilde{s} \rangle \\ &= \langle y_1 \mid s \rangle + \langle y_3 \mid \tilde{s} \rangle - \langle (y_1, y_2) \mid f^{adj}(\tilde{s}) \rangle \\ &= \langle y_3 \mid \tilde{s} \rangle + \langle (y_1, y_2) \mid (s, 0) - f^{adj}(\tilde{s}) \rangle \\ &= \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} \begin{bmatrix} s \\ 0 \end{bmatrix} - f^{adj}(\tilde{s}) \\ \tilde{s} \end{bmatrix} \\ &= \langle y \mid L_f^{adj}(s, \tilde{s}) \rangle, \end{aligned} \tag{3.16}$$

which concludes the proof. □

Corollary 3.7.1. *If $f(x, u) = Ax + Bu$ for $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$ then $\widetilde{\mathsf{DP}}^*[J]$ simplifies to*

$$\widetilde{\mathsf{DP}}^*[J](s, \tilde{s}) = g^*(s - A^\top \tilde{s}, -B^\top \tilde{s}) + J^*(\tilde{s}). \tag{3.17}$$

Proof. In this case $L_f y = (x, \tilde{x})$ defined by Equation (3.6) becomes

$$L_f y = \begin{bmatrix} y_1 \\ y_3 - f(y_1, y_2) \end{bmatrix} = \begin{bmatrix} y_1 \\ y_3 - Ay_1 - By_2 \end{bmatrix} = \left[\begin{array}{cc|c} I & 0 & 0 \\ -A & -B & I \end{array}\right] \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} =: [L_1 \mid L_2] y. \tag{3.18}$$

Using the fact that $L_f^{adj} = L^\top = [L_1^\top \ \ L_2^\top]^\top$, substitution into Equation (3.10) yields:

$$\begin{aligned} \widetilde{\mathsf{DP}}^*[J](s, \tilde{s}) &= \left((g \oplus J)^* \circ L_f^{adj} \right)(s, \tilde{s}) \\ &= \left((g \oplus J)^* \circ L^\top \right)(s, \tilde{s}) \\ &= g^*(L_1^\top(s, \tilde{s})) + J^*(L_2^\top(s, \tilde{s})) \\ &= g^*(s - A^\top \tilde{s}, -B^\top \tilde{s}) + J^*(\tilde{s}). \end{aligned}$$

□

Remark 3.8. There is a clear distinction between $f^* : \mathcal{S} \times \mathcal{V} \to \mathcal{S}$ and $f^{adj} : \mathcal{X} \to \mathcal{X} \times \mathcal{U}$.

Remark 3.9. The existence of L_f^{adj} is trivial when L_f can be represented by a matrix, i.e. $L_f \in \mathbb{R}^{(2n) \times (2n+m)}$. If $T : \mathcal{H} \to \mathcal{K}$ is a bounded *non-linear* operator then its adjoint can still be defined (see for example [36] or [37]). It may be possible to use this fact to derive results similar to Equation (3.17) for non-linear dynamics, however, this lies outside of the scope of the thesis and is left for future work.

Note that $\mathsf{DP}[J](x) = \widetilde{\mathsf{DP}}[J](x, 0)$ per definition and under Assumption $\mathcal{A}.1$ the following holds $\mathsf{DP}[J](x) = \widetilde{\mathsf{DP}}^{**}[J](x, 0)$. Combining this fact with Equation (3.17) we reach a method of computing $\mathsf{DP}[J](x)$ for *linear* dynamics which is outlined on Figure 3.2. Notice the shortcut from $\widetilde{\mathsf{DP}}^*[J]$ to $\mathsf{DP}[J]$ denoted with a yellow arrow. This is possible because when taking the conjugation there is a freedom of choice at which points we would like to evaluate the conjugate. Thus, it is easy to compute points belonging to $(x, 0)$ instead of all possible $(x, \tilde{x})$.

3.3. Improving conjugate expression for the linear case

Our results in the previous section provide a crude way of computing solutions to DP problem with linear dynamics using conjugation. In this section we dive deeper into the linear dynamics, and will assume the following:

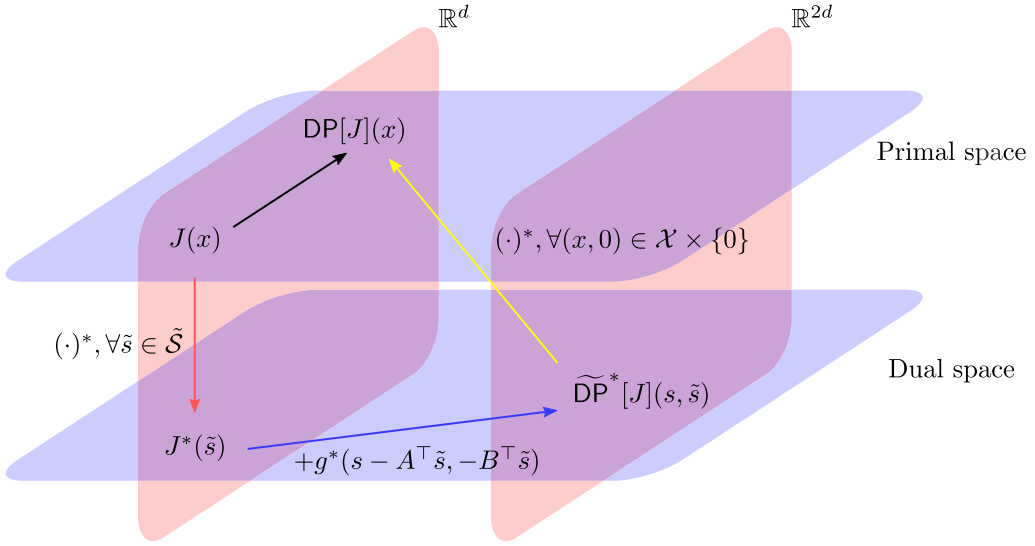


Figure 3.2: Computation of $\mathsf{DP}[J](x)$ by taking detour in the conjugate domain. There is a "shortcut" from $\widetilde{\mathsf{DP}}^*[J]$ to $\mathsf{DP}[J]$ indicated with the yellow arrow. We only need to evaluate $\widetilde{\mathsf{DP}}^{**}[J](x,0)$, and thus going directly from $\mathbb{R}^{2d}$ to $\mathbb{R}^d$.

Assumption $\mathcal{A}$.2. Assume $\mathcal{A}$.1, and that the dynamics of the system is linear, that is:

$$f(x,u) = Ax + Bu, \tag{3.19}$$

where $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$.

Looking at Equation (3.17) it is apparent that minimization disappeared[2], and $\widetilde{\mathsf{DP}}^*[J]$ became a sum of two functions. However, evaluating $g^*(s - A^\top \tilde{s}, \cdot)$ efficiently is difficult because points/sets need to be computed on the so-called Minkowski-sum of $\mathcal{S} \ni s$ and $\tilde{\mathcal{S}} \ni -A^\top \tilde{s}$. The Minkowski-sum of two sets $\mathcal{A}$ and $\mathcal{B}$ is defined as $\mathcal{A} + \mathcal{B} = \{a + b \mid a \in \mathcal{A}, b \in \mathcal{B}\}$. While the Minkowski-sum does not increase the dimensionality of the resulting set, it is highly desirable to eliminate the Minkowski-sum from any expression when thinking about discrete sets; in worst case, if $|\mathcal{A}| = N_A$ and $|\mathcal{B}| = N_B$ then $|\mathcal{A}+\mathcal{B}| = N_A N_B$. Unfortunately, the worst case prevails in most practical purposes as does in our case, e.g. when the elements of A are not (small) integers or rationals with small denominators.

Our goal is to derive an expression for $\widetilde{\mathsf{DP}}^*[J]$ in which each function argument contains only a single variable. To overcome the computational burden we modify the expression for the constraints defined by Eq. (3.18) to our advantage. Let us introduce another modification to the DP operator.

Definition 3.10. The *modified* perturbed DP operator, $\widetilde{\mathsf{DP}}_H$, is defined as:

$$\begin{aligned}\widetilde{\mathsf{DP}}_H[J](w,\tilde{w}) &= (\widetilde{\mathsf{DP}}[J] \circ H)(w,\tilde{w}) \\ &= \min_{y\in\mathcal{Y}} \left\{ (g \oplus J)(y) : L_f y = H \begin{bmatrix} w \\ \tilde{w} \end{bmatrix} \right\}, \qquad \forall w, \tilde{w} \in \mathcal{X},\end{aligned} \tag{3.20}$$

where $H : \mathcal{X}^2 \to \mathcal{X}^2$ and $H \in \mathcal{B}(\mathcal{X}^2, \mathcal{X}^2)$ is bijective.

It is easy to verify from the definition that the relationship between all DP operators is given as:

$$\mathsf{DP}[J](x) = \widetilde{\mathsf{DP}}[J](x,0) = \widetilde{\mathsf{DP}}_H[J](w,\tilde{w})\Big|_{(w,\tilde{w})=H^{-1}(x,0)} = \widetilde{\mathsf{DP}}_H[J](H^{-1}(x,0)). \tag{3.21}$$

Note that since H is bijective, H^{-1} (uniquely) exists.

Proposition 3.11. *Under Assumption $\mathcal{A}$.1 we have:*

$$\widetilde{\mathsf{DP}}_H^{**}[J] = \widetilde{\mathsf{DP}}_H[J]. \tag{3.22}$$

[2]Disappeared is a word used rather loosely here: minimization is hidden behind the conjugation. However, if g^* and J^* are known – or are easily obtained – then $\widetilde{\mathsf{DP}}^*[J]$ really *is* simply a sum of functions.

Proof. Similarly to Proposition 3.5, it is sufficient to show that $\widetilde{\mathsf{DP}}_H[J]$ is proper, convex, lsc according to the Fenchel–Moreau theorem. By [32, Proposition 9.5] $\widetilde{\mathsf{DP}}_H[J]$ is convex and lsc since $\widetilde{\mathsf{DP}}[J]$ is convex and lsc and $H \in \mathcal{B}(\mathcal{H}, \mathcal{H})$. Properness is investigated next. Since the input is transformed (linearly) first and $\widetilde{\mathsf{DP}}[J] > -\infty$, the transformation cannot alter the functions maxima and minima, leading to $\widetilde{\mathsf{DP}}_H[J] > -\infty$. Further, there has to be a point $w, \tilde{w} \in \mathcal{X}$ for which $\widetilde{\mathsf{DP}}[J](w, \tilde{w}) < +\infty$ because of the properness of $\widetilde{\mathsf{DP}}[J]$. Hence, $\widetilde{\mathsf{DP}}_H[J](H^{-1}(w, \tilde{w})) < +\infty$ and thus $\widetilde{\mathsf{DP}}_H[J]$ is proper. □

Proposition 3.12. *Under Assumption A.2, the conjugate of the modified perturbed operator then can be written as:*

$$\widetilde{\mathsf{DP}}_H^*[J](z, \tilde{z}) = g^*(L_1^\top H^{-\top}(z, \tilde{z})) + J^*(L_2^\top H^{-\top}(z, \tilde{z})), \tag{3.23}$$

for $z, \tilde{z} \in \mathcal{S}$, where $L = [L_1 \;\; L_2]$ are given in Eq. (3.18).

Proof. Using Lemma 3.3 and Definition 3.10, the modified perturbed DP operator can be expressed as:

$$\widetilde{\mathsf{DP}}_H[J](w, \tilde{w}) = ((L \rhd (g \oplus J)) \circ H)(w, \tilde{w}). \tag{3.24}$$

Applying Lemma 2.15, Lemma 2.16 and Lemma 2.18 yields the following for the conjugate:

$$\begin{aligned}
\widetilde{\mathsf{DP}}_H^*[J](z, \tilde{z}) &= ((L \rhd (g \oplus J)) \circ H)^*(z, \tilde{z}) \\
&= ((L \rhd (g \oplus J))^* \circ H^{-\top})(z, \tilde{z}) \\
&= \left[g^* \circ L_1^\top \circ H^{-\top} \oplus J^* \circ L_2^\top \circ H^{-\top}\right](z, \tilde{z}) \\
&= g^*(L_1^\top H^{-\top}(z, \tilde{z})) + J^*(L_2^\top H^{-\top}(z, \tilde{z})).
\end{aligned}$$ □

Choosing H appropriately is not trivial and could be investigated in the future.

Corollary 3.12.1. *Take H block triangular:*

$$H = \begin{bmatrix} I & 0 \\ C & D \end{bmatrix}, \qquad H^{-1} = \begin{bmatrix} I & 0 \\ -D^{-1}C & D^{-1} \end{bmatrix}, \tag{3.25}$$

with I the identity matrix of size $\mathbb{R}^{n \times n}$ and D invertible of size $\mathbb{R}^{n \times n}$. Then combining Equations (3.21) and (3.23) with the definition of H leads to:

$$\mathsf{DP}[J](x) = \widetilde{\mathsf{DP}}_H[J](x, -D^{-1}Cx), \tag{3.26}$$

$$\widetilde{\mathsf{DP}}_H^*[J](s, \tilde{s}) = g^*(s - (A + C)^\top D^{-\top}\tilde{s}, -B^\top D^{-\top}\tilde{s}) + J^*(D^{-\top}\tilde{s}). \tag{3.27}$$

Setting $C = -A$ and $D = I$ this further simplifies to

$$\mathsf{DP}[J](x) = \widetilde{\mathsf{DP}}_H[J](x, Ax) = \widetilde{\mathsf{DP}}_H^{**}[J](x, Ax), \tag{3.28}$$

$$\widetilde{\mathsf{DP}}_H^*[J](s, \tilde{s}) = g^*(s, -B^\top \tilde{s}) + J^*(\tilde{s}). \tag{3.29}$$

Figure 3.3 shows the steps of this approach. Notice that $g^*(s, -B^\top \tilde{s})$ is no longer evaluated at the sum of sets. Consequently, it is no longer necessary to evaluate points belonging to the Minkowski-sum of s and $\tilde{s}$.

3.4. Special case: cost function g separable

The results of the previous section become even nicer under these additional assumptions.

Assumption A.3. Extend Assumption A.2 by the running cost g being separable in terms of state (x) and action (u):

$$g(x, u) = g_x(x) + g_u(u), \tag{3.30}$$

where $g_x : \mathcal{X} \to]-\infty, +\infty]$ and $g_u : \mathcal{U} \to]-\infty, +\infty]$.

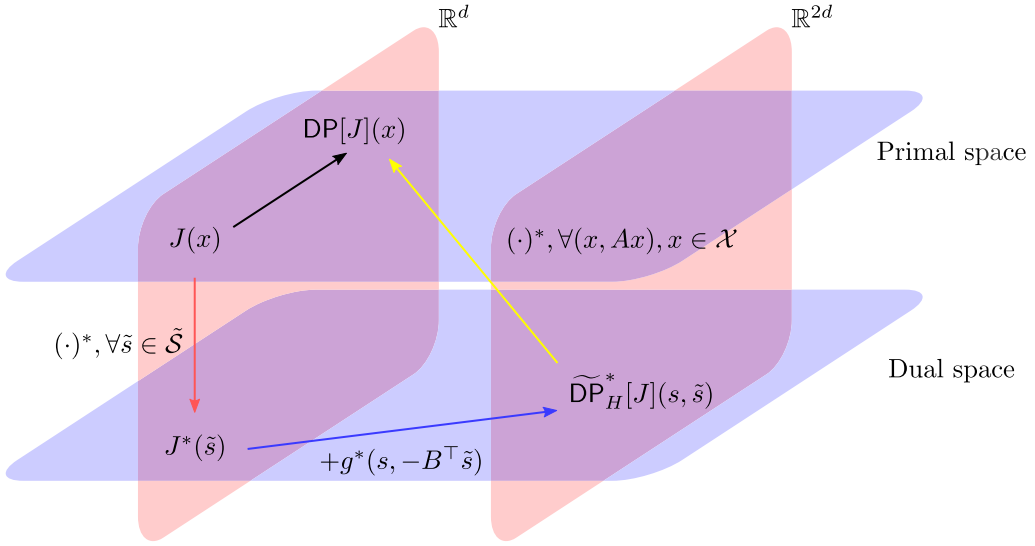


Figure 3.3: Summary of the effect of introducing invertible H to the framework. The overall scheme closely resembles that of Figure 3.4, however, going to and coming back from $\mathbb{R}^{2d}$ need slightly different operations. In turn, this difference allows us to eliminate computation of the Minkowski-sum.

In this case, we can express Equation (3.29) the following way.

Proposition 3.13. *Under Assumption $\mathcal{A}.3$ the conjugate of modified perturbed DP operator becomes:*

$$\widetilde{\mathsf{DP}}^*_H[J](s, \tilde{s}) = (j_s \oplus j_{\tilde{s}})(s, \tilde{s}), \tag{3.31}$$

where $j_s(s) = g_x^*(s)$ and $j_{\tilde{s}}(\tilde{s}) = g_u^*(-B^\top \tilde{s}) + J^*(\tilde{s})$. Further, Equation (3.28) becomes:

$$\mathsf{DP}[J](x) = g_x(x) + j_{\tilde{s}}^*(Ax). \tag{3.32}$$

Proof. Using separability, Lemma 2.18 can be used to express the conjugate of g:

$$\begin{aligned} \widetilde{\mathsf{DP}}^*_H[J](s, \tilde{s}) &= g^*(s, -B^\top \tilde{s}) + J^*(\tilde{s}) \\ &= g_x^*(s) + g_u^*(-B^\top \tilde{s}) + J^*(\tilde{s}) \\ &=: j_s(s) + j_{\tilde{s}}(\tilde{s}) \\ &= (j_s \oplus j_{\tilde{s}})(s, \tilde{s}). \end{aligned} \tag{3.33}$$

This completes the first claim. The second claim is obtained by taking the conjugate once more:

$$\begin{aligned} \mathsf{DP}[J](x) &= \widetilde{\mathsf{DP}}_H[J](x, Ax) \\ &= \left(\widetilde{\mathsf{DP}}^*_H[J]\right)^*(x, Ax) \\ &= j_s^*(x) + j_{\tilde{s}}^*(Ax) \\ &= g_x(x) + j_{\tilde{s}}^*(Ax), \end{aligned} \tag{3.34}$$

where we used Lemma 2.18. Note that under Assumption $\mathcal{A}.1$ – which is included in Assumption $\mathcal{A}.3$ – we have $j_s^* = (g_x^*)^* = g_x$. □

It is important to realize that *all* (intermediate) functions are "univariate", meaning that they either a function of x (resp. s) or $\tilde{x}$ (resp. $\tilde{s}$) but never of both. This is an enormous advantage from a computational point of view since none of the functions need storage for the Cartesian product of the sets $\mathcal{X}^2$ or $\mathcal{S}^2$. Moreover, all computations can be done in $\mathcal{X}$ (or $\mathcal{S}$). This advantage is indicated on Fig. 3.4 by staying in $\mathbb{R}^d$.

Furthermore, if $g_x = 0$ — and under other practical assumptions, such as nonnegative monotone A and a discount factor of $\beta = 1$ — we get back the results of [30, Lemma 4.][3]. Their method stays in the dual domain for computing $\mathsf{DP}^T[J]$ as $T \to \infty$. Application of subsequent DP operations accounts skipping the last addition in our case with g_x set to 0 (see Figure 3.4), moreover, Lemma 2.15 allows computation of $j_{\tilde{s}}^* \circ A = (j_{\tilde{s}} \circ A^{-\top})^*$ in the dual domain. Hence, there is no need to go back to the primal domain.

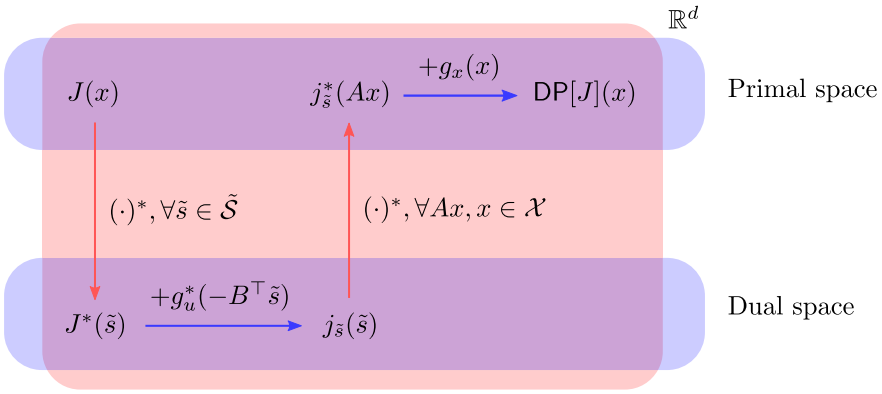


Figure 3.4: Summary of our method when $f(x,u) = Ax + Bu$ and $g(x,u)$ is separable in x and u. Note that we always stay in $\mathbb{R}^d$ and never go to $\mathbb{R}^{2d}$.

[3] In [30], concave functions and the "concave conjugate" is used as opposed to their convex counterparts here. By changing $J \to -J$ and $g \to -g$ we get appropriate concave functions that satisfy conditions in [30].

4

Implementation – Discrete spaces

So far there has been no attempt to make a distinction regarding continuous or discrete sets for the state and action space and their dual counterpart. However, we cannot easily simulate continuous sets on a computer therefore a numerical approach needs to be taken. (This is further supported by the fact that some functions do not have a closed form conjugate.) We investigate the shortcomings of the methods described in Chapter 3 from a computational point of view. First the (reoccurring) problems of discrete conjugation will be mentioned. Then, the algorithms will be presented along with their computational issues. These issues are addressed either by the introduction of new algorithms, or in the following section by approximation. There are also important considerations when choosing the dual variables s and $\tilde{s}$ in terms of approximation error that we briefly explore in this chapter. Last, the time and storage complexities of the algorithms will be discussed.

Instead of doing the calculations with $\mathcal{X}$, $\mathcal{U}$, $\mathcal{S}$ and $\tilde{\mathcal{S}}$ we take $\mathcal{X}_d$, $\mathcal{U}_d$, $\mathcal{S}_d$ and $\tilde{\mathcal{S}}_d$ which are finite representations of the continuous sets. Formally, $\mathcal{X}_d \subset \mathcal{X}$ and $|\mathcal{X}_d| = X < \infty$, and, similarly, $\mathcal{U}_d \subset \mathcal{U}$ and $|\mathcal{U}_d| = U < \infty$, $\mathcal{S}_d \subset \mathcal{S}$ and $|\mathcal{S}_d| = S < \infty$, $\tilde{\mathcal{S}}_d \subset \tilde{\mathcal{S}}$ and $|\tilde{\mathcal{S}}_d| = \mathcal{S} < \infty$.

4.1. Discrete conjugation

Let us start by saying that a discrete subset of $\mathbb{R}^n$ is not a Hilbert space, therefore Legendre–Fenchel transform is no longer well defined. However, the LFT can be naturally extended to the discrete case which becomes an advantages of discretization: discrete conjugation becomes a maximization – instead of supremum – meaning the maximum is always attained. This happens because given a finite set $\mathcal{X}_d \subset \mathcal{X}$ and a proper function $f : \mathcal{X} \to]-\infty, +\infty]$, the function defined as $h_d(x, s) = \langle x, s\rangle - f(x)$ will have finitely many values for any given $x \in \mathcal{X}_d$. Therefore, leading to the following definition.

Definition 4.1. The Discrete Legendre–Fenchel Transform (sometimes referred to as discrete LFT or DLT in literature) of a proper function $f : \mathcal{X}_d \to]-\infty, +\infty]$ and finite set $\mathcal{X}_d$ is given as:

$$f_d^*(s) = \sup_{x \in \mathcal{X}_d} \big(\langle x, s\rangle - f(x)\big) = \sup_{x \in \mathcal{X}_d} h_d(x, s) = \max_{x \in \mathcal{X}_d} \big(\langle x, s\rangle - f(x)\big), \quad \forall s \in \mathcal{S}, \tag{4.1}$$

where $h_d(x, s) = \langle x, s\rangle - f(x)$.

It is worth emphasizing that even though $\mathcal{X}_d$, and consequently $f_d(x_d)$, has finitely many elements, the dual set $\mathcal{S}$ and dual function $f_d^*(s)$ can have infinitely many elements. This property may be exploited to obtain an interpolation of f_d at points lying in $\overline{\mathcal{X}}_d = \mathcal{X} \setminus \mathcal{X}_d$, i.e. points *in between* of $\mathcal{X}_d$, by taking the biconjugate of f_d. Although care should be taken when choosing the dual space $\mathcal{S}$ in this case, as it will be explained later. Taking the biconjugate of f_d at all points of $\mathcal{X}$ will result in the convex hull defined by the points in $\mathcal{X}_d$. This is illustrated on Figure 4.1 with a simple example.

Besides making the problem more tangible from a computational point of view, discretization also brings complications with itself. For one, when the dual space is finite as well, i.e. $\mathcal{S}_d \subset \mathcal{S}$ and $|\mathcal{S}_d| < \infty$, the property $f^{**} = f$ may be lost. This is illustrated in the following example.

Example 4.2. Consider the continuous function in $\mathbb{R}$:

$$f(x) = \frac{3}{4}x^2 - \frac{3}{4}x - \frac{13}{16},$$

and its (continuous) conjugate

$$f^*(s) = \frac{1}{3}s^2 + \frac{1}{2}s + 1.$$

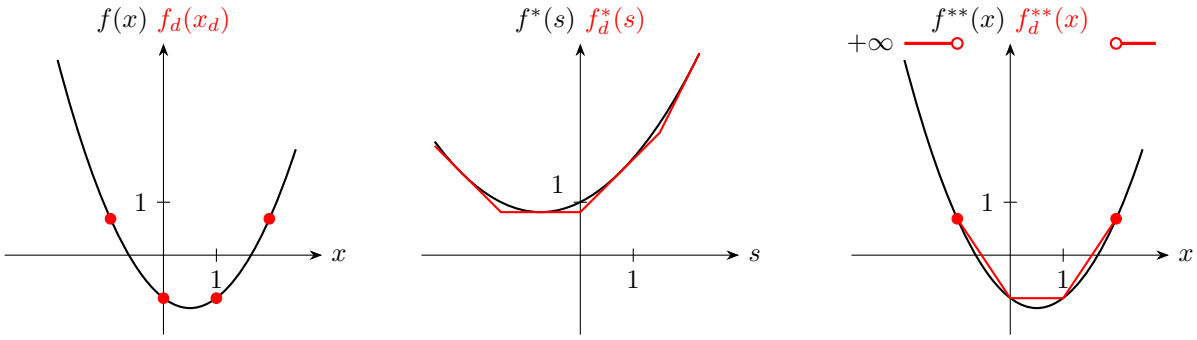


Figure 4.1: Example of discretization with $f(x) = \frac{3}{4}x^2 - \frac{3}{4}x - \frac{13}{16}$ and its conjugate $f^*(s) = \frac{1}{3}s^2 + \frac{1}{2}s + 1$. Here we take $\mathcal{X} = \mathbb{R}$, $\mathcal{X}_d = \{-1, 0, 1, 2\}$, and $\mathcal{S} = \mathbb{R}$.

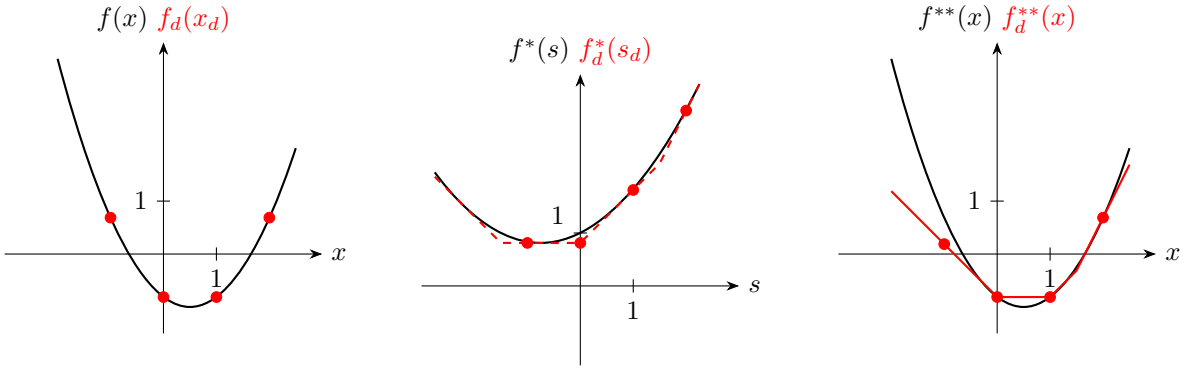


Figure 4.2: Same example as on Figure 4.1 but with $\mathcal{S}_d = \mathcal{X}_d = \{-1, 0, 1, 2\}$. Note that $f_d^{**}(x_d) \neq f_d(x_d)$ for all x_d. Moreover, f_d^{**} does not "kink" at points of $\mathcal{X}_d$, there is a change of slope of f_d^{**} at $x = 1.5$ in between $x = 1, 2 \in \mathcal{X}_d$.

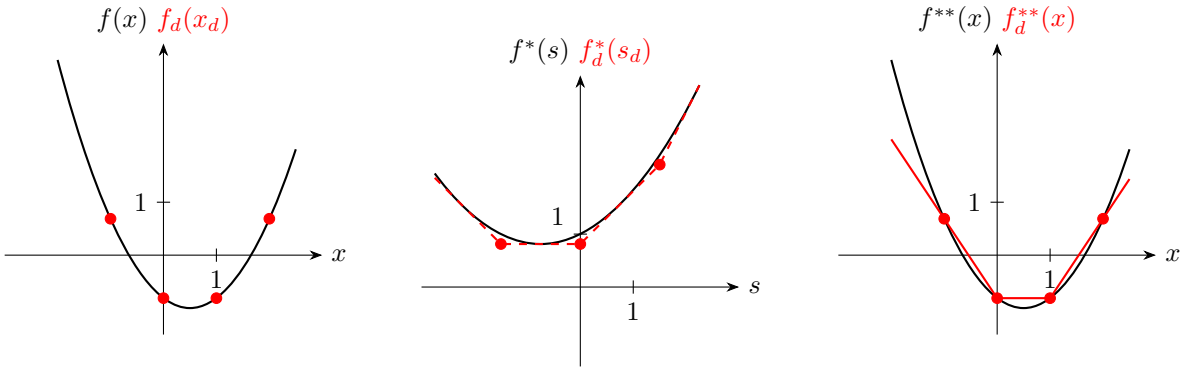


Figure 4.3: Same example as on Figure 4.1 but with $\mathcal{S}_d = \{-\frac{3}{2}, 0, \frac{3}{2}\}$, which are exactly the discrete slopes of $(x_d, f_d(x_d))$. Note that $f_d^{**}(x_d) = f_d(x_d)$ for all x_d. Moreover, f_d^{**} "kinks" only at points of $\mathcal{X}_d$.

Sample the continuous function on the discrete set $\mathcal{X}_d = \{-1, 0, 1, 2\}$ with continuous dual space, i.e. $\mathcal{S}_d = \mathbb{R}$ and apply (bi)conjugation (see Figure 4.1). Notice 3 things: (1) f_d^* is piecewise affine with slopes corresponding to elements of $\mathcal{X}_d$, (2) f_d^* is touching f^* from below, and (3) how the biconjugate

f_d^{**} can be evaluated for all x even though f_d was only defined at $\mathcal{X}_d$. Furthermore, the original samples are perfectly recovered, meaning $f_d^{**}(x_d) = f_d(x_d)$ for all x_d.

Now, take $\mathcal{S} \neq \mathbb{R}$ but some discrete subset of $\mathbb{R}$. Two of such cases are depicted in Figure 4.2 and 4.3. Depending on the choice of the dual space, perfect reconstruction – that is $f_d(x_d) = f_d^{**}(x_d)$ – is possible but not automatically present. Loosely speaking, the reason why this happens is that $\mathcal{S}_d$ does not "represent" f^* well enough, meaning that there are certain slopes of f_d^* to which there is no points in $f_d^*(s_d)$. See, for example on Figure 4.2 $f_d^*(s_d)$ has no point on the slope corresponding to -1, that is why $f_d^{**}(-1)$ fails to recover its original value.

This example illustrates the main disadvantage of discretizing the problem: the dual space is also discrete and biconjugate may not recover the original function. Since our methods derived so far are essentially some form of biconjugation, this disparity is an issue. To tackle this, either the elements of the dual space should be carefully chosen, or approximations have to be made and thus introducing errors.

4.1.1. Linear-time Legendre–Fenchel algorithm

One of the main building blocks of DP is conjugation. Fortunately, there exists an algorithm developed by Lucet [22] that is able to compute the discrete conjugate of any combination of input points (x), input function ($f_d(x)$) and output points (s). We refer to this algorithm as Linear-time Legendre–Fenchel algorithm (LLT). Since discrete conjugation forms such a fundamental cornerstone in our approach it is worth reiterating the algorithm here. In short, computing the conjugation consists of the following steps:

1. Factorization. In a higher dimensional case, i.e. when $\mathcal{X}_d \subset \mathbb{R}^n$ with $n > 1$, the conjugate can be written as [22]:

$$\begin{aligned} f_d^*(s) &= \max_{x \in \times_{i=1}^n \mathcal{X}_{d,i}} \big[\langle x, s \rangle - f(x) \big] \\ &= \max_{x_1 \in \mathcal{X}_{d,1}} \left[x_1 s_1 + \max_{x_2 \in \mathcal{X}_{d,2}} \left[x_2 s_2 + \cdots + \max_{x_n \in \mathcal{X}_{d,n}} [x_n s_n - f_d(x)] \cdots \right] \right]. \end{aligned}$$

   Notice that every maximization concerns a single element x_i of x. Furthermore, evaluating from innermost maximization outwards the resulting functions no longer depends on x_i, they are "maximized away". This formulation allows us to handle higher dimensional cases similarly to one-dimensional cases. However, it is important that the coordinates of x are independent along each dimension, thus forming a hyper-rectangular grid, otherwise factorization may not take place.

2. Compute discrete slopes c_i along a single dimension from consecutive points:

$$c_i = \frac{f_d(x_{i+1}) - f_d(x_i)}{x_{i+1} - x_i}.$$

3. Merge sets (i.e. find optimizers); find indices i such that:

$$c_i < s_j < c_{i+1}, \quad \forall s_j \in \mathcal{S}_d.$$

4. Compute conjugate by simple substitution:

$$f_d^*(s_j) = c_i(s_j - x_i) - f_d(x_i), \quad \forall s_j \in \mathcal{S}_d.$$

A more in-depth explanation can be found in [22]. For us it is important that the LLT algorithm has $O(N+M)$ time and storage complexity for N pairs of input $(x_i, f(x_i))$ and M output/query points s_j.

4.2. Algorithms and computational issues

Before jumping straight to the implementation, let us first start with identifying the underlying computational issues of the methods in Chapter 3 with the focus on discrete state and action sets.

4.2.1. General case

Recall the first algorithm presented in Section 3.2 for the general case to compute $\mathsf{DP}[J]$ (see also Figure 3.1):

$$\mathsf{DP}[J](x) = \widetilde{\mathsf{DP}}^{**}[J](x, 0), \tag{4.2}$$

$$\widetilde{\mathsf{DP}}^{*}[J](s, \tilde{s}) = g^*(s - A^\top \tilde{s}, -B^\top \tilde{s}) + J^*(\tilde{s}). \tag{4.3}$$

There is one main issue with computing single step of DP outlined here. It can be related to the introduction of the auxiliary variables $\tilde{x}$ and $\tilde{s}$ and can be seen on Figure 3.1.

First, Equation (4.3) provides a way to compute $\widetilde{\mathsf{DP}}^{*}[J]$ from J^*, however, computing this directly is computationally *very* expensive. Most notably, the term $g^*(s - A^\top \tilde{s}, -B^\top \tilde{s})$ causes problems as it needs to be evaluated at points defined by $s - A^\top \tilde{s}$. As mentioned earlier, this accounts to computing points that are part of the Minkowski-sum of the sets containing all s and $\tilde{s}$. If there are S and $\tilde{S}$ points in $\mathcal{S}_d$ and $\tilde{\mathcal{S}}_d$, respectively, then cardinality of $s - A^\top \tilde{s}$ is $O(S\tilde{S})$. This presents an addressing/indexing problem of computing g^* with LLT since its arguments are not independent.

Second, from a storage point of view, a problem of the same origin is that in the dual domain both s and $\tilde{s}$ are necessary for the computations while in the primal domain we only care about x and not about $\tilde{x}$. If $|\mathcal{S}_d| \approx |\tilde{\mathcal{S}}_d| \approx |\mathcal{X}_d|$ then evaluating and storing $\widetilde{\mathsf{DP}}^{*}[J](s, \tilde{s})$ would lead to a similar explosion of computation and storage.

The pseudo-code for implementing $\mathsf{DP}[J](x)$ for time-variant dynamics and cost function is given below (see Algorithm 1). The algorithm describes the most general case, however, it also has the poorest performance: since g_t^* needs to be (re)computed for each time instance, it is costly both in terms of computation and storage (Minkowski sum).

Remark 4.3. If the dynamics are time-invariant, Algorithm 1 can be improved by computing g^* outside of the `while` loop ("offline"). Most notably, computing the term $g^*(s - A^\top \tilde{s}, -B^\top \tilde{s})$ becomes a one-time cost if the dynamics and running cost are time-invariant. This happens when g, A and B are not changing with the steps of DPA, i.e. $f_t = f$ and $g_t = g$ for all $t < T$. In other words, after evaluating (and storing) the term $g^*(s - A^\top \tilde{s}, -B^\top \tilde{s})$, re-evaluation of it is no longer required for subsequent application of the DP operator. Depending on the context of the problem, this property may be exploited to compute many steps while saving immense amount of completion time. Not doing so, in general, only adds computational complexity.

Algorithm 1: Linear time-variant dynamics and time-variant cost function

Data: States $\mathcal{X}_d$, actions $\mathcal{U}_d$, running costs $g_t(x, u)$, dynamics $f_t(x, u)$, terminal cost $J_T(x)$, time horizon T
Result: $J_0(x) = \mathsf{DP}^T[J](x)$ for $x \in \mathcal{X}_d$
$t := T$
while $t > 0$ **do**
  Fix discrete sets $\mathcal{S}_d, \tilde{\mathcal{S}}_d$
  foreach $s \in \mathcal{S}_d, \tilde{s} \in \tilde{\mathcal{S}}_d$ **do**
    Compute $q(s, \tilde{s}) = s - A_t^\top \tilde{s}$
    Compute $v(\tilde{s}) = -B_t^\top \tilde{s}$
  end
  Use LLT to compute $g_t^*(q(s, \tilde{s}), v(\tilde{s}))$ for all $s, \tilde{s} \in \mathcal{S}_d$ from $g_t(x, u)$
  Use LLT to compute $J_t^*(\tilde{s})$ for $\tilde{s} \in \mathcal{S}_d$ from $J_t(x)$
  $\widetilde{\mathsf{DP}}^{*}[J_t](s, \tilde{s}) := g_t^*(q(s, \tilde{s}), v(\tilde{s})) + J_t^*(\tilde{s})$
  Use LLT to compute $\widetilde{\mathsf{DP}}^{**}[J_t](x, 0)$ for $x \in \mathcal{X}_d$ from $\widetilde{\mathsf{DP}}^{*}[J_t](s, \tilde{s})$
  $J_{t-1}(x) := \widetilde{\mathsf{DP}}^{**}[J_t](x, 0)$
  $t := t - 1$
end

4.2.2. Improved conjugate expression

To combat this issue mentioned above, the modified perturbed operator, $\widetilde{\mathsf{DP}}_H$, was introduced in Section 3.3. With it, the following method was derived:

$$\mathsf{DP}[J](x) = \widetilde{\mathsf{DP}}_H^{**}[J](x, Ax), \tag{4.4}$$

$$\widetilde{\mathsf{DP}}_H^{*}[J](s, \tilde{s}) = g^*(s, -B^\top \tilde{s}) + J^*(\tilde{s}). \tag{4.5}$$

The pseudo-code is given in Algorithm 2 for the time-invariant case. Similarly to before, the algorithm can be extended to time-varying cases by placing everything inside the `while` loop. The Minkowski sum vanished with a particular choice of H, leading to evaluation of $g^*(s, -B^\top \tilde{s})$ and $\widetilde{\mathsf{DP}}_H^{**}[J](x, Ax)$. Therefore, the indexing issue is no longer present. This, in turn, forms a problem of evaluating the composite of conjugation and linear transformation, and will be explored next.

Algorithm 2: Linear time-invariant dynamics and time-invariant cost function

Data: States $\mathcal{X}_d$, actions $\mathcal{U}_d$, running cost $g(x, u)$, dynamics $f(x, u)$, terminal cost $J_T(x)$, time horizon T
Result: $J_0(x) = \mathsf{DP}^T[J](x)$ for all $x \in \mathcal{X}_d$
Fix discrete sets $\mathcal{S}_d, \tilde{\mathcal{S}}_d$
foreach $\tilde{s} \in \tilde{\mathcal{S}}_d$ **do**
  Compute $v(\tilde{s}) = -B^\top \tilde{s}$
end
Use LLT to compute $g^*(s, v(\tilde{s}))$ for $s, \tilde{s} \in \mathcal{S}_d$ from $g(x, u)$
$t := T$
while $t > 0$ **do**
  Use LLT to compute $J_t^*(\tilde{s})$ for $\tilde{s} \in \mathcal{S}_d$ from $J_t(x)$
  $\widetilde{\mathsf{DP}}^{*}[J_t](s, \tilde{s}) := g^*(s, v(\tilde{s})) + J_t^*(\tilde{s})$
  Use LLT to compute $\widetilde{\mathsf{DP}}^{**}[J_t](x, Ax)$ for all $x \in \mathcal{X}_d$ from $\widetilde{\mathsf{DP}}^{*}[J_t](s, \tilde{s})$
  $J_{t-1}(x) := \widetilde{\mathsf{DP}}^{**}[J_t](x, Ax)$
  $t := t - 1$
end

While the algorithm described above provides a powerful tool for evaluating $\mathsf{DP}[J]$, it comes with a few computational caveats. Namely, it is difficult to compute $g^*(s, -B^\top \tilde{s})$ and $\mathsf{DP}_H^{**}[J](x, Ax)$ with LLT in a multidimensional setting, if A or B are noninvertible. In that case Lemma 2.15 cannot be used to speed up the process. Further, the LLT algorithm works best when the output points lie on a grid, or rather, more precisely, when the output points are independent along each dimension [22].

Consider the general setting, for a proper function $h : \mathcal{X} \to]-\infty, +\infty]$ and matrix M, the composite $h^*(Mx)$ needs to be computed. If the elements of M are not (small) integers or rationals with small denominators, it is difficult, if not impossible, to construct a set of independent coordinates whose cardinality is less than the cardinality of all input vectors combined. To illustrate this, take the following example.

Example 4.4. Let:

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad x \in \mathcal{X}_d = \{-1, 0, 1\}^2, \quad |\mathcal{X}_d| = 9.$$

Then, it is easy to come up with:

$$y \in \mathcal{Y}_d = \{-3, -2, -1, 0, 1, 2, 3\} \times \{-7, -4, -3, -1, 0, 1, 3, 4, 7\}, \quad |\mathcal{Y}_d| = 7 \times 9 = 63,$$

that covers *all* possible values of Mx. The conjugate of $h^*(Mx)$ for all $x \in \mathcal{X}_d$ can be thus computed with $h^*(y)$ for all $y \in \mathcal{Y}_d$ with LLT, however, each point in $\mathcal{X}_d$ is computed $|\mathcal{Y}_d|/|\mathcal{X}_d| = 7$ times and those computations are wasted! Ideally, $|\mathcal{Y}_d| \approx |\mathcal{X}_d|$ is wanted. The reason why LLT fails at computing $h^*(Mx)$ is that factorization can no longer be applied, hence the n-dimensional minimization problem does not decouple into n 1-dimensional minimizations.

4.2.3. Separable cost function g – Fast Dynamic Programming

Undoubtedly, the greatest advantage of separability is that all functions throughout the algorithm are "univariate", meaning they are only a function of x or $\tilde{s}$ but never of $(x, \tilde{x})$ or $(s, \tilde{s})$. This formulation alleviates the problem of extending the (dual) space to $\mathbb{R}^{2n}$, which is illustrated on Figure 3.4. Staying in $\mathbb{R}^n$ at all times is extremely helpful from a computational and storage point of view.

If the running cost is separable in state and action, i.e. $g(x,u) = g_x(x) + g_u(u)$, the framework described in Section 3.4 can be used which exploits the structure of underlying problem:

$$\mathsf{DP}[J](x) = g_x(x) + j^*_{\tilde{s}}(Ax), \tag{4.6}$$

$$j_{\tilde{s}}(\tilde{s}) = g^*_u(-B^\top \tilde{s}) + J^*(\tilde{s}). \tag{4.7}$$

The corresponding algorithm is outlined in Algorithm 3. Note that the problem of computing the composite $h^*(Mx)$ is still present here in $g^*(-B^\top \tilde{s})$ and $j^*_{\tilde{s}}(Ax)$. In the next section possible approaches will be discussed to compute the composite.

Algorithm 3: Linear time-invariant dynamics & separable time-invariant cost function (Fast Dynamic Programming)

Data: States $\mathcal{X}_d$, actions $\mathcal{U}_d$, running cost $g(x,u)$, dynamics $f(x,u)$, terminal cost $J_T(x)$, time horizon T
Result: $J_0(x) = \mathsf{DP}^T[J](x)$ for all $x \in \mathcal{X}_d$
Fix discrete set $\tilde{\mathcal{S}}_d$
foreach $\tilde{s} \in \tilde{\mathcal{S}}_d$ **do**
  Compute $v(\tilde{s}) = -B^\top \tilde{s}$
end
Use LLT to compute $g^*_u(v(\tilde{s}))$ for all $\tilde{s} \in \tilde{\mathcal{S}}_d$ from $g_u(u)$
$t := T$
while $t > 0$ **do**
  Use LLT to compute $J^*_t(\tilde{s})$ for all $\tilde{s} \in \tilde{\mathcal{S}}_d$ from $J_t(x)$
  $j_{\tilde{s}}(\tilde{s}) := g^*_u(v(\tilde{s})) + J^*_t(\tilde{s})$
  Use LLT to compute $j^*_{\tilde{s}}(Ax)$ for all $x \in \mathcal{X}_d$ from $j_{\tilde{s}}(\tilde{s})$
  $J_{t-1}(x) := g_x(x) + j^*_{\tilde{s}}(Ax)$
  $t := t - 1$
end

4.3. Error analysis of implementation

The question we will answer in this section is how the results of brute-force and fast DP compare to each other.

4.3.1. Discrete Dynamic Programming

It makes little sense to compare the results of Fast DP applied to a discretization of some continuous function. The discrete nature of Fast DP prohibits finding the same optimizer as the continuous.

Example 4.5. Take discrete sets $\mathcal{X}_d = \mathcal{U}_d = \{-1, 0, 1\}$, and their continuous counterparts $\mathcal{X}_c = \mathcal{U}_c = [-1, 1]$, as well as these quadratic cost functions: $g(x,u) = x^2 + u^2$ and $J(x) = x^2$ and dynamics $f(x,u) = x + u$. For the continuous case we have $\mathsf{DP}[J_c] = \frac{3}{2}x^2$ with $u^{opt} = -\frac{1}{2}x$, while in the discrete case $\mathsf{DP}[J_d]$ takes values $2, 0, 2$ with $u^{opt} = \{0, 1\}, \{0\}, \{-1, 0\}$ for $x = -1, 0, 1$ respectively.

Therefore the discrete DP operator needs to be defined.

Definition 4.6. The discrete DP operator or brute-force minimization, DP_d, is defined as:

$$\mathsf{DP}_d[J](x) = \min_{u \in \mathcal{U}_d} \{g(x,u) + J(f(x,u))\}, \qquad \forall x \in \mathcal{X}_d, \tag{4.8}$$

with given running cost $g(x,u) : \mathcal{X}_d \times \mathcal{U}_d \to \mathbb{R}$ and given dynamics $f(x,u) : \mathcal{X}_d \times \mathcal{U}_d \to \mathcal{X}_d$.

Perturbed versions of DP_d can be obtained similarly.

4.3.2. Computational problems with evaluating $h^*(Mx)$

Notice that all algorithms use LLT extensively and, perhaps, in a multidimensional setting. Since we have some freedom of choice of the dual parameters, namely $\mathcal{S}_d$, we may introduce errors if not the ideal $\mathcal{S}_d$ is chosen, as also illustrated on Figures 4.1 through 4.3. Given that the LLT gives only a numerical approximation of f^*, the choice of $\mathcal{S}_d$ is not obvious.

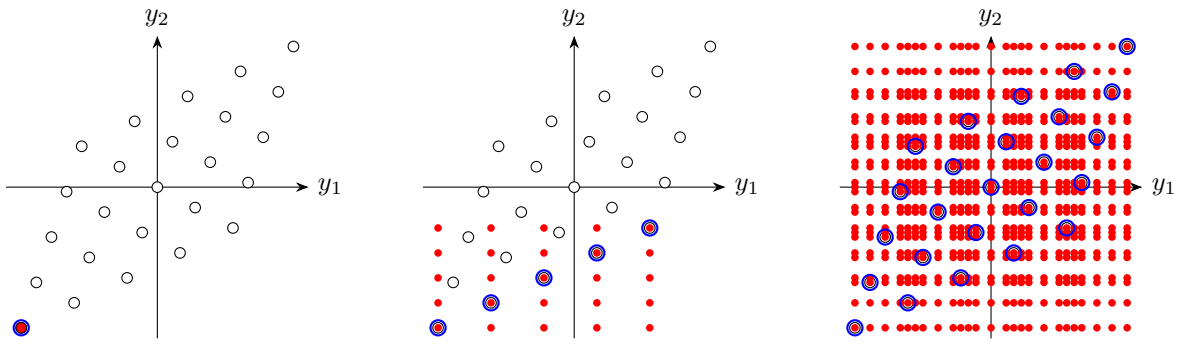
Previously, we deliberately left out the details of computing the composite of conjugation and linear transformation, e.g. $g^*(-B^\top \tilde{s})$. This step is non-trivial in the discrete case, and there are two fundamentally different options:

1. **Error-free:** One computes the composite without introducing errors. Since the output points (e.g. Ax) may not necessarily have independent coordinates along each dimension, the computation may be burdensome. There are several ways to tackle this issue – while ensuring error-free operation – but they all suffer immense penalty, either time, storage or both. Here we list a few of these options:

   - Direct computation of the conjugate at every possible Ax point: this can be achieved by running through every x, then calculating Ax, and **computing the conjugate at a single point** with LLT, see Figure 4.4a. There are X points and each LLT takes $O(S+1)$ operations. Hence we have $O(X(S+1)) = O(XS)$ complexity.
   - Repeatedly compute the conjugate for **points lying on a line**, see Figure 4.4b. Since the input set is a boxed set and the output space is its linear transformation, points lying on a line in the input space will also be on a line in the output space. Each query takes $O(S+X)$ time and this should be repeated $\sqrt[n]{X}^{n-1}$ times to cover all points in the output space – assuming there are equal number of points along each dimension. Hence we have $O(\sqrt[n]{X}^{n-1}(S+X)) \approx O(XS+X^2)$ complexity.
   - Compute conjugate over **all possible combinations of coordinates** in the output space. This approach has time complexity of $O(S+X^2)$ because the output space has X coordinates (compared to the "original" $\sqrt[n]{X}$ coordinates), see Figure 4.4c.

   As the number of dimensions (n) grow, these options become worse and worse.

2. **With errors:** Set $\mathcal{S}_d$ a grid, such that $|\mathcal{S}_d| \approx |\mathcal{X}_d|$ and compute the conjugate on those points. Use (linear) interpolation to evaluate points lying in between elements of $\mathcal{S}_d$.



(a) Single point. (b) Points on line. (c) All possible coordinate pairs.

Figure 4.4: Possible realizations of the query $f^*(Ax)$ in $\mathbb{R}^2$. White represents all $y = Ax$, red are points at which the conjugate is computed in each iteration, blue shows the points of which we get information in each iteration.

Under certain (strict) conditions, perfect reconstruction of the algorithm may be guaranteed, that is, $\mathsf{DP}_d[J](x)$ can be computed directly by evaluating its perturbed versions.

Proposition 4.7. *By ensuring*

1. *the discrete state/control sets are control invariant ($\forall x_d \in \mathcal{X}_d, \exists u_d \in \mathcal{U}_d$ such that $Ax_d + Bu_d \in \mathcal{X}_d$), and*

2. *the discrete dual spaces $\mathcal{S}_d$ and $\tilde{\mathcal{S}}_d$ are such that $\exists s \in \mathcal{S}_d, \exists \tilde{s} \in \tilde{\mathcal{S}}_d$ such that $\forall u \in \mathcal{U}_d$, the following holds:*

$$s - A^\top \tilde{s} \in \partial_x g(x,u), \quad \textit{and} \quad \tilde{s} \in \partial J(Ax+Bu),$$

we have perfect reconstruction, i.e. $\mathsf{DP}_d[J](x) = \widetilde{\mathsf{DP}}_d^{**}[J](x,0)$.

Proof. In the continuous case, we established $\mathsf{DP}[J](x) = \widetilde{\mathsf{DP}}[J](x,0) = \widetilde{\mathsf{DP}}^{**}[J](x,0)$ from Equation 3.2 and Proposition 3.5. However, due to the discrete nature of the state and action sets, equality may not always hold. Hence, we can no longer use this to prove equality, but we can express the the discrete version of $\widetilde{\mathsf{DP}}_d[J]$. Thus:

$$\begin{aligned}
\widetilde{\mathsf{DP}}_d^{**}[J](x,0) &= \max_{\substack{s\in\mathcal{S}_d\\ s\in\tilde{\mathcal{S}}_d}} \left\{ \left\langle \begin{bmatrix} x \\ 0 \end{bmatrix} \,\middle|\, \begin{bmatrix} s \\ \tilde{s} \end{bmatrix} \right\rangle - \widetilde{\mathsf{DP}}_d^{*}[J](s,\tilde{s}) \right\} \\
&= \max_{\substack{s\in\mathcal{S}_d\\ s\in\tilde{\mathcal{S}}_d}} \left\{ \left\langle \begin{bmatrix} x \\ 0 \end{bmatrix} \,\middle|\, \begin{bmatrix} s \\ \tilde{s} \end{bmatrix} \right\rangle - g^*(s - A^\top\tilde{s}, -B^\top\tilde{s}) + J^*(\tilde{s}) \right\} \\
&= \max_{\substack{s\in\mathcal{S}_d\\ s\in\tilde{\mathcal{S}}_d}} \left\{ \left\langle \begin{bmatrix} x \\ 0 \end{bmatrix} \,\middle|\, \begin{bmatrix} s \\ \tilde{s} \end{bmatrix} \right\rangle - \max_{\substack{w_1\in\mathcal{X}_d\\ u\in\mathcal{U}_d}} \left\{ \left\langle \begin{bmatrix} s - A^\top\tilde{s} \\ -B^\top\tilde{s} \end{bmatrix} \,\middle|\, \begin{bmatrix} w_1 \\ u \end{bmatrix} \right\rangle - g(w_1,u) \right\} - \right. \\
&\qquad \left. - \max_{w_2\in\mathcal{X}_d} \left\{ \langle \tilde{s} \mid w_2 \rangle - J(w_2) \right\} \right\} \\
&= \max_{\substack{s\in\mathcal{S}_d\\ s\in\tilde{\mathcal{S}}_d}} \min_{\substack{w_1,w_2\in\mathcal{X}_d\\ u\in\mathcal{U}_d}} \left\{ \langle x \mid s \rangle - \left\langle w_1 \mid s - A^\top\tilde{s} \right\rangle - \left\langle u \mid -B^\top\tilde{s} \right\rangle - \langle \tilde{s} \mid w_2 \rangle + g(w_1,u) + J(w_2) \right\} \\
&= \max_{\substack{s\in\mathcal{S}_d\\ s\in\tilde{\mathcal{S}}_d}} \min_{\substack{w_1,w_2\in\mathcal{X}_d\\ u\in\mathcal{U}_d}} \left\{ \langle x - w_1 \mid s \rangle - \langle \tilde{s} \mid Aw_1 + Bu - w_2 \rangle + g(w_1,u) + J(w_2) \right\}. \qquad (4.9)
\end{aligned}$$

Swapping the min and max operators we obtain:

$$\widetilde{\mathsf{DP}}_d^{**}[J](x,0) \leq \min_{\substack{w_1,w_2\in\mathcal{X}_d\\ u\in\mathcal{U}_d}} \max_{\substack{s\in\mathcal{S}_d\\ s\in\tilde{\mathcal{S}}_d}} \left\{ \langle x - w_1 \mid s \rangle - \langle \tilde{s} \mid Aw_1 + Bu - w_2 \rangle + g(w_1,u) + J(w_2) \right\}. \qquad (4.10)$$

By setting $w_1 = x$ and $w_2 = Ax + Bu$ we have:

$$\widetilde{\mathsf{DP}}_d^{**}[J](x,0) \leq \min_{u\in\mathcal{U}_d} \{ g(x,u) + J(Ax+Bu) \} = \mathsf{DP}_d[J](x), \qquad \forall x \in \mathcal{X}_d. \qquad (4.11)$$

Since $w_2 \in \mathcal{X}_d$ we need to have $\forall x \in \mathcal{X}_d, \exists u \in \mathcal{U}_d$ such that $Ax + Bu \in \mathcal{X}_d$, which is exactly our first requirement. In other words, this means that the discrete sets have to be control invariant. To finish the proof, we need to show $\widetilde{\mathsf{DP}}_d[J]^{**} \geq \mathsf{DP}_d[J]$:

$$\max_{\substack{s\in\mathcal{S}_d\\ \tilde{s}\in\tilde{\mathcal{S}}_d}} \min_{\substack{w_1,w_2\in\mathcal{X}_d\\ u\in\mathcal{U}_d}} \left\{ \langle x - w_1 \mid s \rangle - \langle \tilde{s} \mid Az_1 + Bu - z_2 \rangle + g(w_1,u) + J(w_2) \right\} \geq \min_{u\in\mathcal{U}_d} \{ g(x,u) + J(Ax+Bu) \}, \qquad (4.12)$$

which holds if there exists $s \in \mathcal{S}, \tilde{s} \in \tilde{\mathcal{S}}_d$ such that for all $w_1, w_2 \in \mathcal{X}_d$:

$$\min_{u\in\mathcal{U}_d} \left\{ \langle x - w_1 \mid s \rangle - \langle \tilde{s} \mid Aw_1 + Bu - w_2 \rangle + g(w_1,u) + J(w_2) \right\} \geq \min_{u\in\mathcal{U}_d} \{ g(x,u) + J(Ax+Bu) \}. \qquad (4.13)$$

This, in turn, can be guaranteed if for each $u \in \mathcal{U}_d$ there exists $s \in \mathcal{S}, \tilde{s} \in \tilde{\mathcal{S}}_d$ such that for all $w_1, w_2 \in \mathcal{X}_d$:

$$\begin{aligned}
\langle x - w_1 \mid s \rangle - \langle \tilde{s} \mid Aw_1 + Bu - w_2 \rangle + g(w_1,u) + J(w_2) &\geq g(x,u) + J(Ax+Bu) \\
\langle x - w_1 \mid s \rangle - \langle \tilde{s} \mid Aw_1 + Bu - w_2 + Ax - Ax \rangle + g(w_1,u) + J(w_2) &\geq g(x,u) + J(Ax+Bu) \\
\left\langle x - w_1 \mid s - A^\top\tilde{s} \right\rangle + g(w_1,u) - g(x,u) + & \\
+ \langle \tilde{s} \mid Ax + Bu - w_2 \rangle + J(w_2) - J(Ax+Bu) &\geq 0. \qquad (4.14)
\end{aligned}$$

Split the inequality in two:

$$\langle x - w_1 \mid s - A^\top \tilde{s} \rangle + g(w_1, u) - g(x, u) \geq 0, \tag{4.15}$$
$$\langle \tilde{s} \mid Ax + Bu - w_2 \rangle + J(w_2) - J(Ax + Bu) \geq 0, \tag{4.16}$$

and rearrange:

$$g(w_1, u) \geq \langle w_1 - x \mid s - A^\top \tilde{s} \rangle + g(x, u), \tag{4.17}$$
$$J(w_2) \geq \langle \tilde{s} \mid w_2 - (Ax + Bu) \rangle + J(Ax + Bu). \tag{4.18}$$

Per definition of the subgradient, Equations (4.17) and (4.18) mean that if $s - A^\top \tilde{s} \in \partial_x g(x, u)$, and $\tilde{s} \in \partial J(Ax + Bu)$ for all pairs $(x, u) \in \mathcal{X}_d \times \mathcal{U}_d$ then $\widetilde{\mathsf{DP}}_d^{**}[J] \geq \mathsf{DP}_d[J]$. Together with Equation (4.11) this concludes the proof. □

Remark 4.8. The equivalence $\widetilde{\mathsf{DP}}_{H,d}^{**} = \mathsf{DP}_d$ as well as in the case when separable cost function g is used, can be derived following the same steps, arriving to the same conditions as in Proposition 4.7.

Remark 4.9. The first condition in Proposition 4.7 is rather difficult to ensure for given arbitrary dynamics A, B and given arbitrary sets $\mathcal{X}_d$,$\mathcal{U}_d$. On the other hand, if the sets $\mathcal{X}_d$,$\mathcal{U}_d$ can be chosen, then given a set $\mathcal{X}_d$ with $|\mathcal{X}_d| = X$, the action set $\mathcal{U}_d$ can be constructed such that the first condition hold for all $x \in \mathcal{X}_d$. For example, by taking the pseudo-inverse of B:

$$u = B^\dagger (x_k - A x_j),$$

for all $j \in \{1, 2, \ldots, X\}$ and – not necessarily different – $k \in \{1, 2, \ldots, X\}$. In best case this gives $|\mathcal{U}_d| = X$ meaning for each x_j there is exactly one u_i that satisfies the first condition.

To ensure the second condition in Proposition 4.7 for $|\mathcal{X}_d| = X$ and $|\mathcal{U}_d| = U$ we need $|\mathcal{S}_d| \geq XU$ and $|\tilde{\mathcal{S}}_d| \geq XU$. This is because $|Ax + Bu| = XU$ for all pairs (x, u) and thus $|\tilde{\mathcal{S}}_d| \geq XU$. Then, for each $\tilde{s}(x, u) \in \tilde{\mathcal{S}}_d$ an s can be found by, for example,

$$s(x, u) = \partial_x g(x, u) + A^\top \tilde{s}(x, u).$$

In practice, one could compute the set of (discrete) slopes of g and J_T based on the data points available and use those slopes for $\mathcal{S}_d$. However, when multiple steps of DP are required, this would imply the (costly) computation of, for instance, $g_u^*(-B^\top \tilde{s}_t)$ for all $\tilde{s}_t \in \tilde{\mathcal{S}}_{d,t}$. Depending on the particular function g_u, this may be not be feasible to do in each step. Therefore, to save on computational time, a fixed $\mathcal{S}_{d,t} = \mathcal{S}_d$ would be more advantageous. Care should be taken when choosing a fixed $\mathcal{S}_d$, because as $\mathsf{DP}^t[J]$ evolves in a rather complex way so does its slopes, and thus when $s_t \notin \partial\, \mathsf{DP}^t[J]$ a significant error is introduced.

Because it is inefficient to have the perfect reconstruction from a computational and storage point of view for large cardinalities, we have to resort to approximation. Preliminary results suggest the error is proportional to the step-sizes of the primal and dual grids, as well as the Lipschitz-constants of the used cost functions. It is left for future work to put rigorous bounds on the error generated this way.

4.4. Time complexity

For brevity, we will denote the cardinalities as follows: for the states $|\mathcal{X}_d| = X$ in the primal domain, $|\mathcal{S}_d| = S$ and $|\tilde{\mathcal{S}}_d| = \tilde{S}$ in the dual domain; for actions $|\mathcal{U}_d| = U$ in primal domain.

We also assume the coordinates of the elements in these discrete sets are equidistant along each dimension. This makes the cost of interpolation $O(1)$ for steps that include the composite of conjugation and linear transformation. If the coordinates are not equidistant, linear interpolation has $O(\log N)$ time assuming binary tree was used to store the data. Technically this alters the derived complexities, however, in practice $O(\log N)$ grows slowly.

The brute-force minimization has $O(TXU)$ time complexity: at each step of the DP (T) the objective function needs to be optimized for all x. Fixing x leads to $O(U)$ computations to find the optimum, which needs to computed for all x, hence a single step costs $O(XU)$. As for storage, strictly speaking, only $O(X)$ storage is necessary. If one finds the optimizer "in-place" always keeping track of the current optimum, u^{opt}, for a fixed x and update the contents of $J(x)$ based on the value of the optimum. However, this may lead to slower execution times. With modern hardware it is often more advantageous to compute and store $g(x, u)$ for all (x, u) and in one go, and then access it each step of DP. Doing so leads to $O(XU)$ storage complexity, but (possibly) faster execution time.

Proposition 4.10. *The time complexity of Algorithm 1 is $O(T(XU + S\tilde{S}))$ and it needs $O(S\tilde{S} + X)$ storage, if the system is time-invariant it needs $O(XU + T(X + S\tilde{S}))$ computations and the same $O(S\tilde{S} + X)$ storage.*

Proof. Computing points $q(s, \tilde{s})$ accounts to $O(S+\tilde{S})$ computations and storing q needs $O(S\tilde{S})$ memory. Computing $g^*(s - A^\top \tilde{s}, -B^\top \tilde{s})$ with LLT has time complexity of $O(XU + S\tilde{S})$ if error is introduced, and $(O(XU + XU \times XU) = O(X^2U^2)$ without errors (see Remark 4.9). Storing g^* needs $O(S\tilde{S})$ space in memory. These are one-time costs in the time-invariant case. In each further steps of DP one has to compute two LLT's and an addition. The first LLT has complexity $O(\tilde{S}+X)$, the second is $O(S\tilde{S}+X)$. Addition in this step is $O(S\tilde{S})$. Storing intermediate results, J_t requires $O(X)$ memory. All in all, the algorithm has

$$O(T(S + \tilde{S} + XU + S\tilde{S} + \tilde{S} + X + S\tilde{S} + X + S\tilde{S}) = O(T(XU + S\tilde{S}))$$

computations and needs

$$O(S\tilde{S} + S\tilde{S} + X) = O(S\tilde{S} + X)$$

storage space in the time-varying case. In the time-invariant case the algorithm has

$$O(S + \tilde{S} + XU + S\tilde{S} + T(\tilde{S} + X + S\tilde{S} + X + S\tilde{S}) = O(XU + T(X + S\tilde{S}))$$

computations and needs

$$O(S\tilde{S} + S\tilde{S} + X) = O(S\tilde{S} + X)$$

storage space. □

Corollary 4.10.1. *Algorithm 2 has time complexity $O(XU + T(X + S\tilde{S}))$ and storage complexity $O(\tilde{S}S + X)$.*

Proof. The `while` loop of Algorithm 2 is essentially the same as the `while` loop of Algorithm 1 in the time-invariant case. Before this loop, only g^* is computed, which has $O(\tilde{S}+XU+S\tilde{S})$ time complexity, and storing it requires $O(S\tilde{S})$ space. All things considered, Algorithm 2 has

$$O(\tilde{S} + XU + S\tilde{S} + T(X + S\tilde{S})) = O(XU + T(X + S\tilde{S}))$$

time complexity, and needs

$$O(S\tilde{S} + S\tilde{S} + X) = O(S\tilde{S} + X)$$

memory. □

Comparing time complexities $O(XU + T(X + S\tilde{S}))$ of Algorithm 1 and $O(T(XU))$ of brute-force, there is essentially no computational gain of using Algorithm 1 for the time-variant case. Only if the system is time-invariant, T is large (so that one-time costs vanish), and if $X + S\tilde{S} < XU$ then there is a gain. Referring back to Remark 4.9 it is difficult to choose $\mathcal{S}_d$ and $\tilde{\mathcal{S}}_d$ such that brute force is beaten in terms of accuracy and speed, and therefore Algorithm 1 is not a preferred method of computing DP.

We now analyse the case with separable cost function.

Proposition 4.11. *The time complexity of Fast Dynamic Programming (Algorithm 3) is $O(U+T(X+\tilde{S}))$ and it needs $O(\tilde{S} + X)$ storage.*

Proof. Computing points $v(\tilde{s})$ accounts to $O(\tilde{S})$ computations and storing v needs $O(\tilde{S})$ memory. g_u^* can be computed with $O(U + \tilde{S})$ operations and storing the result amounts to $O(\tilde{S})$ memory. This is a one-time cost. In the iteration, we need to compute the conjugate of two functions, these require $O(X + \tilde{S})$ and $O(\tilde{S} + X)$ operations respectively. Addition costs $O(\tilde{S})$. The immediate results need $O(\tilde{S})$ and $O(X)$ memory. Altogether, the algorithm has

$$O(\tilde{S} + U + \tilde{S} + T(X + \tilde{S} + \tilde{S} + X)) = O(U + T(X + \tilde{S}))$$

operations and requires

$$O(\tilde{S} + \tilde{S} + X) = O(\tilde{S} + X)$$

memory. □

Again, comparing the time complexities of Fast Dynamic Programming's $O(U + T(X + \tilde{S}))$ and naive $O(T(XU))$ we see an enormous advantage. Simply put, if we construct $\tilde{\mathcal{S}}_d$ such that $\tilde{S} < XU$, which is generally easy to achieve, for example, by taking $\tilde{S} \approx X$, FDP greatly outperforms brute-force minimization.

5

Numerical examples

Previously, we claimed superior performance of Fast DP compared to brute-force minimization. To support our claims three examples will be discussed. The first is a more traditional example in $\mathbb{R}$ showing input constrained system with quadratic costs under perfect reconstruction. The second showcases approximating solutions of a system with non-quadratic cost functions in $\mathbb{R}$. The last example demonstrates the scalability of our method through a system with quadratic costs in $\mathbb{R}^3$.

In all examples the results will be compared to results obtained by brute-force minimization of $\mathsf{DP}_d[J]$. Throughout this chapter direct minimization will be referred to as brute-force (BF). Because of their poor theoretical performance, Algorithms 1 and 2 will not be considered. Thus, only Algorithm 3, which will be referred to as Fast DP (FDP), will be matched against BF. For simplicity, their respective outputs will be denoted with $J_{t,BF} = \mathsf{DP}_d^t[J_T]$ and, similarly for $J_{t,FDP}$. Furthermore, the algorithms were implemented in MATLAB® and were run on a personal computer with Intel® i5-750 2.66 GHz processor and 6 GB RAM.

Note that setting the sets to be between some lower and upper limits, e.g. $\mathcal{X} \in [a, b]^n$ for some $a < b \in \mathbb{R}$, amounts to solving the continuous problem with additional constraints restricting the states, actions or both. Knowing this is useful for implementing state/action constraints of the same kind. Furthermore, this restriction does not necessarily take away the properties of Hilbert spaces. Recall from Section 2.1.1 that for a proper function $f : \mathcal{X} \to]-\infty, +\infty]$ one could define a function $f_{\mathcal{X}} : \mathcal{H} \to \mathbb{R}$ as:

$$f_{\mathcal{X}}(x) = \begin{cases} f(x) & \text{if } x \in \mathcal{X}, \\ +\infty & \text{otherwise.} \end{cases}$$

This extension does not alter convexity, properness or (lower) semicontinuity of f as long as $\mathcal{X}$ is closed and convex, hence all theory derived so far can be applied. As mentioned in the previous chapter, the situation changes dramatically when $\mathcal{X}_d \subset \mathcal{X}$ is discrete, which, in general, is *not* a convex subset. Thus, similar extension of f to $f_{\mathcal{X}_d}$ would, strictly speaking, not be a convex function.

5.1. Simulation results

5.1.1. Quadratic costs, simple dynamics in $\mathbb{R}$

Example 5.1. Consider the following system in $\mathbb{R}$ with quadratic costs.

$$\begin{aligned} \mathcal{X} &= [-1, 3], \\ \mathcal{U} &= [-1, 2], \\ f(x, u) &= x + u, \\ J_T(x) &= (x - 2)^2, \\ g(x, u) &= x^2 + (u - 1)^2. \end{aligned}$$

Note that this system complies with Assumption $\mathcal{A}.3$ (separable running cost; linear dynamics; convex, proper, lsc cost functions), therefore Fast Dynamic Programming (Algorithm 3) can be used

for computations. The analytical results for two steps of DP are given as:

$$\mathsf{DP}[J_T](x) = \frac{3}{2}x^2 - x + \frac{1}{2}, \qquad -1 \le x \le 3, \tag{5.1}$$

$$\mathsf{DP}^2[J_T](x) = \begin{cases} \frac{8}{5}x^2 + \frac{4}{5}x + \frac{3}{5}, & -1 \le x \le \frac{8}{3}, \\ \frac{5}{2}x^2 - 4x + 7, & \frac{8}{3} \le x \le 3. \end{cases} \tag{5.2}$$

In this case, the general solution $\mathsf{DP}^t[J](x)$ is (piecewise) quadratic but a closed form solution is difficult to obtain due to the state and control constraints.

According to Proposition 4.7 and Remark 4.8 we have perfect reconstruction for these choices of discrete sets:

$$\mathcal{X}_d = \{-1, -1+\Delta, -1+2\Delta, \dots, 3\} \subset [-1, 3], \tag{5.3}$$

$$\mathcal{U}_d = \{-1, -1+\Delta, -1+2\Delta, \dots, 2\} \subset [-1, 2]. \tag{5.4}$$

For properly chosen Δ, this particular choice of $\mathcal{X}_d$ and $\mathcal{U}_d$ guarantees the first condition.[1] The dual variable can be obtained by, for example,

$$\tilde{s}(x, u) = 2x + 2u - 4 + \Delta, \quad \forall x \in \mathcal{X}_d, \forall u \in \mathcal{U}_d. \tag{5.5}$$

This leads to $\tilde{\mathcal{S}}_d = \{-8+\Delta, -8+3\Delta, \dots, 6+\Delta\}$ which satisfies the second condition. Note that this set of dual variables only guarantees equality to the brute-force method for applying DP_d only once, however, perfect reconstruction happens for multiple steps in some cases.

We show performance – both time complexity and error – for different values of Δ, see Table 5.1. Iteration and setup times for both BF and FDP were measured, as well as the largest absolute difference between BF and FDP, i.e. $\bar{\epsilon}_t := \max \epsilon_t = \max |J_{t,BF} - J_{t,FDP}|$. Some of results are compared to brute-force method, see Figures 5.1 and 5.2.

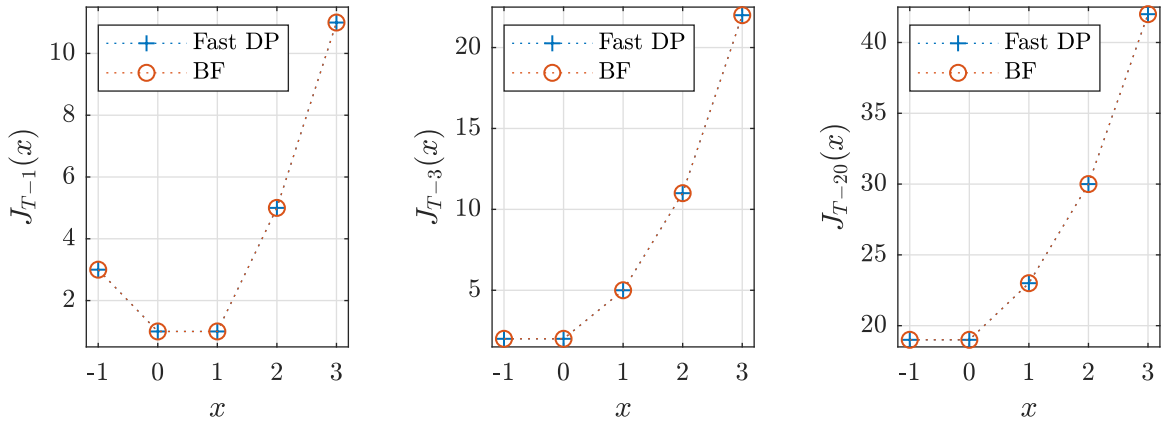
As expected, there is no difference between BF and FDP for the first step. When $\Delta = 1$, equality holds for multiple steps as well. The reason behind this is that $\mathcal{S}_d$ is able to cover the subdifferentials ∂g and ∂J_t as J_t changes. For $\Delta \neq 1$ there is an error emerging. Looking at the graph of J_t (Figure 5.2) it is apparent that the difference between BF and FDP only persists near the extremities, i.e. near $x = 3$. Here, the slope of J_t is the largest, which possibly lies outside of the range of $\tilde{\mathcal{S}}_d$. By increasing the boundaries of $\tilde{\mathcal{S}}_d$ – say $\tilde{\mathcal{S}}_d \subset [-8, 10]$ instead of the current $\tilde{\mathcal{S}}_d \subset [-8, 6]$ – the difference between BF and FDP for multiple steps may be reduced.

Table 5.1: Time comparison of brute-force minimization (BF) and Fast Dynamic Programming (FDP) for solving Example 5.1. The running times are given in milliseconds, were averaged over 20 runs, and each run computed solutions for $T = 20$. The largest difference between BF and FDP is measured as $\bar{\epsilon}_t := \max \epsilon_t = \max |J_{t,BF} - J_{t,FDP}|$.

Δ	X	U	$\tilde{S}$	Setup BF [ms]	Iter BF [ms]	Setup FDP [ms]	Iter FDP [ms]	$\bar{\epsilon}_1$	$\bar{\epsilon}_3$	$\bar{\epsilon}_{20}$
1	5	4	8	0.03	0.17	1.69	0.45	0	0	0
0.1	41	31	71	0.13	0.23	2.54	0.46	0	0.18	1.02
0.01	401	301	701	2.33	4.22	2.82	0.60	0	0.22	1.08
0.001	4001	3001	7001	410.02	578.32	7.47	2.11	0	0.22	1.09

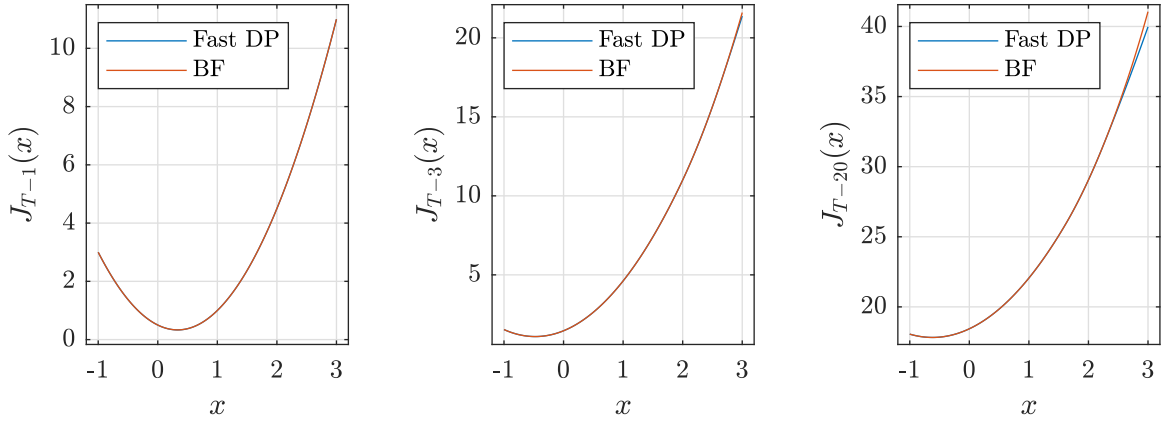
Remark 5.2. For this particular example, the iteration converges in the sense that $\mathsf{DP}[J] = J + \alpha$ up to a constant α. This convergence is attained after $T = 6$ steps, more explicitly, $\max |\mathsf{DP}_d^6[J] - \mathsf{DP}_d^T[J]| < 10^{-4}$ for any $T > 6$.

[1] As long as Δ is chosen such that the upper bounds of the sets are also included in the discrete sets, the result will cover the intended continuous sets. For example, $\Delta = 0.1$ covers both $\mathcal{X}$ and $\mathcal{U}$, but $\Delta = \frac{11}{28}$ does not.



(a) 1 step. (b) 3 steps. (c) 20 steps.

Figure 5.1: Results of simulation of Example 5.1 with $\Delta = 1$. In this particular case we have equality for all steps in the given time horizon.



(a) 1 step. (b) 3 steps. (c) 20 steps.

Figure 5.2: Results of simulation of Example 5.1 with $\Delta = 0.01$. The graph of BF covers that of FDP for most x since they are essentially the same. As the conditions of Proposition 4.7 are not fulfilled for all t – note that $\tilde{\mathcal{S}}$ is fixed in this example, – the solution of Fast DP tends to drift off near $x = 3$.

5.1.2. Non-quadratic cost function

Example 5.3. Consider the following system in $\mathbb{R}$ with non-quadratic costs:

$$\begin{aligned} \mathcal{X} &= [-2, 2], \\ \mathcal{U} &= [-2, 2], \\ f(x, u) &= ax + bu = 0.7343x + 1.1313u, \\ J(x) &= \cosh(x) - 1, \\ g(x, u) &= \cosh(x/2) + 0.9513 \cosh(1.1892u) - 1.9513. \end{aligned}$$

For this example there is no closed form analytical solution available. The cost functions g and J_T in this example contain constant additive terms to make $x = 0$ the optimum solution with $J_t(0) = 0$. Ensuring this is not necessary and could be left out without any penalty whatsoever. The motivation behind including such constants is that the graph of the cost functions J_t will have the same optimizer ($u^{opt} = 0$ at $x = 0$), and thus changes in the graphs may be compared more easily.

The state spaces were covered with equidistant points starting from -2 and ending at 2 for $\mathcal{X}_d$. As for the action and dual spaces, 3 cases are presented:

I $\mathcal{U}_d$ and $\tilde{\mathcal{S}}_d$ are such that Proposition 4.7 holds. As expected, the error is (close to) zero after the first step (see Figure 5.3a), however, the error accumulates for multiple steps. There is still a non-

zero error after the first step for larger cardinalities of $\mathcal{X}_d$ that may be caused by (1) numerical errors in computation of the correct elements of $\mathcal{U}_d$, or (2) some deeper underlying issue that Proposition 4.7 failed to uncover. Note that in this case $U \approx X^2$, and therefore $S \approx X^2$. For larger X this made infeasible to compute BF within reasonable time, hence the NaNs in Table 5.2. Further, computing all possible u such that the first condition of Proposition 4.7 holds is really expensive as can be seen from the starred element of Table 5.2.

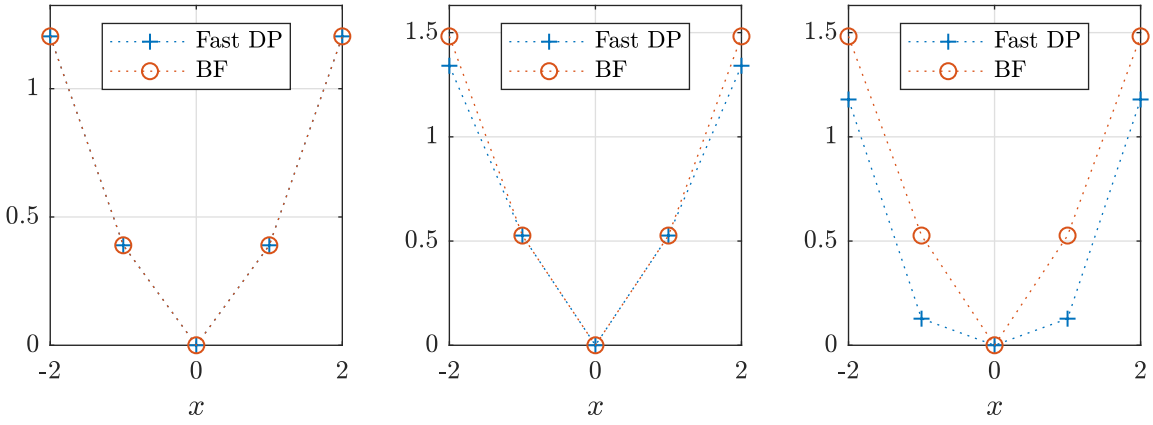
II $\mathcal{U}_d = \mathcal{X}_d$. Doing so fails the first condition of Proposition 4.7. Meanwhile, $\tilde{\mathcal{S}}_d$ is kept such that the second part of the same proposition is still satisfied. Error comes into play immediately after the first step (see Figure 5.3b). However, the computational time required to compute the result using FDP with great accuracy dwarfs that of BF for large cardinalities.

III $\mathcal{U}_d = \mathcal{X}_d$ and the dual space was chosen to be $\tilde{\mathcal{S}}_d \in [-6, 6]$ with the same spacing as that of $\mathcal{X}_d$ and $\mathcal{U}_d$. This interval and spacing was purposefully chosen such that S of case II is the same as S of this case, making case II and III more comparable in terms of complexity. Similarly to case II, error is present after the first step of DP (see Figure 5.3c).

In all cases and $g^*(-b\tilde{s})$ and $j^*_{\tilde{s}}(ax)$ were computed exactly. In $\mathbb{R}$, computing the composite of conjugation and linear transformation only marginally increases time complexity: direct computation is possible with LLT, only reordering of entries may be needed $b > 0$ or $a < 0$.

The execution times and maximum absolute errors $\bar{\epsilon}_t := \max \epsilon_t = \max |J_{t,BF} - J_{t,FDP}|$ – whenever available – can be found in Table 5.2. In terms of absolute error, case I dominates as expected. However, since U has to be large to comply with the requirements, case I takes the longest amount of time to compute for the same X. Case II and case III show minor differences in terms of execution time, which is reasonable since they operate with the same cardinalities. These differences originate from the extra computations necessary to compute the elements of $\tilde{\mathcal{S}}_d$. For these runs, this contributed to a 3–5 % increase in setup time of FDP compared to case III. More importantly, the absolute error is lower on average for case II than for case III – about 57 % better compared to case III, – but not exclusively. This suggests that partially satisfying Proposition 4.7 can be still beneficial. It is up to the designer to weigh the benefits of time complexity and error in the one dimensional case, and choose between cases I, II and III.

This example reaches a steady-state after about 7 steps. The solution changes insignificantly after about 7 steps, meaning $\max |J_{T-7} - J_{T-20}| < 10^{-4}$, for both BF and FDP.



(a) Case I, $U = 21, S = 29$. (b) Case II, $U = 5, S = 13$. (c) Case III, $U = 5, S = 13$.

Figure 5.3: $J_{T-1}(x)$ for Example 5.3 with $X = 5$ for different cases. The difference between BF and FDP is at its highest near the boundaries for cases I and II, i.e. when $x = \pm 2$, while for case III the largest difference is at $x = \pm 1$.

5.1.3. Example in $\mathbb{R}^3$

In the following example we demonstrate the scalability of Fast DP.

Table 5.2: Time comparison of brute-force minimization (BF) and Fast Dynamic Programming (FDP) for solving Example 5.3. The running times were averaged over 20 runs, and each run computed solutions for $T = 20$. The largest difference between BF and FDP is measured as $\bar{\epsilon}_t := \max \epsilon_t = \max |J_{t,BF} - J_{t,FDP}|$.

Case	X	U	$\tilde{S}$	Setup BF [ms]	Iter BF [ms]	Setup FDP [ms]	Iter FDP [ms]	$\bar{\epsilon}_1$	$\bar{\epsilon}_3$	$\bar{\epsilon}_{20}$
I	5	21	29	0.04	0.18	5.20	0.45	0	5.75E-2	7.10E-2
I	21	377	417	0.39	0.54	12.39	0.50	1.63E-6	6.80E-4	1.13E-3
I	101	8893	9093	21.05	40.52	351.22	1.68	3.65E-8	7.47E-6	1.62E-5
I	501	219697	220697	NaN	NaN	44594*	41.15	NaN	NaN	NaN
II	5	5	13	0.04	0.16	4.53	0.45	1.41E-1	1.64E-1	2.30E-1
II	21	21	61	0.10	0.19	6.10	0.46	2.00E-2	8.72E-3	1.12E-2
II	101	101	301	0.75	0.64	8.63	0.51	5.11E-4	5.15E-4	6.52E-4
II	501	501	1501	12.10	9.95	18.85	0.75	3.98E-5	2.10E-5	3.48E-5
II	2501	2501	7501	299.60	284.66	78.84	2.22	1.58E-6	8.70E-7	1.60E-6
III	5	5	13	0.03	0.16	4.38	0.44	3.99E-1	9.29E-1	9.18E-1
III	21	21	61	0.11	0.18	5.81	0.46	1.53E-2	3.90E-2	6.11E-2
III	101	101	301	0.75	0.62	8.23	0.50	8.54E-4	1.70E-3	4.27E-3
III	501	501	1501	11.83	9.75	18.14	0.71	3.73E-5	7.94E-5	1.98E-4
III	2501	2501	7501	294.38	285.69	76.38	2.00	1.60E-6	3.28E-6	1.01E-5

Example 5.4. Consider the following system in $\mathbb{R}^3$ with quadratic costs:

$$\begin{aligned}
\mathcal{X} &= [-2, 2]^3, \\
\mathcal{U} &= [-4, 4]^2, \\
f(x, u) &= Ax + Bu, \\
J_T(x) &= (x - x_0)^\top P(x - x_0), \\
g(x, u) &= x^\top Qx + (u - u_0)^\top R(u - u_0),
\end{aligned}$$

where

$$A = \begin{bmatrix} 1.1123 & 0.7038 & -1.1879 \\ -0.9585 & 0.9226 & -0.9054 \\ 0.7233 & -0.2692 & 1.5495 \end{bmatrix}, \quad B = \begin{bmatrix} 0.9920 & -0.2552 \\ -0.2778 & -1.3998 \\ 0.4704 & 0.8058 \end{bmatrix}, \quad x_0 = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix},$$

$$P = \begin{bmatrix} 5.9932 & 3.6069 & 0.7188 \\ 3.6069 & 2.5632 & 0.9088 \\ 0.7188 & 0.9088 & 1.5555 \end{bmatrix}, \quad Q = \begin{bmatrix} 1.8016 & 2.7613 & 0.9746 \\ 2.7613 & 5.8743 & 0.0903 \\ 0.9746 & 0.0903 & 2.0239 \end{bmatrix},$$

$$u_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad R = \begin{bmatrix} 1.1036 & -0.2157 \\ -0.2157 & 1.3525 \end{bmatrix}.$$

Note that matrices P, Q, and R are symmetric positive definite, thus making J_T and g convex functions. Table 5.3 shows the execution times and errors associated with this example. Figure 5.4 further details the error distribution for some selection of cardinalities of the discrete sets.

Similarly to before, the discrete state and action spaces were set with equidistant points including their respective limits. The dual space was chosen to be $\tilde{\mathcal{S}}_d \in [-20, 20]^3$ with equal spacing. Referring to Proposition 4.7, this discrete dual space is not adequate for error-free solutions. Furthermore, an additional dual space is introduced in this case: $\mathcal{V}_d$ with $|\mathcal{V}_d| = V$ which is the dual of $\mathcal{U}$. Since $g^*(-B^\top \tilde{s})$ needs to be computed for all $\tilde{s}$ and $\tilde{S} \gg U$, it is costly to compute it exactly. Therefore, we first compute $g^*(v)$ for all $v \in \mathcal{V}_d$ and then apply the same approximation (i.e. linear interpolation) to obtain $g^*(-B^\top \tilde{s})$ from the samples of $g^*(v)$. For this example, $\mathcal{V}_d$ was chosen as a rectangular box with limits $v_{i,min} = \min_{\tilde{s}}(-B^\top \tilde{s})_i$ and $v_{i,max} = \max_{\tilde{s}}(-B^\top \tilde{s})_i$ for the two dimensions of the action space, i.e. $i \in \{1, 2\}$. Although the choice of $\mathcal{V}_d$ should affect the error, it only marginally improves the error. The number of points between $v_{i,min}$ and $v_{i,max}$ is what ultimately defines how the error of interpolation comes to play in the end result.

Looking at Table 5.3, a clear advantage of FDP can be seen for the execution times: in all cases BF is slower and, with increasing cardinalities, its iteration time becomes orders of magnitude worse than that of FDP. Moreover, the iterations times support our claims of time complexity in Corollary 4.10.1.

The maximum error, $\bar{\epsilon}_t := \max \epsilon_t = \max |J_{t,BF} - J_{t,FDP}|$, seems to be disproportionally large for small cardinalities (see Table 5.3). Since $\tilde{\mathcal{S}}_d$ is only a crude representation of ∂g and ∂J_t when $\tilde{S}$ is small, this is to be predicted. Interestingly, increasing $\tilde{S}$ alone does not necessarily yields better results proportional to the increase, see for example the runs with $\tilde{S} = 21^3 = 9261$ and $\tilde{S} = 41^3 = 68921$. In these cases, the error decreased by 12–60% at the expense of increasing the amount of points to be computed by $41^3/21^3 = 7.44$ times. Consequently, there is a diminishing return in terms of accuracy when increasing $\tilde{S}$.

The reason why the maximum *relative* error remains large ($> 40\%$) in Figure 5.4 is because while the absolute error decreases (see last columns of Table 5.3), it is still present near the optimum which is close to zero. This way the absolute error gets magnified around the optimum, resulting in large relative error. However, in general, the mean (relative) error tends to zero for increasing cardinalities: on Figure 5.4 the mean relative errors are 36.88%, 3.81%, and 1.89% respectively.

Table 5.3: Time comparison of brute-force minimization (BF) and Fast Dynamic Programming (FDP) for the solutions Example 5.4. The running times show a single run, and each run computed solutions for $T = 2$. The largest difference between BF and FDP is measured as $\bar{\epsilon}_t := \max \epsilon_t = \max |J_{t,BF} - J_{t,FDP}|$.

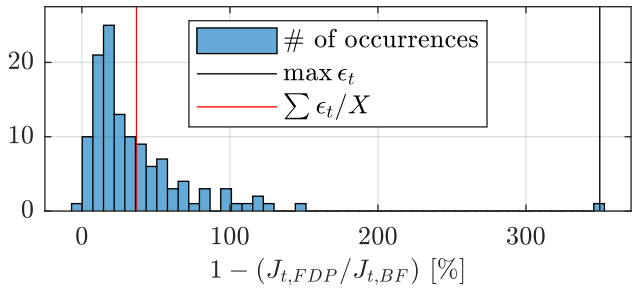
X	U	$\tilde{S}$	V	Setup BF [s]	Iter BF [s]	Setup FDP [s]	Iter FDP [s]	$\bar{\epsilon}_1$	$\bar{\epsilon}_2$
125	81	1331	121	<0.01	0.21	0.03	0.02	48.67	58.18
125	81	9261	441	<0.01	0.21	0.07	0.05	43.67	47.40
125	289	1331	121	<0.01	0.49	0.03	0.02	11.48	21.15
125	289	1331	441	<0.01	0.49	0.03	0.02	6.59	11.60
125	289	9261	441	<0.01	0.49	0.07	0.05	4.22	6.92
125	289	68921	441	<0.01	0.49	0.33	0.17	3.41	5.72
125	289	68921	1681	<0.01	0.49	0.33	0.17	2.91	3.52
729	289	68921	1681	<0.01	2.88	0.33	0.19	5.55	7.71
729	625	9261	1681	<0.01	5.63	0.08	0.06	3.44	4.18
729	625	68921	1681	<0.01	5.55	0.33	0.19	3.01	3.28
729	1089	9261	1681	<0.01	9.21	0.08	0.06	2.51	3.62
729	1089	68921	1681	<0.01	9.26	0.34	0.19	1.49	2.58
4913	1089	68921	1681	0.02	62.11	0.36	0.24	2.38	3.69
4913	1089	531441	1681	0.02	62.09	2.53	0.78	2.26	3.53
4913	1089	531441	6561	0.02	62.36	2.55	0.78	1.82	2.90
4913	4225	9261	6561	0.02	231.73	0.10	0.09	1.91	3.24
4913	4225	68921	6561	0.02	237.32	0.39	0.24	0.78	1.26
4913	4225	4173281	6561	0.02	231.73	21.12	3.22	0.66	0.99

5.2. General remarks

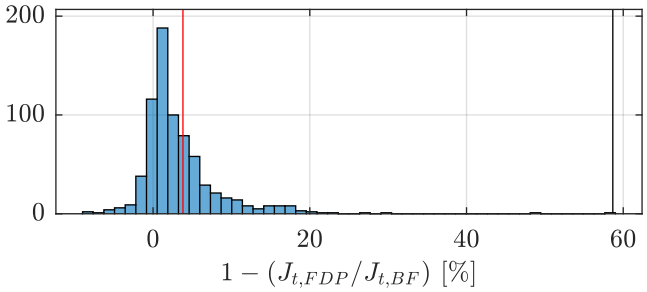
Two instances of the most common type of examples were shown: systems with quadratic costs. These are well-studied and well-understood problems with analytical results available for unconstrained states and actions.

The algorithms work also with non-quadratic costs as it is demonstrated in Example 5.3. As long as Assumption $\mathcal{A}.1$ is fulfilled, the algorithms presented here can be used. In fact, for a finite set of points $\{x_i\}_{i\in I}$ and $\{u_j\}_{j\in J}$, if the points on the cost functions $J_T(x_i)$ and $g(x_i, u_j)$ contribute to the convex hull of J_T and g for all $i \in I$ and $j \in J$, then our algorithms could be used. In other words, there need not be a (closed form) function that describes the relationship between x_i, u_j and $J_T(x_i)$ and $g(x_i, u_j)$; everything can be data-driven.

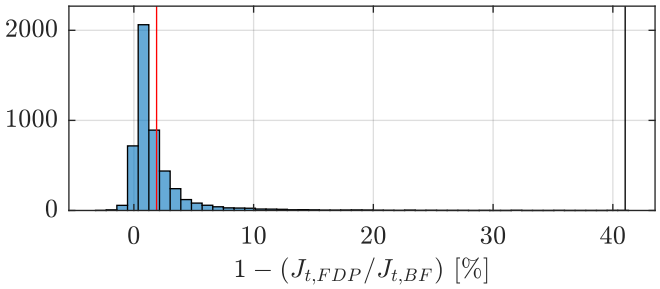
The setup times of Example 5.1 and Example 5.3 that can be seen in Table 5.1 and 5.2 require some comments:



(a) $X = 125, U = 81, \tilde{S} = 1331, V = 121$.



(b) $X = 729, U = 289, \tilde{S} = 68921, V = 1681$.



(c) $X = 4913, U = 4225, \tilde{S} = 68921, V = 6561$.

Figure 5.4: Distribution of relative error as a percentage compared to BF in Example 5.4. Different cardinalities are shown. The number of occurrences of $\epsilon_t = 1 - (J_{t,FDP}/J_{t,BF})$ are in blue, the mean relative error, $\sum \epsilon_t / X$, is represented as a red vertical line, and the maximum relative error, $\max \epsilon_t$, is the black vertical line.

- It turned out that for these particular examples and implementation, BF performed better in the iterations when $g(x, u)$ was precomputed and stored before entering the loop. Hence the setup time of BF is of $O(XU)$, and this showcases a "best-case" scenario for BF. On the other hand, this required storage of all $g(x, u)$ which prohibited large-scale tests ($X \approx U > 10^4$) due to memory limitations. For example, observe the last row of case I in Table 5.2 where BF failed to compute because of insufficient memory. While FDP could perform in such an extreme situation, its setup time was greatly penalized due to the computation of points $-b\tilde{s}$ (see starred element of Table 5.2).
- The comparably large setup time for FDP for small cardinalities in this case is due to the overhead of computing $g(x, u)$ and $g^*(-b\tilde{s})$ with LLT. To this end, a custom code was implemented that would work in the general case, for multiple dimensions at a cost of additional overhead. This is why the BF needs less setup time than FDP for small cardinalities.
- The setup times of the last example used a different implementation of BF to allow larger discrete sets to be tested. Note that this has put a heavy penalty on the iteration times of BF: for $X = 2501$ and $U = 2501$ (Example 5.3) each iteration cost around 280 ms, while this for $X = 729$ and $U = 625$ (Example 5.4) is around 5 s(!).

It is apparent from Table 5.1 that FDP is not always the fastest solution to solve DP problems

numerically. Namely, for small state and action spaces it is better to compute the optimal cost by naive minimization because of the reasons mentioned above. However, the performance of brute-force minimization quickly becomes inferior if the cardinality of the state and action spaces reach about 100. Keep in mind that this holds also in higher dimensions meaning that this border is reached rather quickly – assuming the sets have roughly the same elements along all dimensions. This is also supported by the execution times of Example 5.4 which can be seen in Table 5.3.

The choices of $\mathcal{S}_d$ and $\tilde{\mathcal{S}}_d$ are difficult. When fixed, g^* can be precomputed which places the burden of costly computations outside of the loop. However, when the dual spaces are fixed, they may be no longer capable of covering the subdifferentials of evolving cost functions J_t leading to FDP being smaller than BF for extreme slopes. This, in turn, introduces unwanted error as can be seen, for example, from Figure 5.2. After 3 steps (Figure 5.2b), the dual space becomes "obsolete" and loses its ability to cover the subdifferentials of J_t. The effect is more pronounced after 20 steps (Figure 5.2c). As a rule of thumb, $\mathcal{S}_T = \partial g \cup \partial J_T$ may be extended to twice or three times its size to accommodate the growth of $\partial\,\mathsf{DP}^t[J]$ without sacrificing much performance. Note that the algorithm has linear time complexity in $|\mathcal{S}|$ so doubling its size "only" costs double the time. Another ad-hoc way of determining the size of $\mathcal{S}$ for longer horizon is to look at the slopes of the solution after t steps. If they saturate around the extremes – near the edges of the bounding box of x – then the dual sets *likely* need to be extended.

On the other hand, if the dual spaces can be varied with t, then, in principle, $\mathcal{S}$ and $\tilde{\mathcal{S}}_d$ can be chosen such that they provide adequate coverage of the dual space. This is not an issue for problems in $\mathbb{R}$, however, when the dimension of the problem increases, computing such dual sets is expensive. In general, there are X coordinates along each dimension for such error-free $\mathcal{S}_d$ and $\tilde{\mathcal{S}}_d$ which leads to $|\mathcal{S}_d| = X^n$. This quickly becomes impractical, even for small $n > 1$. Moreover, the setup times of FDP indicated in Tables 5.1 and 5.3, would transfer to the iteration times. In all cases this is a significant increase of the overall algorithm. Possible mitigations to this could be to:

- update $\mathcal{S}_d$ after every kth step,
- choose the (convex) domain of $\mathcal{S}_d$ larger than necessary before starting the algorithm, and fix it,
- approximating $s \notin \mathcal{S}_d$ from samples within the set (this is what we did in Example 5.4),
- some combination of the above.

Unsurprisingly, there is a trade-off between error and time complexity ("There is no free lunch"). Leaving no room for error leads to problems and plausible solutions mentioned above, all of which increase execution time of the algorithms considerably. In turn, if one allows error the running times of FDP beats that of BF by a huge margin as the problem scales. On top of this, the error shrinks to zero with finer levels of discretization.[2] This means that for problems with large number of states/actions, the results obtained by FDP come close to the results of BF at a fraction of the time! For instance, observe the last rows of Table 5.3. The mean error in those cases were less than about 1–2%, however, the computational time was around 70–2450 times faster(!) not including the setup time. The discrepancy of decreasing error and execution time compared to BF for growing cardinalities puts FDP truly in a greatly desired spot for computing solutions to DP problems.

[2]This holds if the convex hull of $\mathcal{S}$ covers all subdifferentials of J_t. Choosing such $\mathcal{S}$ is not difficult, only the boundaries of ∂J_t has to be known.

6

Conclusion

In this chapter we reiterate what has been achieved in the thesis and give directions for future work.

First, the dynamic programming (DP) problem was reformulated using the Legendre–Fenchel conjugate in the continuous domain. Under different levels of assumptions, alternative routes were derived to solve DP. The most general case concerned with convex, lower semicontinuous, proper cost functions. Equivalence of direct minimization and "biconjugation" was proven under mild assumptions using the Fenchel–Moreau theorem. In order to handle the constraints imposed by the dynamics, perturbed versions of solution operator were introduced. Doing so allowed the problem to be solved more conveniently in the dual domain by simple addition, and, because of the aforementioned equivalence, by applying conjugation once more to the dual problem.

Further, we showed the true potential of our approach when the running costs are separable in state and action. In this case, the cost of introducing perturbations disappeared, leading to *no* state expansion. This later proved to be an enormous advantage from a computational and storage point of view and has lead to the foundations of Fast Dynamic Programming (FDP) algorithm. Note that, in the realms of optimal control, it is not uncommon to have such separable cost function; think about LQ control.

Second, the continuous results were further refined to be able to handle discrete sets, thus giving rise to numerical algorithms. Implementation of the theory proved to be a challenge due to the discrete nature of the problem: the equivalence of direct minimization and biconjugation is only guaranteed under much stricter conditions – compared to the continuous case. Ensuring these conditions, however, is rather costly in general, therefore approximations are made to enjoy the computational gain of our framework. These approximations appeared when the composite of conjugation and linear transformation needed to be computed, and with them errors were introduced to the algorithms.

For separable running cost functions, FDP was derived which has $O(U+T(X+S))$ time complexity for U actions, X states, S dual states and T time-horizon.

All these detours from direct optimization hid the burden of minimization behind conjugation, and the solution operator became a combination of addition and (linear) transformations in the dual domain. We utilized Linear-time Legendre–Fenchel (LLT) algorithm [22] to compute the conjugates of the functions.

Last, numerical examples were shown supporting theoretical derivations. We demonstrated superior performance compared to direct minimization. While for small state and action spaces brute force minimization provides more accurate – and in some cases faster – results than FDP, this advantage of the brute force method quickly diminishes with growing state/action spaces.

Our work provides a way to compute DP using conjugation, however, we believe that there is much more to explore in this topic. First, how to choose H appropriately such that a computational advantage is gained for $\widetilde{\mathsf{DP}}_H$ which forms the foundation of FDP? Generally, H, or more precisely H^{-1} and its transpose, controls 3 things: (1) $\mathsf{DP}[J](x) = \widetilde{\mathsf{DP}}_H[J](H^{-1}(x,0))$, i.e. which points of $\widetilde{\mathsf{DP}}_H[J]$ need to be calculated to get $\mathsf{DP}[J]$; (2) where g^* is evaluated in $\widetilde{\mathsf{DP}}_H^*$; and (3) where J^* needs to be computed in $\widetilde{\mathsf{DP}}_H^*$. Ideally, we want to avoid creating Minkowski-sums of the dual spaces in (2) and (3), as well as avoid creating computation of the composite of conjugation and linear transformation $(h^*(Mx))$ for

all cases. The former causes explosion of states in the discrete algorithms, while the latter generates problems in the evaluation of the composite with boxed grids. Managing H to satisfy all our needs is complicated and the "optimal" solution may require H to depend on (the type of) the problem.

Another question related to the algorithms presented here is how to choose (discrete) dual sets $\mathcal{S}_d$ and $\tilde{\mathcal{S}}_d$ such that these algorithms are fast, accurate, most desirably both? Can they be chosen a priori such that they produce no/minimal error for multiple steps; if so, how? We demonstrated with numerical examples that choosing a fixed $\tilde{\mathcal{S}}_d$ may lead to (approximation) errors under quicker execution times compared to brute-force minimization. While error-free definition of the dual spaces is possible, doing so in each step of DP hinders the potential gain in execution times of the algorithms. This could be circumvented by introducing approximations, but it remains to be shown how the trade-off between speed and accuracy may be balanced in an optimal way.

Additionally, here are some unanswered questions, in no particular order, that came up during development and that may be worth pursuing in the future:

- How may state and/or action constraints, other than e.g. $x \in [a, b]$, be incorporated in our framework?
- How can the composite of conjugation and linear transformation, for example $g^*(-B^\top \tilde{s})$, be computed in an efficient way? Can it be exact without a great hit on the performance? If not, how can the error be improved?
- How does the error propagate through multiple steps? Does it stay bounded if $T \to +\infty$? Under what conditions does it have bounded error? What modifications need to take place in the algorithm – possibly by choosing dual discrete sets and/or H more carefully – to ensure bound on the error?
- Convergence? Under what (minimal) conditions do the algorithms presented here converge to a "steady-state" solution after $T \gg 1$ steps, i.e. $\mathsf{DP}[J] = J$? Is it possible to compute this steady-state solution efficiently using our framework, by possibly "skipping" multiple steps in one computation?
- Could these theoretical results be in some way expanded to:
  - Nonlinear cases? What if the dynamics, f, is nonlinear?
  - Nondeterministic cases, i.e. Stochastic Dynamic Programming?
  - Non-convex cost functions?
- Is it possible to remain in the conjugate domain for multiple steps – similarly to [30] – under less restrictive conditions than in [30]?
- By involving results from max-plus algebra, is it possible to extend the theoretical part?
- Quadratic functions of the form $q = (1/2)\|\cdot\|^2$ are eigenfunctions of the conjugation, i.e. $q = q^*$. Could this be in some way incorporated in our framework to derive results for quadratic cost functions?
- How does our algorithms presented here compare to other state-of-the-art algorithms in terms of computational time, storage, and accuracy?

It is hoped that these questions get their answers in the future.

Bibliography

[1] F. Alvarez and N. L. Stokey, "Dynamic programming with homogeneous functions," *Journal of economic theory*, vol. 82, no. 1, pp. 167–189, 1998.

[2] J. Wijngaard *et al.*, "Decomposition for dynamic programming in production and inventory control," *Engineering and process economics*, vol. 4, no. 2-3, pp. 385–388, 1979.

[3] W. H. Fleming and W. M. McEneaney, "A Max-Plus-Based Algorithm for a Hamilton–Jacobi–Bellman Equation of Nonlinear Filtering," *SIAM Journal on Control and Optimization*, vol. 38, no. 3, pp. 683–710, 2000.

[4] G. Righini and M. Salani, "New dynamic programming algorithms for the resource constrained elementary shortest path problem," *Networks: An International Journal*, vol. 51, no. 3, pp. 155–170, 2008.

[5] D. Wang, D. Liu, H. Li, and X. Yang, "Robust optimal control for a class of nonlinear dynamic systems using single network adaptive dynamic programming," in *Proceedings of the 33rd Chinese Control Conference*. IEEE, 2014, pp. 8711–8716.

[6] B. Luo, D. Liu, T. Huang, and J. Liu, "Output tracking control based on adaptive dynamic programming with multistep policy evaluation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, no. 99, pp. 1–11, 2017.

[7] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[8] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley & Sons, 2007, vol. 703.

[9] A.-m. Farahmand, C. Szepesvári, and R. Munos, "Error propagation for approximate policy and value iteration," in *Advances in Neural Information Processing Systems*, 2010, pp. 568–576.

[10] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*. Athena Scientific Belmont, MA, 1996, vol. 5.

[11] M. Schetzen, *Linear time-invariant systems*. J Wiley-Interscience, 2003.

[12] J. W. Cooley, "The Re-Discovery of the Fast Fourier Transform Algorithm," *Microchimica Acta*, vol. 93, no. 1-6, pp. 33–45, 1987.

[13] D. N. Rockmore, "Recent progress and applications in group FFTs," in *Computational noncommutative algebra and applications*. Springer, 2004, pp. 227–254.

[14] S. Gaubert and M. Plus, "Methods and Applications of (max,+) Linear Algebra," in *Annual symposium on theoretical aspects of computer science*. Springer, 1997, pp. 261–282.

[15] J. Komenda, S. Lahaye, J.-L. Boimond, and T. Van den Boom, "Max-plus algebra in the history of discrete event systems," *Annual Reviews in Control*, vol. 45, pp. 240–249, 2018.

[16] H. Kaise and W. M. McEneaney, "Idempotent expansions for continuous-time stochastic control: compact control space," in *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 2010, pp. 7015–7020.

[17] M. Akian and E. Fodjo, "A probabilistic max-plus numerical method for solving stochastic control problems," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 7392–7397.

[18] A. Fahim, N. Touzi, X. Warin *et al.*, "A probabilistic numerical method for fully nonlinear parabolic PDEs," *The Annals of Applied Probability*, vol. 21, no. 4, pp. 1322–1364, 2011.

[19] M. Fidler and S. Recker, "Conjugate network calculus: A dual approach applying the Legendre transform," *Computer Networks*, vol. 50, no. 8, pp. 1026–1039, 2006.

[20] R. Bellman and W. Karush, "Mathematical Programming and the Maximum Transform," *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, no. 3, pp. 550–567, 1962.

[21] B. Burgeth and J. Weickert, "An explanation for the logarithmic connection between linear and morphological system theory," *International Journal of Computer Vision*, vol. 64, no. 2-3, pp. 157–169, 2005.

[22] Y. Lucet, "Faster than the fast Legendre transform, the linear-time Legendre transform," *Numerical Algorithms*, vol. 16, no. 2, pp. 171–185, 1997.

[23] ——, "A fast computational algorithm for the Legendre–Fenchel transform," *Computational Optimization and Applications*, vol. 6, no. 1, pp. 27–57, 1996.

[24] L. Corrias, "Fast Legendre–Fenchel transform and applications to Hamilton–Jacobi equations and conservation laws," *SIAM journal on numerical analysis*, vol. 33, no. 4, pp. 1534–1558, 1996.

[25] P. F. Felzenszwalb and D. P. Huttenlocher, "Distance transforms of sampled functions," *Theory of computing*, vol. 8, no. 1, pp. 415–428, 2012.

[26] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison, "Applications of Legendre-Fenchel transformation to computer vision problems," *Department of Computing at Imperial College London. DTR11-7*, vol. 45, 2011.

[27] T. Hisakado, K. Okumura, V. Vukadinovic, and L. Trajkovic, "Characterization of a simple communication network using legendre transform," in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS'03.*, vol. 3. IEEE, 2003, pp. III–III.

[28] Y. Lucet, "What shape is your conjugate? a survey of computational convex analysis and its applications," *SIAM review*, vol. 52, no. 3, pp. 505–542, 2010.

[29] C. Klein and T. Morin, "Conjugate Duality and the Curse of Dimensionality," *European journal of operational research*, vol. 50, no. 2, pp. 220–228, 1991.

[30] R. Carpio and T. Kamihigashi, *Fast Bellman Iteration: an Application of Legendre-fenchel Diuality To a Class of Deterministic Dynamic Programming Problems in Discrete Time*. Research Institute for Economics and Business Administration, Kobe University, 2016.

[31] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[32] H. H. Bauschke, P. L. Combettes *et al.*, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, 2011, vol. 408.

[33] D. P. Bertsekas, *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 2017, vol. 1–2, no. 4.

[34] S. Kesavan, *Functional analysis*. Springer, 2009, vol. 52.

[35] D. P. Bertsekas, *Convex optimization theory*. Athena Scientific Belmont, 2009.

[36] K. Fujimoto, J. M. Scherpen, and W. S. Gray, "Hamiltonian realizations of nonlinear adjoint operators," *Automatica*, vol. 38, no. 10, pp. 1769–1775, 2002.

[37] J. M. Scherpen and W. S. Gray, "Nonlinear Hilbert adjoints: Properties and applications to Hankel singular value analysis," *Nonlinear analysis, theory, methods & applications*, vol. 51, no. 5, pp. 883–901, 2002.