Conjugate Dynamic Programming

Rodopoulos Charalampos

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Conjugate Dynamic Programming

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Rodopoulos Charalampos

November 25, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

In decision making problems, the ability to compute the optimal solution can pose a serious challenge. Dynamic Programming (DP) aims to provide a framework to deal with a category of such problems, namely ones that involve sequential decision making. By dividing the original control problem into sub problems and solving it backwards in time, from the end of the time horizon to the start, the method can compute a map of optimal solutions with respect to the initial condition. In order to divide the original problem into subproblems the DP method takes advantage of the principle of optimality, which states that a sub-solution of the optimal solution should be the optimal solution for the equivalent subproblem. In control systems, where the state and decision spaces are continuous, the original DP framework can be intractable due to the size of the discretization needed to simulate the continuous space. Therefore, efficient approximations are needed to solve such problems. One promising method is called Conjugate Dynamic Programming (CDP). The CDP algorithm is able to transform the original framework and solve problems in the conjugate domain providing a computational advantage over the standard method. In this work, we aim to improve and extend the setting under which the CDP algorithm operates, thus providing a more concrete advantage over standard method . In that regard, we will extract the optimal control actions from within the CDP algorithm, eliminating the need for solving an extra optimization problem for their computation. In addition, we will introduce a different interpolation technique that can outperform the current one, in certain scenarios, thus granting the user more choice when solving a decision making problem.

Table of Contents

List of Figures

List of Tables

Acknowledgements

The current thesis would not have been possible without my daily supervisors A. Sharifi Kolarijani and G.F. Max . Your feedback throughout this process was invaluable and allowed me to learn and grow as a person. Also a big thank you to my supervisor Dr. P. Mohajerin Esfahani for always being available when needed and for giving me the opportunity to work on such a challenging and rewarding problem.

To all my friends here in Delft, for all the good times we had together and for the fun inside and outside our studies. Also to my friends in Greece, who even though they were not here physically, they were always available when I needed someone to speak to.

Delft, University of Technology
November 25, 2021

Rodopoulos Charalampos

Chapter 1

Introduction

The motivation behind this work lies on solving the Dynamic Programming (DP) problem by expanding on the work of [1]. Dynamic Programming aims to solve the Bellman equation,

$$J_t(x_t) = \min_{u_t}\{C_t(x_t, u_t) + J_{t+1}(x_{t+1})\}, \tag{1-1}$$

by computing the optimal cost-to-go functions J_t backwards in time, given the running cost $C_t(x_t, u_t)$ of choosing an action u_t at a given state x_t, and the cost to go $J_{t+1}(x_{t+1})$. at the next step $t + 1$.

At the core of the DP formulation lies the principle of optimality introduced by Bellman [2]. By this we divide the original control problem into sub-problems which are simpler to solve. What makes the formulation of the DP problem attractive is that many interesting real world applications can be modeled and subsequently solved using DP. Some of these can be found in fields such as finance [3], power systems and the peak shaving problem [4, 5, 6, 7], as well as optimal control [8, 9], and inventory routing [10].

The ability to solve complex problems like the ones mentioned above in a computationally efficient manner marks the main point of this thesis. Approximating the cost-to-go function, also knows as value function, is the main interest behind solving the subclass of problems known as Approximate Dynamic Programming (ADP). The approximation is needed in problems where the computation of the exact cost-to-go function is impossible or requires an extremely large amount of resources . This can include problems in continuous spaces, where a sharp discretization would lead to the problem being intractable. An extensive overview of the theory and methods on how to solve such problems can be found in [11, 12, 13]. Popular methods include Q-Learning [14] and Temporal Difference (TD) Learning [15]. Furthermore, in [16], a method based on linear programming is proposed. The method tries to fit a linear combination of predetermined functions to approximate the cost-to-go function. Lastly, in [17, 18] a method is developed to compute max-plus linear approximations of the value function within a dictionary of functions. The necessity for ADP and generally more efficient ways to solve the DP problem stem from its biggest drawback, its high computational cost

when dealing with high dimensional state and action spaces, otherwise known as the curse of dimensionality [2]. In [1] an algorithm was developed, which exploits the Legendre-Fenchel transform [19],

$$F^*(s) = \sup_{x \in X} \{\langle x, s \rangle + F(x)\}, \qquad \forall s \in S \subseteq \mathbb{R}^N, \tag{1-2}$$

in order to solve the DP equation (1-1). In equation (1-2), F denotes a function with domain $X \subseteq \mathbb{R}^n$ and codomain $\mathbb{R}$, while F^* denotes its conjugate function with domain $S \subseteq \mathbb{R}^n$ and codomain $\mathbb{R}$. In equation 1-2 $\langle x, s \rangle$ denotes the inner product between the two variables s and x. In that regard, a linear time algorithm for computing the discrete Legendre transform [20], has been used for transforming the original problem and solving it in the conjugate domain. This has resulted in a more efficient implementation than solving the Bellman equation explicitly, which involves minimizing over the action space for any given state. Specifics on how the proposed algorithms in [1] work will be given in Chapter 2.

Contributions. The aim of this project is to improve the setting under which the Legendre-Fenchel transform can be used to solve the DP problem. In that regard, during this thesis we will aim to improve the computational aspect of working in the conjugate domain thus providing a more concrete advantage over traditional methods. More specifically, one key improvement revolves around extracting the optimal control action from within the CDP algorithm [1]. In the current implementation, in order to obtain the optimal control action, a minimization problem is solved during the forward stage of the algorithm, thus adding to the total computational cost. By leveraging the powers and capabilities of the LLT algorithm [20], we can extract the optimal control action during the backwards iteration at little to no extra computational cost, thus gaining an overall computational advantage. Furthermore, we aim to improve on other computational aspects of the algorithm by employing, possibly more efficient methods of interpolation/extrapolation without greatly affecting the overall computational cost of the algorithm.

Thesis organization. The current report is organized as follows. In Chapter 2, some theory that will be used throughout the thesis will be presented. This includes a short introduction into the CDP algorithm. In Chapter 3, we show theoretical results regarding the topics tackled in the present thesis. These include the theory behind extracting the optimal control input from within the CDP algorithm, and whether Kuhn triangulation can interpolate to a convex function, given that the discrete set of points are convex extensible. In Chapter 4, we present the numerical simulations and results associated with the theory of Chapter 3. In Chapter 5, we discuss the results and provide possible future research avenues.

Chapter 2

Theoretical Prerequisites

In this Chapter we will present some theoretical prerequisites that will be used throughout the thesis and are necessary for the reader to understand the following Chapters.

2-1 Legendre-Fenchel transform and LLT algorithm

The Legendre-Fenchel transform and the ability to compute it in linear time via the LLT algorithm [20] allows the CDP algorithm to outperform standard methods in computing the solution to DP problem. For completeness' sake, we briefly present the workings of the LLT algorithm. For more information regarding LLT the reader can refer to [20].

In equation (1-2), we presented the Legendre-Fenchel transform in continuous primal and dual spaces. The LLT algorithm computes the conjugation via an approximation, by discretizing the primal X and dual S spaces of equation (1-2). The discrete Legendre transform is presented in equation (2-1). The spaces $X_d, S_d \subset \mathbb{R}^n$ are both discrete.

$$F^{d*}(s) = \max_{x \in X_d} \{\langle x, s \rangle + F^d(x)\}, \qquad \forall s \ \in \ S_d. \tag{2-1}$$

The superscript d, as in F^d, stands for a discrete function with a finite domain, while the subscript d stands for a discrete (finite) set.

For the univariate case, in order to compute the conjugate function $F^{d*}(s)$, the algorithm takes the pairs of points in primal space,

$$\{(x_1, F^d(x_1)), (x_2, F^d(x_2)), ..., (x_n, F^d(x_n))\}, \tag{2-2}$$

and computes a convex hull of the function F^d if it is not convex. This can be done in linear time. Then the following slopes are computed:

$$c_i := \frac{F^d(x_{i+1}) - F^d(x_i)}{x_{i+1} - x_i}. \tag{2-3}$$

Since the function is convex these slopes are in ascending order. Given the points of interest in the dual space $S := \{s_1, s_2, ..., s_m\}$, Lemma 3 of [20] proves that we can compute the conjugate function in the following manner:

(i) If $c_{i-1} < s_i < c_i$, $F^{d*}(s_i) = s_i x_i - F^d(x_i)$

(ii) If $c_i = s_i$, $F^{d*}(s_i) = s_i x_i - F^d(x_i)$ or $F^{d*}(s_i) = s_i x_{i+1} - F^d(x_{i+1})$.

In the case we are dealing with multivariate functions, the discrete Legendre transform reads

$$F^{d*}(s1, s2) = \max_{x_1 \in X_{d1}} \left[s_1 x_1 + \max_{x_2 \in X_{d2}} [s_2 x_2 - F^d(x_1, x_2)] \right], \qquad \forall (s_1, s_2) \in S_{d1} \times S_{d2}$$

,

with primal domain $X_d = X_{d1} \times X_{d2} \subset \mathbb{R}^2$ and dual domain $S_d = S_{d1} \times S_{d2} \subset \mathbb{R}^2$. This procedure is essentially a factorization which casts the original multivariate function as univariate ones and transforms these using the univariate method above. The algorithm for the two dimensional case will involve computing the intermediate function

$$F^d_{s_2}(x_1) = \max_{x_2 \in X_{d2}} [s_2 x_2 - F^d(x_1, x_2)], \qquad \forall s_2 \in S_{d2}, \tag{2-4}$$

and then computing the conjugate function

$$F^{d*}(s_1, s_2) = \max_{x_1 \in X_{d1}} [s_1 x_1 - (-F^d_{s_2}(x_1))], \qquad \forall s_1 \in S_{d1}. \tag{2-5}$$

This procedure can be generalised in higher dimensions.

2-2 Conjugate Dynamic Programming

2-2-1 Original DP framework

In [1], two novel algorithms are developed, in order to solve the DP in the conjugate domain. Below we present the transformation of the original DP problem to its equivalent in the conjugate domain and a short introduction into the two settings in the paper. We start with the discrete system:

$$x_{t+1} = f(x_t, u_t), \quad t = 0, 1, ..., T-1, \tag{2-6}$$

where $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ denotes the dynamics. The system is constrained in the input and state as follows:

$$x_t \in X \subset \mathbb{R}^n, \quad t \in \{0, 1, ..., T\}, \tag{2-7}$$

$$u_t \in U \subset \mathbb{R}^m, \quad t \in \{0, 1, ..., T-1\}. \tag{2-8}$$

The stage and terminal cost functions are $C_t : X \times U \to \bar{\mathbb{R}}$ and $C_T : X \to \mathbb{R}$, where $\bar{\mathbb{R}}$ represents the extension of the real numbers to include $+\infty$. The reason for that is that if the state constraint is violated then the cost for that stage becomes $+\infty$.

The DP formulation for this problem reads:

$$J_t(x_t) = \min_{u_t}\{C(x_t, u_t) + J_{t+1}(x_{t+1}) : (2-6) - (2-8)\}, \quad x_t \in X, \tag{2-9}$$

Solving (2-9) backwards in time, we obtain the optimal cost-to-go functions as well as the control laws that will result in those functions. In order to solve such a problem, we need to iterate over all $x \in X$ and $u \in U$. When dealing with continuous spaces in X and U this procedure is intractable. Thus we need to approximate the solution by discretizing the state and input spaces. The discrete DP (d-DP) formulation reads:

$$J_t^d(x_t) = \min_{u_t}\{C(x_t, u_t) + \bar{J}_{t+1}^d(x_{t+1})\}, \quad x_t \in X_d \subset X, \quad u_t \in U_d \subset U. \tag{2-10}$$

The function $\bar{J}_{t+1}^d$ denotes the extension of the discretized cost-to-go function J_{t+1}^d. Since we are dealing with discrete spaces and only have access to the function values in a finite set of points the extension is needed to compute points outside of the given discrete domain.

For the remainder of this thesis every time we mention d-DP, we are referring to a brute force solution of the DP problem over the input and state spaces. In particular, by having finite discrete spaces, we can iterate over all $x \in X^d$ and $u \in U^d$ in order to find the optimal solution. The discrete spaces employed throughout this thesis are all grid like. This is done in order to take advantage of the speed up in computing the conjugate function using the LLT algorithm (see Remark 5 in [20]).

2-2-2 CDP algorithm

From the original DP formulation, we can create an equivalent problem in the dual domain. In [1], two different settings are presented for solving the DP problem in the conjugate domain. Below we will present the requirements that need to be met so that the appropriate setting can be used. For setting 1 of the CDP algorithm we need:

(i) Input-affine dynamics, $f(x, u) = f_s(x) + f_i(x)u$.

(ii) The sets X and U are compact and convex. Additionally, for every $x \in X$, the set of admissible inputs is non empty.

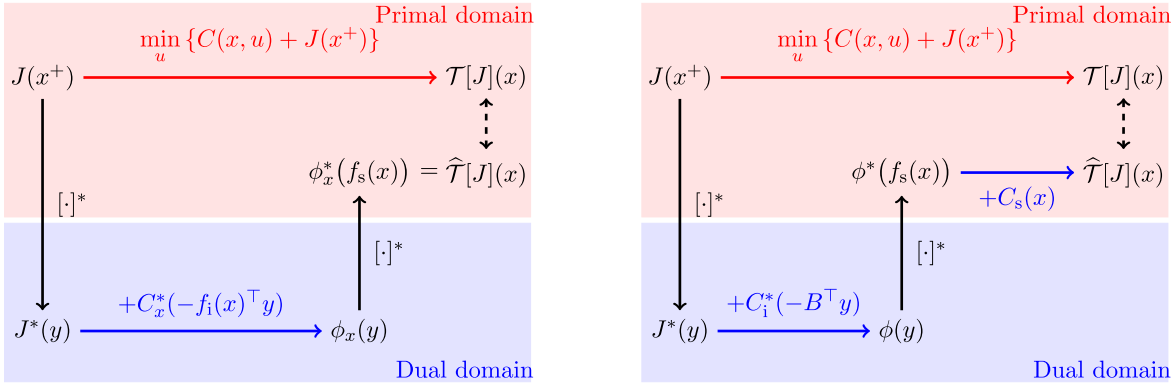
(iii) The stage cost is jointly convex in the state and input variable with a compact effective domain. The terminal cost needs to be also convex.

For the second setting we need the requirements described in setting 1 as well as:

(i) The input dynamics f_i need to be state-independent. That means $f_i(\cdot) = B \in \mathbb{R}^{n \times m}$.

(ii) The stage cost is separable in the state and input variables. That means $C_t(x,u) = C_s(x) + C_i(u), \quad C_s : X \to \mathbb{R} \quad$ and $C_i : U \to \mathbb{R}$.

If a problem meets the above requirements then we can transform the original DP. In Figure 2-1, we present the alternate route of solving the DP problem for the two settings of the CDP algorithm developed in [1].



(a) Setting 1 [1] **(b)** Setting 2 [1]

Figure 2-1: Settings of CDP algorithm [1]

The current Settings of the CDP algorithm compute the cost-to-go functions backwards in time via the alternate path presented in the figure. As it stands, none of the alternative paths compute the optimal control laws for a given step of the control problem. In order to acquire these laws the algorithm solves an additional minimization problem, for each step in the horizon, during the forward phase of the DP problem. The minimization problem is the following:

$$u_t^* = \underset{u_t \in U_d}{\arg\min} \{C(x_t, u_t) + \bar{J}_{t+1}^d(x_{t+1})\}.$$

One of the contributions of the current thesis involves extracting the optimal control laws from the alternate path of the CDP algorithm and thus eliminating the minimization problems of presented above.

Chapter 3

Analytical Results

In this Chapter, we will be presenting the preliminary results regarding the problems tackled in the present thesis. This chapter is structured as follows: firstly, we will explore whether Kuhn's triangulation can create a convex extension out of a convex extensible set of discrete points. The reason why preserving convexity is important, is that the CDP algorithm is essentially blind to non convexity. This means that if the cost-to-go function J_t is non convex, we will be introducing an error by approximating the function by its convex envelope. In the case of a convex cost-to-go function this error disappears. In addition, we will present the proof regarding the extraction of the optimal control input from within the CDP algorithm and present the calculations followed in a one dimensional example.

3-1 Kuhn triangulation as an interpolation method

3-1-1 Method outline and example in 2D

Using Kuhn's triangulation as an interpolation technique was originally presented in [21, 22]. Below we will present an outline of how the method works for completeness. This method is based on splitting the unit hypercube around the query point into simplexes. Then, by finding the simplex that contains our query point, we use a linear combination of the vertices of that simplex. The outline of the method is:

- First we need to translate and scale our coordinate system so that the cube around our query point becomes the unit hypercube. The new coordinates of the query point are $(x'_1, ..., x'_d)$, where d is the number of dimensions of the space.

- Next we must employ a sorting algorithm in order to sort the coordinates of the translated query point. This will be used to identify the simplex in which the point lies.

- From the identified simplex, we use the coordinates of its vertices in order to create linear combinations of the query point's coordinates.

- After finding the coefficients from the previous step, we can use them as weights for the sum of the function values at each vertex.

In Figure 3-1, we present the random query point (x_{1q}, x_{2q}) and the hypercube around it, which in two dimensions is a square.

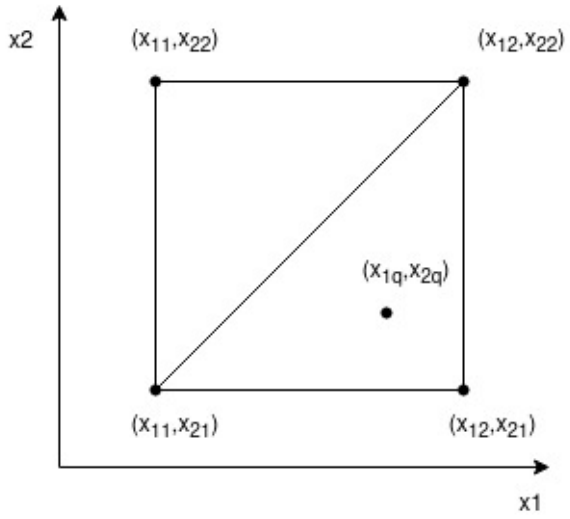


Figure 3-1: Random query point and closest grid hypercube around it.

The first step involves translating and scaling the coordinate system in order for the hypercube around the query point to be the unit hypercube. For any point inside or on the perimeter of the hypercube we can use the following translation.

$$x_{1p\_new} = F(x_{1p}) = (x_{1p} - x_{11})/(x_{12} - x_{11}), \quad x_{11} \leq x_{1p} \leq x_{12} \tag{3-1}$$

$$x_{2p\_new} = G(x_{2p}) = (x_{2p} - x_{21})/(x_{22} - x_{21}), \quad x_{21} \leq x_{2p} \leq x_{22}, \tag{3-2}$$

where p denotes any point inside or on the perimeter of the hypercube. The results of the above procedure lead to the new coordinate system of Figure 3-2.

Using equations (3-1)-(3-2) the new coordinates of our query point become:

$$x_{1q\_new} = (x_{1q} - x_{11})/(x_{12} - x_{11}) \tag{3-3}$$

$$x_{2q\_new} = (x_{2q} - x_{21})/(x_{22} - x_{21}). \tag{3-4}$$

Next, we will divide the hypercube in $d!$ simplexes. The coordinates of the points inside each simplex have to satisfy the following equation.

$$0 \leq x_{r(1)} \leq x_{r(2)} \leq ... \leq x_{r(d)} \leq 1,$$

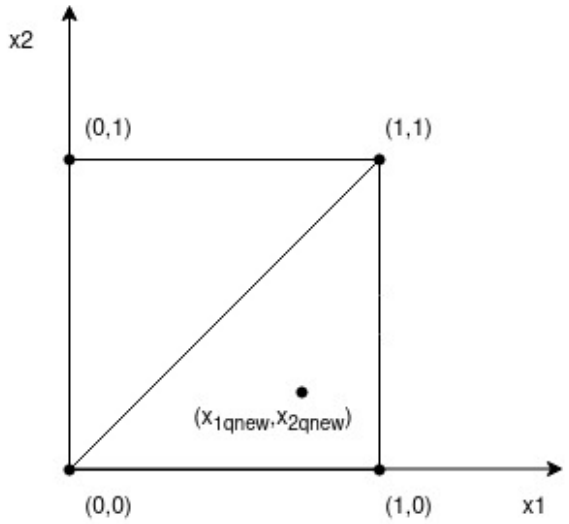


Figure 3-2: Translated coordinate system so that the hypercube around the query point is the unit hypercube

where r represents a possible permutation of $(1, 2, 3, ..., d)$.

In our example, we will separate the square in triangles (2D simplexes) by drawing the diagonal of the square connecting the origin $(0, 0)$ to the following vertex $(1, 1)$. In two dimensions this procedure creates two triangles which can be seen in Figure 3-2. The lower triangle in the figure is described by the equation $x_2 \leq x_1$, while the upper triangle is described by the equation $x_1 \leq x_2$. Next, the procedure requires that we use a sorting algorithm to sort the coordinates of the new query point. The sorting procedure is the most important step of the algorithm. It will allow us to easily identify the simplex inside which the point lies and its corresponding vertices. We know that every simplex inside the hypercube will have two known vertices in $(0, 0, 0, ...)$ and $(1, 1, 1, ...)$. Thus we only need to identify $d-1$ points, where d is the dimension we are working with. An easy way to identify the simplex requires using the dimension on which the sorted coordinates lie. We start with a matrix of size $(d + 1) \times d$ filled with zeros. Then starting from the largest coordinate we fill the appropriate column with 1's for d rows. Then we move onto the next coordinate where we fill the appropriate column with 1 for d-1 rows. The procedure continues to the smallest coordinate where we fill the appropriate column with 1 for one row. Every row of the matrix will represent the coordinates of the vertices of each simplex. The procedure for the 2D case can be seen below. We start from the matrix filled with 0:

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{3-5}$$

Since in our example the coordinate of the first dimension is larger that that of the second we will fill the first column with 1s for the first two rows. This leads to

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}. \tag{3-6}$$

Then we move on to the second largest dimension, and last in our example, x_2. We will fill the second column of the matrix with 1s for one row. The final result can be seen below

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}. \tag{3-7}$$

If the query point lied in the upper triangle, the coordinate of the second dimension would have been bigger and following the same procedure we would first fill the second column with 1s for two rows and then the first column with 1 for one row. Every row of that matrix represents a vertex of the simplex the query point lies inside of. Comparing the matrix in equation (3-7) and the Figure 3-2 we can see that the coordinates of the vertices of the simplex have been identified correctly.

Having acquired the vertices of the simplex, we can now express the coordinates of the query point as a linear combination of the coordinates of each vertex. By solving the following system of equations.

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ x_{1qnew} \\ x_{2qnew} \end{bmatrix}. \tag{3-8}$$

Upon solving the preceding equation we obtain the coefficients a_1, a_2, a_3. Then, we can now compute the function value at the query point as follows:

$$F(x_{1q}, x_{2q}) = a_1 F(x_{11}, x_{21}) + a_2 F(x_{12}, x_{21}) + a_3 F(x_{12}, x_{22}). \tag{3-9}$$

Essentially the function value at the query point is calculated as a linear combination of the function values at the vertices of the simplex, based on the proximity of the query point to the vertices themselves.

3-1-2 Convex extension of a convex extensible function

Definition 3-1.1. *A discrete function $F^d : X_d \to \bar{\mathbb{R}}$ is said to be convex extensible when there exists a continuous convex function $F : \mathbb{R}^n \to \bar{\mathbb{R}}$ where $\forall x \in X_d$ the following holds $F(x) = F^d(x)$.*

The CDP algorithms in settings 1 and 2 both work with discrete functions evaluated at a grid-like discrete set of points. At certain steps, it is necessary to evaluate these functions at points outside of the available points. Thus, we need a interpolation technique. Assuming we start from a convex extensible set of points, we would like this technique to identify a continuous convex function that satisfies the definition of convex extensibility listed above.

The theoretical framework that aims to describe the extension of convexity properties in discrete functions is called Discrete Convex Analysis [23, 24, 25].

Unfortunately, convex extensibility is not a sufficient condition for Kuhn triangulation to lead to a convex extension for a discrete function. We will show that by presenting the following example. Assume the continuous function

$$f(x_1, x_2) = \max(x_1 - 3x_2, -2x_1 + 3x_2). \tag{3-10}$$

The above function is convex as the maximum of two affine functions. In Figure 3-3, we present the function plotted in 3D space.

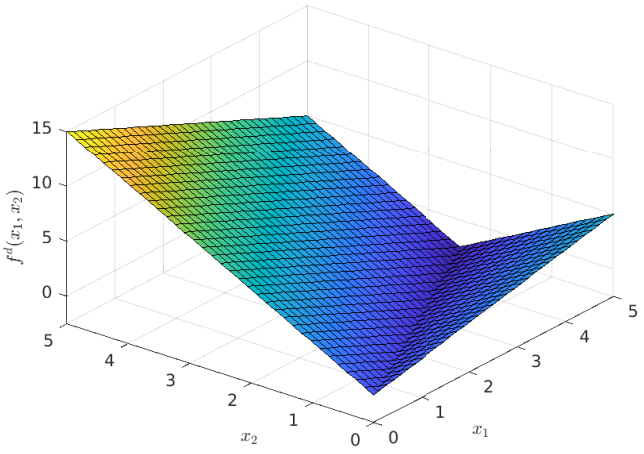


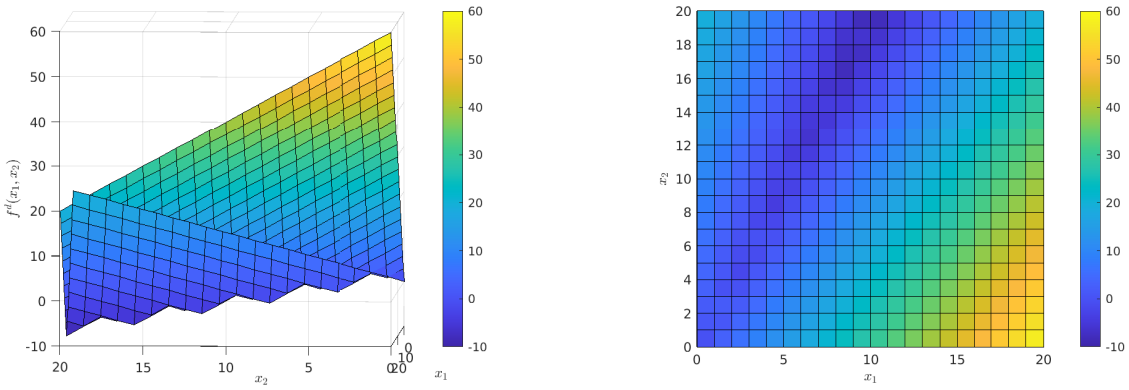
Figure 3-3: Plot of continuous function in 3D space.

We now assume the following discretization of the domain of the function. The function value at these points only, is assumed to be known. The discrete function is as follows:

$$f^d(x_1, x_2) = \max(x_1 - 3x_2, -2x_1 + 3x_2), \quad x_1, x_2 \in X_d^1 = [0, 2, 4, 6, ..., 20]^2. \tag{3-11}$$

We will be using Kuhn triangulation to interpolate the function value at the point $x_q = (x_1, x_2) = (2, 1)$. The continuous function value at this point is $f(2, 1) = -1$. The points around our query point are $x_{ll} = [0, 0]$, $x_{lr} = [2, 0]$, $x_{ul} = [0, 2]$, $x_{ur} = [2, 2]$. We transform these points so that $x_{ll} = [0, 0]$ and $x_{ur} = [2, 2]$ become the opposite corners of the unit square. Now our points are: $x_{ll} = [0, 0]$, $x_{lr} = [1, 0]$, $x_{ul} = [0, 1]$, $x_{ur} = [1, 1]$. Our query point x_q lies in the 2D simplex created by the points $x_{ll} = [0, 0]$,$x_{lr} = [1, 0]$ and $x_{ur} = [1, 1]$. Our point x_q is the following linear combination of the simplex's corners:

$$x_q = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} x_{ll} \\ x_{lr} \\ x_{ur} \end{bmatrix} = \begin{bmatrix} 0 & 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} x_{ll} \\ x_{lr} \\ x_{ur} \end{bmatrix}. \tag{3-12}$$



(a) Angle 1. **(b)** Angle 2.

Figure 3-4: Scatter plot from data interpolated with Kuhn's triangulation.

We then use these coefficients to calculate the value of the function at our query point:

$$\bar{f^d}(x_q) = \begin{bmatrix} 0 & 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} f^d(x_{ll}) \\ f^d(x_{lr}) \\ f^d(x_{ur}) \end{bmatrix} = 0 \cdot f(x_{ll}) + 0.5 \cdot f(x_{lr}) + 0.5 \cdot f(x_{ur}) \Rightarrow$$

$$\bar{f^d}(x_q) = 0 \cdot 0 + 0.5 \cdot 2 + 0.5 \cdot 2 = 2.$$

The notation $\bar{f^d}$ indicates extending the discrete function to points outside its domain.

It is evident that the two values are not the same. Thus the method fails to recover the original convex function. Since there can be more than one convex function fitting a set of convex extensible points we need to investigate this possibility as well. If we use Kuhn triangulation to extend the discrete function of equation (3-11) to the following domain

$$X_d^2 = [0, 1, 2, 3, ..., 20] \times [0, 1, 2, 3, ..., 20]$$

we obtain the result shown in Figure 3-4. The new domain X_d^2 consists of the original points in the domain X_d^1, for which we have the functions values available, and the points on which we used Kuhn triangulation to extend the original discrete function.

What we can observe in Figure 3-4 is that the extended discrete function presents this region where the function values form a non convex area. The method fails because the original discretization is not dense enough to capture the direction of the discretization in every square of the domain. Thus, in the example presented, the true function value of the query point lies below the function values of both points used to compute it. Although the method cannot guarantee the extended function will also be convex extensible, it is important to check whether the method provides a more accurate result when extending a discrete function than the method currently used in the CDP algorithm, namely multilinear interpolation (LERP).

3-2 Extraction of optimal control input

In [1], the proposed algorithms only return the cost-to-go functions but not the optimal control actions that will lead to those cost-to-go functions. For that reason, an additional minimization problem has to be solved in order to acquire the optimal control laws. In order to avoid the extra computational cost of the minimization problem, we will inquire whether optimal control laws can be extracted from within the CDP algorithm. This idea is based on the fact that the LLT algorithm [20] allows us to keep the optimal pairs between the primal and dual domains. This should allow us to find a relationship between the current state x and the optimal input u that will minimize the one step cost function.

3-2-1 Theoretical Framework

Lemma 3-2-1. *For the Settings in the CDP algorithm we can extract the optimal control action by employing the optimal pairs in the conjugation steps as follows.*

$$y^*(x) = \arg\max_y \{\langle y, f_s(x)\rangle - \phi_x(y)\}, \tag{3-13}$$

$$u^*(y) = \arg\max_u \left[\langle -f_i^T(x)y^*, u\rangle - C(x,u)\right]. \tag{3-14}$$

Proof. Following the proof of Appendix B.2.1 of [1], we start from the reformulated DP problem.

We know

$$\phi_x(y) = C_x^*(-f_i^T(x)y) + J^*(y), \tag{3-15}$$

and

$$\hat{\mathcal{T}}[J](x) = \phi^*(f_s(x)) = \max_y \max_u \{\langle y, f_s(x)\rangle - \langle -f_i^T(x)y, u\rangle + C(x,u) - J^*(y)\}.$$

We define

$$L_1(x,u,y) := \langle y, f_s(x)\rangle - \langle -f_i^T(x)y, u\rangle + C(x,u) - J^*(y).$$

Then,

$$\begin{aligned} L_2(x,y) = L_1(x,u^*(y),y) &= \langle y, f_s(x)\rangle - \langle -f_i^T(x)y, u^*\rangle + C(x,u^*) - J^*(y) \\ &\overset{(3-14)}{=} \langle y, f_s(x)\rangle - C_x^*(-f_i^T(x)y) - J^*(y) \\ &\overset{(3-15)}{=} \langle y, f_s(x)\rangle - \phi_x(y). \end{aligned}$$

Finally,

$$\begin{aligned} L_3(x) = L_2(x,y^*(x)) &= \langle y^*, f_s(x)\rangle - \phi_x(y^*) \\ &\overset{(3-13)}{=} \phi^*(f_s(x)). \end{aligned}$$

We have proven that

$$\hat{\mathcal{T}}[J](x) = L_1(x,u^*,y^*),$$

using the mappings in equations (3-13)-(3-14) □

The procedure to obtain the optimal control input from the current state is outlined below

1. From the current state x compute $f_s(x)$.
2. Find the optimal dual variable y from $f_s(x)$, using eq. (3-13).
3. Compute $-f_i^T(x)y$.
4. Find the optimal input variable u from $-f_i^T(x)y$, using eq. (3-14).

3-2-2 Example of SISO system

Below we will present the procedure outlined above in a SISO system that can be solved analytically. The time horizon will be one step ($T = 1$). The procedure can be extended to longer than one step horizons. In Chapter 4, we will also present results from simulations in more complex systems.

We assume the following system

$$x_{t+1} = 2x_t + u_t \tag{3-16}$$
$$C_t(x_t, u_t) = x_t^2 + u_t^2 \tag{3-17}$$
$$C_T = x_T^2. \tag{3-18}$$

For this system, we first compute the conjugate of the stage cost with respect to the input variable u as follows.

$$C^*(y) = \max_u\{\langle y, u\rangle - C(x, u)\} = \max_u\{yu - x^2 - (u)^2\} = \frac{y^2}{4} - x^2.$$

Next we compute the conjugate of the cost-to-go function at the terminal step which is the terminal cost C_T. Since we are only solving for one step, from here onwards, we will not use the subscript t to indicate the step of the horizon.

$$\begin{aligned} J^*(y) = C_T^*(y) &= \max_x\{\langle y, x\rangle - C_T(x)\} \\ &= \max_x\{yx - x^2\} = \frac{y^2}{4}. \end{aligned}$$

We can now form the function in (3-15) and then compute the optimal pairs between the primal state and the dual state variables.

$$\begin{aligned} y^*(x) &= \arg\max_y\{\langle f_s(x), y\rangle - C_x^*(-f_i^T(x)y) - J^*(y)\} \\ &= \arg\max_y\{2xy - \frac{y^2}{4} + x^2 - \frac{y^2}{4}\} \\ &= 2x. \end{aligned} \tag{3-19}$$

Next, we will compute the optimal pair between the input variable u and the dual state variable y.

$$\begin{aligned} u^*(y) &= \arg\max_u \{\langle -f_i^T(x)y, u\rangle - C(x,u)\} \\ &= \arg\max_u \{-yu - x^2 - u^2\} \\ &= -\frac{y}{2}. \end{aligned} \tag{3-20}$$

By combining the optimal pairs acquired in (3-19) and (3-20), we end up with the optimal control law

$$u^*(x) = u^*(y(x)) = \frac{-y(x)}{2} = -x. \tag{3-21}$$

On the other hand, by solving the original DP formulation analytically we would get

$$\begin{aligned} u^*(x) &= \arg\min_u \{C(x,u) + J(x_T)\} \\ &= \arg\min_u \{x^2 + u^2 + x_T^2\} \\ &= \arg\min_u \{x^2 + u^2 + (2x+u)^2\} \\ &= \arg\min_u \{x^2 + u^2 + 4x^2 + u^2 + 4xu\} \\ &= \arg\min_u \{5x^2 + 2u^2 + 4xu\} \\ &= -x. \end{aligned} \tag{3-22}$$

As we can see, the optimal control law obtained in both cases is exactly the same. A few further remarks regarding the proposed way of extracting the optimal control input. The analytical procedure outlined above is being computed with all spaces being continuous. That means that whatever the optimal pairings are, we know that the space contains them and we can extract them without error. The conjugations in the CDP algorithm are performed over discrete spaces with the LLT algorithm. As input to the LLT algorithm, we need to provide both the primal and dual grids. Since we have no way of knowing the optimal pairing between the two grids beforehand, the choice of the two grids will affect the result of the conjugation. The difference this will have in the real world simulations will be examined in Chapter 4.

Chapter 4

Simulation Results

In this chapter, we will present the simulation results regarding the various topics tackled in this thesis. Firstly, we will present the difference between using Kuhn triangulation as an interpolation technique and multilinear interpolation (LERP). Then, we will use the theory presented in the previous chapter to extract the optimal control input from within the CDP algorithm.

4-1 Kuhn triangulation versus LERP

In this section, we will present the results of using Kuhn triangulation versus LERP as an interpolation technique. The interpolation techniques will be tested on line 6 of Setting 2 of the CDP algorithm in order to calculate $\bar{\phi}^{d*d}$ [1, p. 19]. For the reader's convenience the algorithms of [1] are included in the appendix of the current thesis. For both techniques we will calculate the optimal cost-to-go functions J_t, that will be used to extract the optimal control inputs via a minimization. We will calculate the cost-to-go function for a horizon of $T = 10$ for the following system.

$$x_{t+1} = \begin{bmatrix} -0.5 & 2 \\ 1 & 3 \end{bmatrix} x_t + \begin{bmatrix} 1 & 0.5 \\ 1 & 1 \end{bmatrix} u_t \tag{4-1}$$

$$C_t(x_t, u_t) = ||x_t||_2^2 + e^{|u_{t1}|} + e^{|u_{t2}|} - 2 \tag{4-2}$$

$$C_T = ||x_T||_2^2 \tag{4-3}$$

The system is constrained as follows

$$x_t \in [-1, 1]^2 \tag{4-4}$$

$$u_t \in [-2, 2]^2 \tag{4-5}$$

For our testing, the system is discretized in 11, 21 and 41 points in each dimension of the state and input grids. In Table 4-1, we will present the performance of the two interpolation techniques with respect to the overall cost of the trajectory that was generated from the control inputs obtained from the cost-to-go functions. These control actions were generated for a total of 100 initial conditions. In more detail, we present the interpolation technique used, the discretization of the state and input grid, the percentage of cases each interpolation technique outperforms the other, and the average cost improvement in those cases.

Table 4-1: Comparison between Kuhn triangulation and LERP as interpolation techniques.

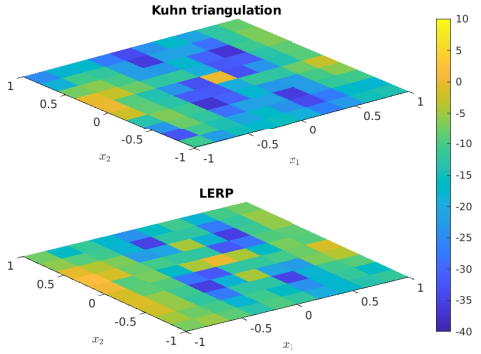
Method & Discretization	C.I w.r.t other method (% of cases)[1]	Avg Cost Improvement
Kuhn triangulation - 11	17	0.5080
LERP - 11	17	0.7140
Kuhn triangulation - 21	15	0.3097
LERP - 21	17	0.2959
Kuhn triangulation - 41	11	0.0840
LERP - 41	15	0.1461

As we can see each, interpolation technique outperforms the other in about 15% of the cases and that number does not change considerably when increasing the discretization. Furthermore, the average cost improvement ranges from $0.08 - 0.71$ which translates to a percentage improvement of $1.5 - 8\%$. This improvement, although important in the sparser grids, does not translate to any algorithm being the superior over the other. In about two thirds of the cases the algorithms performed exactly the same, which can be attributed to the sparse discretisation of the input grid in the sparser cases. The two methods could be computing different values for the control input but since the discretization is not dense enough, the difference is lost in the end result. In the more dense cases, the dense descritization of the state grid is leading to smaller errors and thus the methods perform similarly. Another reason why the two algorithms perform similarly is that when the query point, we are trying to compute the function value for, lies on or very close to the grid lines, then Kuhn's triangulation becomes the same as linear interpolation.

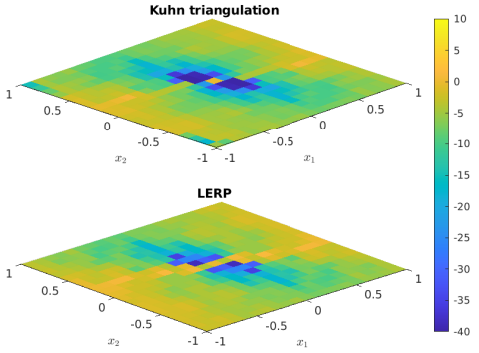
For the sake of completeness in Figure 4-1, we present the error of the cost-to-go functions obtained from the two interpolation methods versus our benchmark cost-to-go function for different discretizations for the first step in the horizon. The benchmark cost-to-go function was computed using a high density d-DP solution. As is evident from the figure, both methods perform very similarly to each other, which supports the comparison presented in Table 4-1. Also, as the discretization becomes more dense, the methods approach the benchmark which is something to be expected.

Overall, it is evident that the two methods perform very close to one another, and any small advantage one has over the other could be due to the different cost-to-go functions and not a clear advantage of one method.

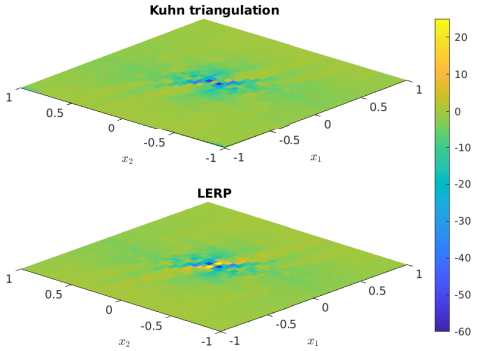
[1] This column depicts the percentage of cases one method outperforms the other for the same discretization.



(a) Discretization of 11 points.



(b) Discretization of 21 points.



(c) Discretization of 41 points.

Figure 4-1: Error between cost-to-go functions calculated with interpolation methods and the benchmark.

4-2 Extraction of optimal control input

As we mentioned in Chapter 3, although theoretically the extraction of the optimal control input can be achieved, there are factors that could influence the result like the choice of the dual state grid. In Chapter 3, we extracted the optimal pairs analytically in the continuous domain. Since the CDP algorithm is working in the discrete domain and since we need to specify both the primal and dual grids before the conjugation, it is likely that the theoretical optimal pairs are not contained in the specified grids. In that case, the LLT algorithm will return a set of sub-optimal optimizers between the primal and dual grid. These pairs will then introduce an error in the control action we will compute. In the current version of the CDP algorithm the control inputs are computed by solving a minimization problem during the forward iteration of the DP problem. In the next sections, we will compare these with the control inputs obtained from our proposed method.

4-2-1 Effects of discretization on extraction of optimal pairs

As we mentioned already, extracting the optimal pairs between the primal and dual domain during the conjugation could be affected by the chosen grids. For our example, we will use the following function to perform the conjugation on.

$$f(x_1, x_2) = x_1^2 + x_2^2, \quad x_1, x_2 \in \mathbb{R}, \tag{4-6}$$

for which we can analytically compute the conjugate function

$$f^*(s_1, s_2) = \frac{s_1^2}{4} + \frac{s_2^2}{4}, \tag{4-7}$$

with the optimizer between the primal and dual domains being

$$x_1^*(s_1) = \arg\max_{x_1}\{\langle x_1, s_1\rangle + f_{x_2}(x_1)\} = \frac{s_1}{2}, \tag{4-8}$$

$$x_2^*(s_2) = \arg\max_{x_2}\{\langle x_2, s_2\rangle + f_{x_1}(x_2)\} = \frac{s_2}{2}. \tag{4-9}$$

Now consider the following discretization of the function above.

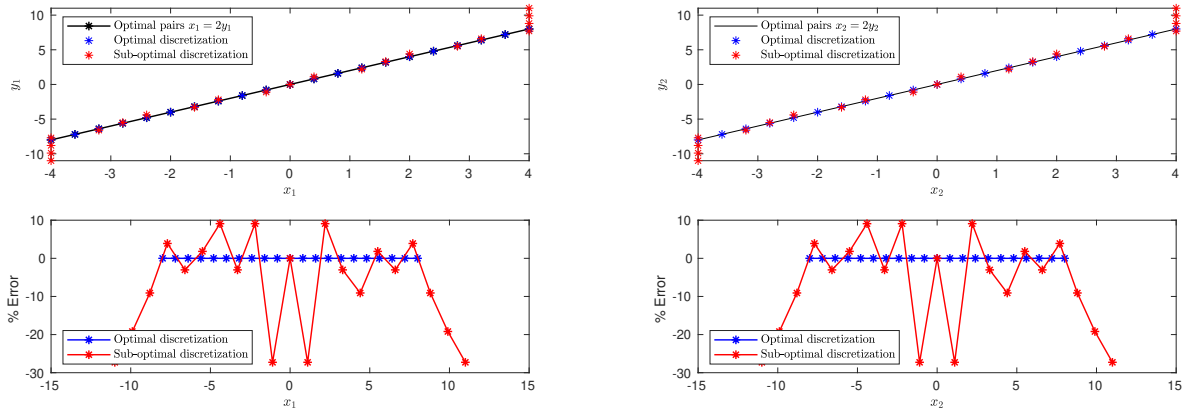
$$f^d(x_1, x_2) = x_1^2 + x_2^2, \tag{4-10}$$

$$(x_1, x_2) \in \{-4, -3.6, -3.2, -2.8, ..., 2.8, 3.2, 3.6, 4\}^2. \tag{4-11}$$

In order to perform the conjugation via the LLT, we will choose two different grids for the dual domain.

$$(s_1, s_2) \in S_{opt} = \{-8, -7.2, -6.4, ..., 6.4, 7.2, 8\}^2, \tag{4-12}$$

$$(s_1, s_2) \in S_{subopt} = \{-11, -9.9, -8.8, ..., 8.8, 9.9, 11\}^2. \tag{4-13}$$

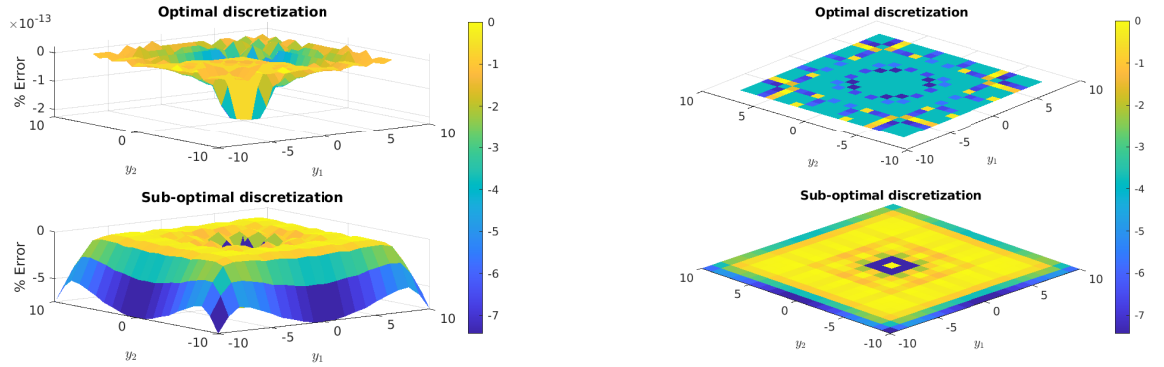


(a) First dimension. **(b)** Second dimension.

Figure 4-2: Optimal pairs for the two discretizations and the analytical solution (top) and percentage of error each pair for the two discretizations differs from the analytical solution (bottom).

All the chosen grids contain 21 equidistant points in every dimension. The reason we chose the two above discretizations of the dual domain is that the first one is the optimal discretization based on the analytical result in equations (4-8) and (4-9), while the second contains a larger space that given the same number of points should not contain all the optimal pairs. In Figure 4-2, we plot the pairs obtained from each discretization on top of the optimizer mappings we obtained in equations (4-8)-(4-9), as well as the percentage each pair differs from its analytically calculated optimal. Given that we calculate the error as a percentage of the optimal value, we must note that in the case of the pair being $(0, 0)$, both times the algorithm has identified the pair correctly and we set the error manually to zero. As we can observe from the figure, when the dual grid is chosen to be exactly the optimal of the primal grid, LLT computes the optimal pairs without error. In the case of the sub-optimal grid we can see that the error in the sub-optimal pairs can reach 30% of the optimal value. In our case this can lead to sub-optimal control inputs that will increase the overall cost for a trajectory. The LLT algorithm, in higher than one dimensions, employs a factorization scheme (see Section 2.1). This means that in order to extract the control input in the second dimension, we need the pair from the first dimension. This could introduce an additional error in the control inputs of the dimensions following the first. A higher discretization of the space could potentially lead to better results, since a denser grid is more likely to contain the points needed for the optimal pairs. Given that we are using the LLT algorithm to solve a DP problem, we must consider the curse of dimensionality. A dense grid in high dimensions could lead to large time requirements for computing the solution, or even make the problem intractable. To summarize, the first important result that we should keep in mind is that the choice of the dual grid can greatly affect our ability to extract the optimal pairs.

In the CDP algorithm's current state, the optimal pairs of the conjugations are not employed. In every call to the LLT algorithm only the conjugate function is needed, in order to compute the cost-to-go functions. These functions are later used to extract the control inputs via a minimization problem. It is important to see how a sub-optimal discretization can affect the values of the conjugate function, since the function is used as an alternative to extract the optimal control inputs. In Figure 4-3, we plot how much the conjugate functions differ from the analytically available result in equation (4-7) for the two discretization mentioned above.



(a) Side view of the error.

(b) Top view of the error.

Figure 4-3: Percentage of error of conjugate function computed with LLT with respect to the analytically available conjugate function.

Again, as these Figures represent the percentage of error, in the case of $(0, 0)$, the function value was correctly identified and thus the error was set to zero.

The function values when the discretization is optimal, match the ones from the analytical result. For the sub-optimal discretization we can see that the function values can be off by up to 8%. This result means, that in this case, and for the same sub-optimal choice for the dual grid, the conjugate function is closer to the analytically available conjugate, than the conjugation optimizers are to the analytically available optimizer. Since the current method of extracting the control inputs in the CDP algorithm, employs only the conjugate functions, it could be that it will lead to a smaller error compared to our proposed method which employs the conjugate pairs.

This will be tested also, in the following section on two different systems, a 1D and a 2D.

4-2-2 1D example

Firstly, we will apply the procedure to the model described below. The dynamics is given by:

$$x_{t+1} = 2x_t + u_t,$$

The system is constrained as follows:

$$-1 \leq x_t \leq 1, \tag{4-14}$$
$$-2 \leq u_t \leq 2. \tag{4-15}$$

The stage and terminal costs are:

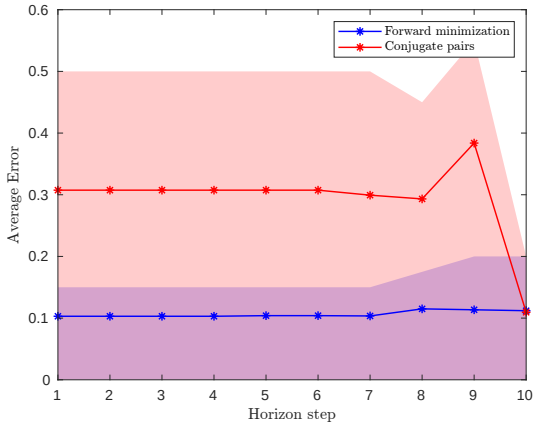
$$C_t = x_t^2 + e^{|u_t|}, \tag{4-16}$$
$$C_T = x_T^2. \tag{4-17}$$

We will solve the DP problem and obtain the optimal control inputs via 3 different methods for a time horizon of $T = 10$. First is the procedure using the optimal conjugation pairs described in the previous Chapter. Then the procedure that the CDP currently uses, which is the minimization of the cost-to-go function during the forward iteration of the DP problem. Lastly, for our benchmark we will use the d-DP solution from a high density discretized grid (81 points in each dimension). .

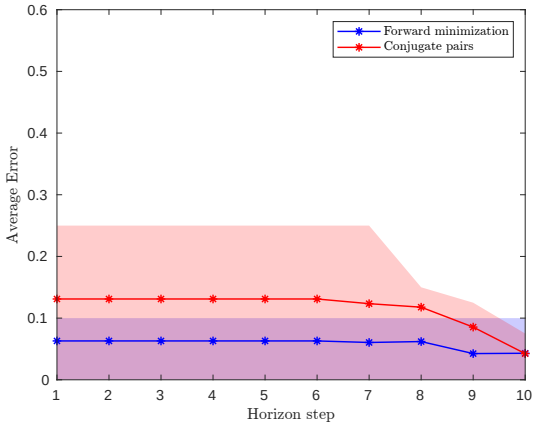
In Figure 4-4, we present for each step in the horizon the absolute error between the control input obtained from the two methods and the benchmark, averaged out over 100 initial conditions. For the sake of comparing the two methods fairly, at each step we start from the same initial condition. That means that the starting state in each step is the same for all methods. At each step we assume that x_t is the same for both methods and we calculate $u^*(x_t)$ with the proposed method, the forward minimization and the d-DP solution. The areas filled with color depict the area where 80% of the samples lie. The initial conditions were selected randomly from a uniform distribution over the interval $[-1, 1]$. The reason we used a uniform distribution is that we wanted to include as much of the state space as possible so as the performance of the methods is not influenced by parts of the state space where one method could outperform the other. The subfigures present how the error changes for different discretizations of the input and state space. The proposed method could not control all the initial conditions at each step, so in the figure we present only those conditions that were able to be controlled. In Figure 4-5, we present the number of initial conditions that could not be controlled at each step. Note that we add the initial conditions that were not controlled at the previous steps to the current one.

As we can see, in the Figure 4-4 the methods perform extremely well. As we increase the discretization of the state and input spaces both errors are decreasing reaching very satisfactory levels. The difference between the errors seems to be decreasing with a higher discretization, which should be expected with more dense grids. Since we start from the same initial conditions spanning the entire state space x we can see that both methods perform similarly at each step. This is expected as for the first few steps the cost-to-go functions are the same for each method, which should in turn give us the same results for the control inputs.

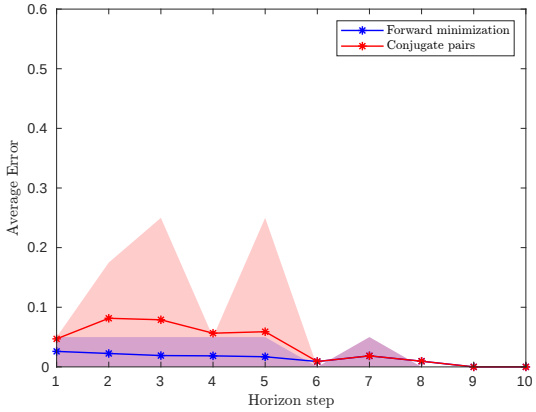
Since we are working with a time horizon of $T = 10$, it is important to check how the methods perform over the entire horizon and how sub-optimal control actions computed by each method can affect the next steps of the horizon. For that reason, in Figure 4-6, we present the cost-to-go at every step when using the control actions we obtained above from each method and the benchmark. The costs are again averaged over the 100 initial conditions and at every step the cost is accumulated over the previous time steps. The filled areas represent the area inside which 80% of the our costs for the current step lie. The proposed method was not able to control all initial conditions, with a number of trajectories breaking the constraints. For that reason in the Figures 4-6 and 4-7 we have only included the trajectories that did not violate the constraints. In Figure 4-8, we present the number of invalid trajectories, for the proposed method, at each step of the horizon for different discretizations. Note that at each step we accumulate the number of trajectories over time. As we can see in the sparse discretization the proposed method cannot control all trajectories. The reason being that the extraction of the optimal conjugation pairs with that discretization of the state and input spaces, is not possible, leading to higher errors. Both the forward minimization method and the benchmark were able to control all trajectories without violating the constraints.



(a) Discretization of 11 points.



(b) Discretization of 21 points.



(c) Discretization of 41 points.

Figure 4-4: Absolute error of one step ahead optimal control inputs against the benchmark averaged over 100 initial conditions.

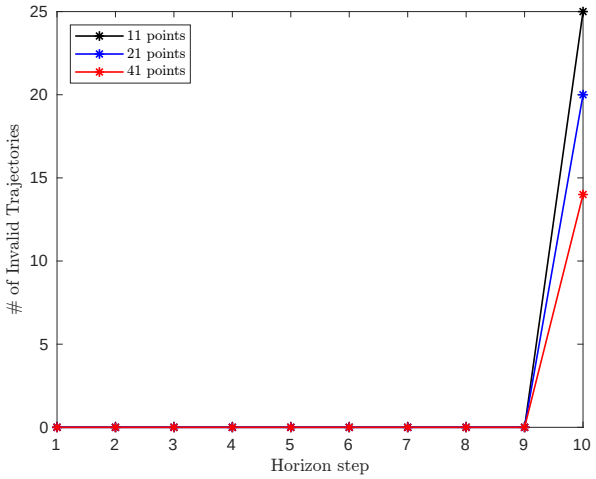
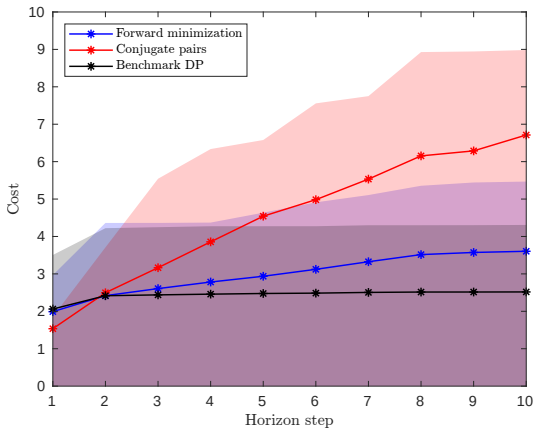


Figure 4-5: Number of invalid trajectories at each step in the horizon for the one step ahead predictions.

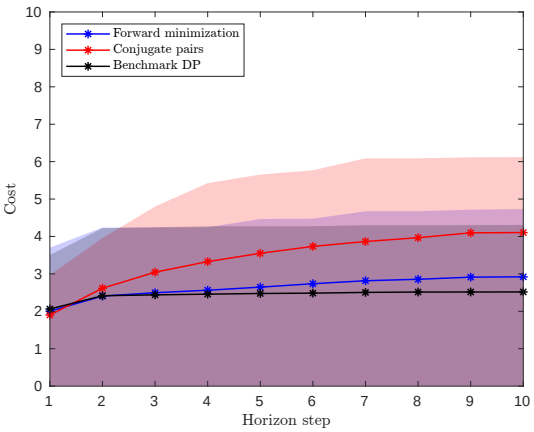
In Figure 4-6, the forward minimization method outperforms the conjugation pairs by a considerable amount. The difference between the two decreases as the grid becomes more dense. The difference between the two methods can be attributed to a snowballing effect of the errors at each step. As the proposed method computes slighty worse control inputs at each step, new sub-optimal states will be reached by the system. These sub-optimal states will then lead to larger inputs in order to be controlled and thus a propagation effect is created where each choice of control input affects the cost at later stages, since its effect needs to be mitigated. That is the reason why the cost in the case of the conjugate pairs keeps increasing, or takes longer to reach a steady state than the other two cases.

Lastly, we present in Figure 4-7 the absolute error of the trajectory of the system when controlled by the actions obtained by each method versus the benchmark averaged over the 100 initial conditions. Even though the snowballing effect described above will heavily influence the following figure, we think its important to include it since these are the "optimal" trajectories the methods were able to compute, for the given horizon. The error is indicative of the methods' multi-step performance and thus an important result in the comparison between the two methods.

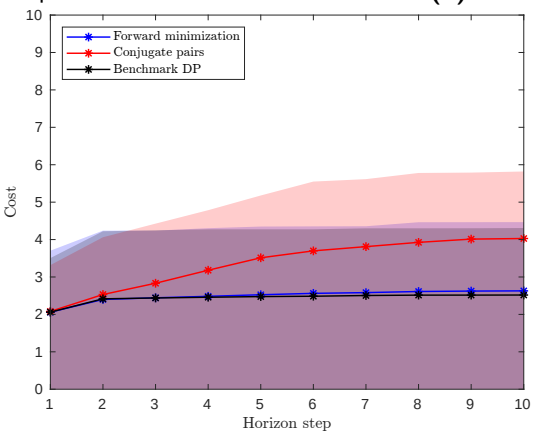
In the figure, we observe a similar relation to the previous plots, where the forward minimization control inputs outperform the control inputs from the conjugate pairs, and the difference becomes smaller as we increase the discretization. There is one more interesting aspect we need to discuss in these plots. In the last step of the horizon, the error of the trajectories is increasing. This could be explained because as we get closer to the horizon and the state remains within the constraints, it is no longer optimal to incur large control inputs on the system, as it would lead to higher costs overall. So the system is left to drift, in a sense. Given the dynamics being unstable, without control the sub-optimal state computed by the two methods will drift faster than the optimal state computed by the benchmark. This lead to the increase of the overall error. This can also be backed up by the results in Figure 4-8 where only at the last step we have uncontrollable states.



(a) Discretization of 11 points.

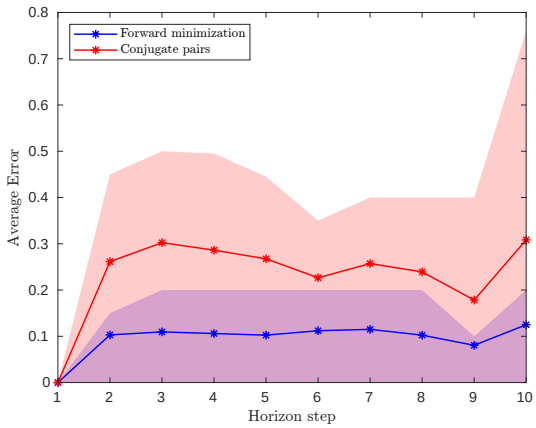


(b) Discretization of 21 points.

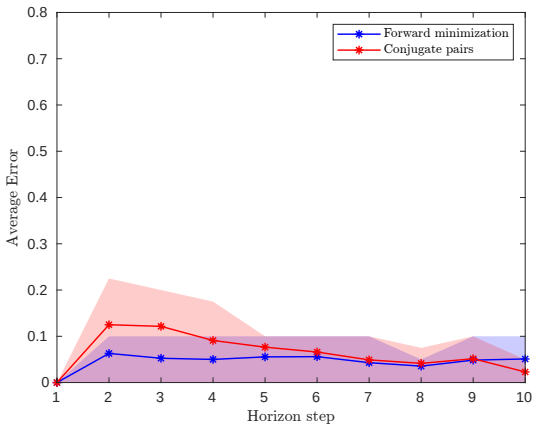


(c) Discretization of 41 points.

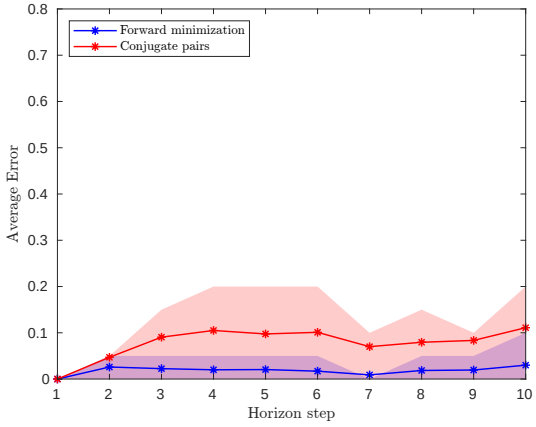
Figure 4-6: Average aggregate cost at each step for the control inputs calculated by the 3 different methods.



(a) Discretization of 11 points.



(b) Discretization of 21 points.



(c) Discretization of 41 points.

Figure 4-7: Trajectory error for each step averaged over 100 initial positions.

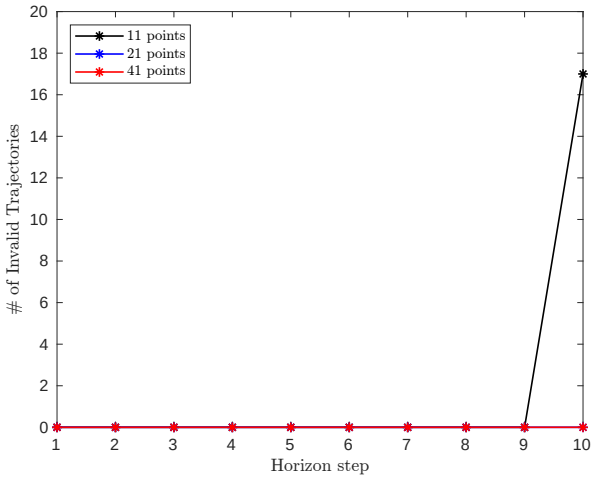


Figure 4-8: Number of invalid trajectories at each step in the horizon.

4-2-3 2D example

In this section, we will be presenting the same plots for a 2-state, 2-input system. The system is described by the Equations (4-1)-(4-5).

We will be solving for a finite time horizon of $T = 10$. Again, the control actions are computed with the the proposed method, the current method implemented in the CDP algorithm and our benchmark, which is a high density d-DP solution. We will test the performance of the methods against each other and the benchmark. The results are averaged over 100 initial conditions. The proposed method was not able to control the system for each initial condition. In Figure 4-9, we present the accumulating number of trajectories that violated the constraints for different discretizations. The reason why the method failed is most likely that the chosen grids introduced an error big enough in part of the space, that the optimal control inputs could not be identified anymore. This can be backed up by the fact that as the discretization becomes more dense the number of uncontrollable trajectories diminish. Also important that again at the last step the number of uncontrollable trajectories increase which could be explained by the fact that the system does not want to incur large control inputs, leading to some uncontrollable states. Compared to the 1D case, where we had fewer uncontrollable trajectories in the one step ahead predictions, it is important to note, that as the systems become more complex, a higher discretization is needed for the proposed method to be able to perform well. Both the forward minimization method and our benchmark were able to control all trajectories for all discretizations.

In Figure 4-10, we present the absolute error between the two methods and our benchmark, for the one step ahead control input, averaged over the initial conditions. We assume at each step the starting state to be the optimal one, so that we compare the two methods fairly. The highlighted regions represent the space in which 80% of the data points for each step lie.

Similarly to the 1D example the proposed method performs worse than the current one and the difference becomes smaller as we increase the discretization. Even in the denser grids, the proposed method performs two times worse than the current method. This can be attributed to the larger error of extracting the conjugate pairs compared to the cojugate function that was discussed in a previous section.

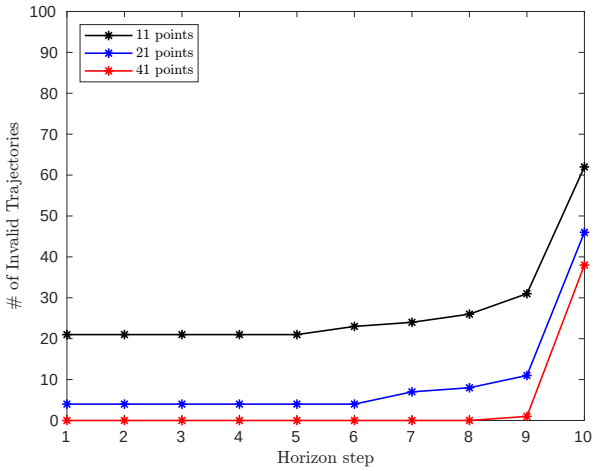
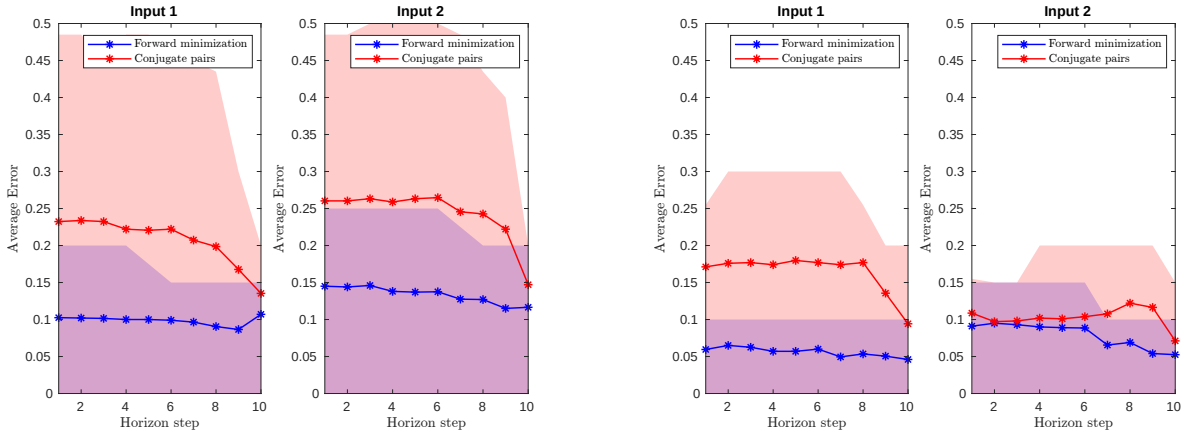


Figure 4-9: Number of invalid trajectories at each step in the horizon.

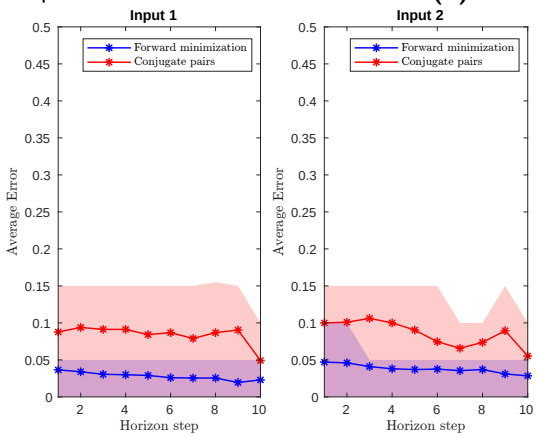
As is the case in the 1D system, we will also present the methods' performance over the entire time horizon. Again the proposed method failed to control all of the trajectories without violating the constraints. In Figure 4-11, we present the accumulated number of trajectories that broke the constraint, at each step, for different discretizations. For the smallest discretization of 11 points, 90% of the total initial conditions could not be controlled by the algorithm. As the discretization increases, the number of invalid trajectories declines steeply. This result clearly emphasizes the need for increasing the discretization of the grid, or finding a better way of choosing the dual grid during the conjugation. For the remaining plots, we will be including only the valid trajectories for the proposed method. Both the forward minimization method and the benchmark were able to control all initial conditions without violating the constraint.

In Figure 4-12, we present the accumulating cost of the three methods at each step averaged over the initial conditions. Additionally, in Figure 4-13, we present the absolute error of the corresponding trajectories resulting from applying the control inputs we calculated from each method. The results of these plots, are mostly unchanged from what we observed in the 1D example. Two things need to be mentioned. The error of the trajectory seems to be increasing for both methods early in the horizon. For more complex systems, we can observe that the effect of sub-optimal control inputs creates a larger snowballing effect in the states as the trajectories drift further from the optimal one earlier than the 1D case. The second observation is the performance of the proposed method in the smallest discretization. In Figure 4-13a, even when including only the trajectories that didn't violate the constraints, the method barely manages to control them. The error for the method is hovering around 0.25 which is enormous given that the state space spans the interval $[-1, 1]$. The much bigger errors, result in high control inputs in an effort to control the system which results to the almost 5 times bigger average total cost of the trajectories over the other methods. The sparse discretization coupled with a sub-optimal choice for the dual grid makes the proposed method essentially impractical.



(a) Discretization of 11 points.

(b) Discretization of 21 points.



(c) Discretization of 41 points.

Figure 4-10: Absolute error of one step ahead optimal control inputs against the benchmark averaged over 100 initial conditions.

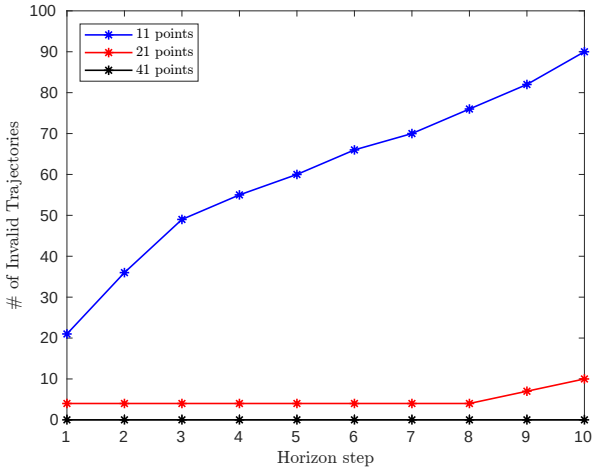
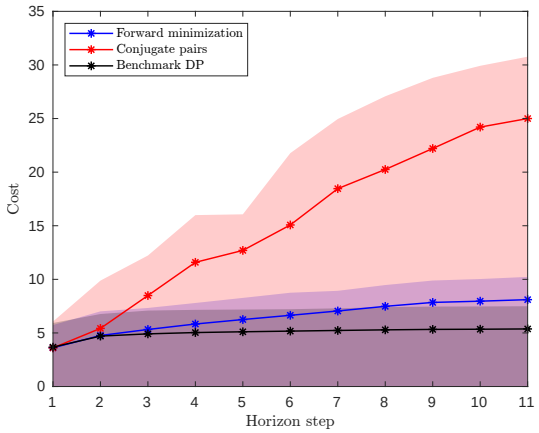
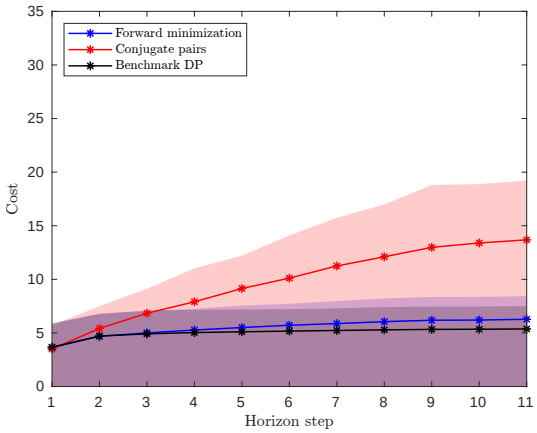


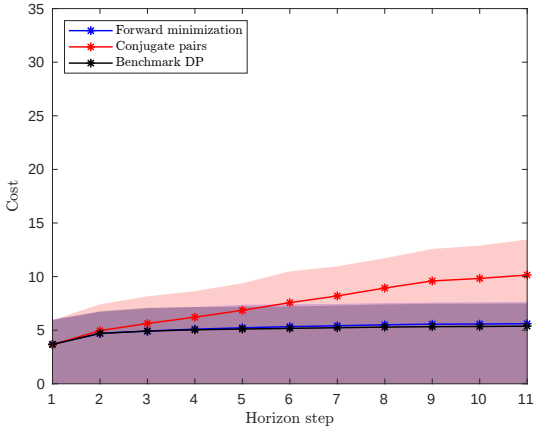
Figure 4-11: Number of invalid trajectories at each step in the horizon for the multistep implementation.



(a) Discretization of 11 points.

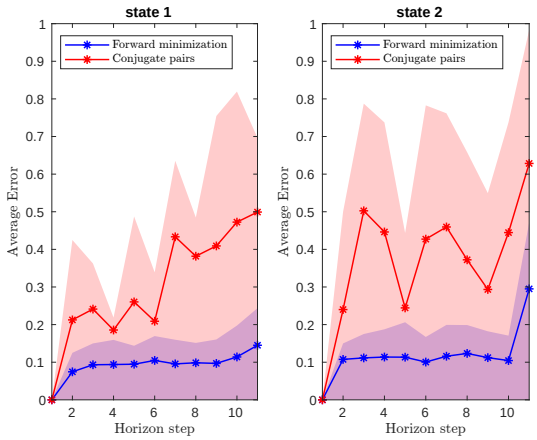


(b) Discretization of 21 points.

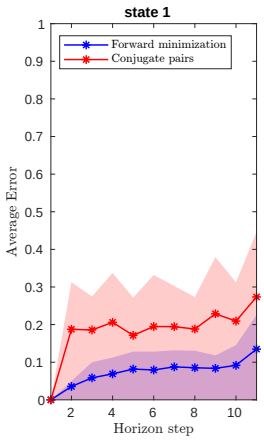


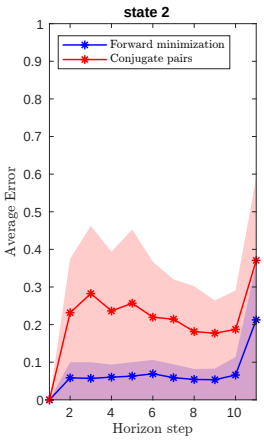
(c) Discretization of 41 points.

Figure 4-12: Average aggregate cost at each step for the control inputs calculated by the 3 different methods.

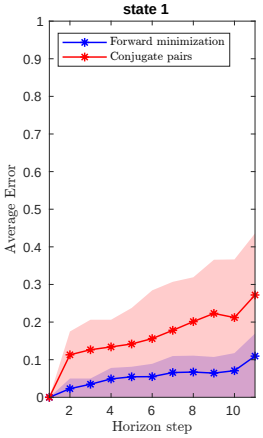


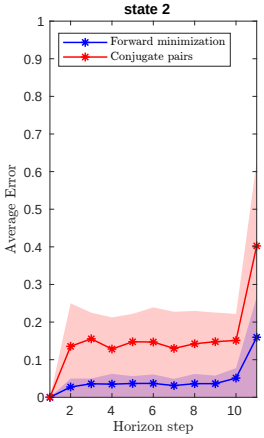
(a) Discretization of 11 points.





(b) Discretization of 21 points.





(c) Discretization of 41 points.

Figure 4-13: Trajectory error for each step averaged over 100 initial positions.

Chapter 5

Conclusions and Future Research

In this final chapter of the thesis, we will shortly discuss the work conducted and shortcomings of the proposed methods as well as try to provide any willing reader with a few research directions in order to improve the overall CDP algorithm.

5-1 Conclusions

In this thesis, several improvements of the algorithms proposed in [1] were developed. Starting, we considered the application of Kuhn's triangulation as an interpolation technique. The goal here was twofold: First, check whether we can preserve convexity in a convex extensible set of discrete points. We proved that it is not possible to do that every time. Second, we tested the method against the current interpolation method used by CDP, LERP, in order to see whether one method outperforms the other. A more accurate representation of the cost-to-go function will lead to smaller errors in the optimal control input and thus smaller costs for the trajectory of the system. In our testing, the two methods performed similarly with one outperforming the other in about 15% of the total cases. The difference between the two methods became smaller as the grid we used became more dense. Overall, the use of a different interpolation technique will provide the algorithm with more versatility as the user can choose between the two and use the best performing one for their specific use case.

Lastly, we considered the extraction of the optimal control input from within the CDP algorithm. To achieve that, we employed the conjugation's optimal pairs in order to create the optimal control law $u = u^*(y^*(x))$, where our optimal control input u is a function of the dual variable y, which in turn is a function of our current state x. We showed that extraction of the optimal control input is indeed feasible when dealing with continuous spaces, but depends highly on the discretization of the dual grid, when the spaces we are working with are discrete. Since we need to choose the dual grid before actually performing the conjugation, the actual optimal pairs may not be within that grid and thus we introduce an error creating a sub-optimal set of pairs. We showed in our example that for the same discretization of the dual grid, the error introduced by the sub-optimal pairs is larger than the error of the conjugate

function itself. This is extremely important since the conjugate functions computed in [1] are currently used in order to extract the optimal control sequence. So for sub-optimal grids the current method employed by the CDP should outperform the proposed method. Another important observation is that for the same grid size, the error of the dual pairs is much larger outside the border of the optimal dual grid than inside it. This can be explained since every point outside the border of the optimal dual grid gets assigned to the same point in the primal grid, thus allowing the error to increase the further away we go. By having a larger dual grid than required the points outside the optimal dual grid are not useful and thus we make the effective region of the space more sparse, resulting in worse results when extracting the optimizer mappings. In our simulations, the same behaviour was again evident. The proposed method performed slighty worse than the current method for the one step ahead predictions of the control input. For the entire horizon the cost of the proposed method was much worse due to the accumulating error of the multistep implementation. By increasing the density of the grid, we observed that the both methods performed better and also closer to each other. Since both methods can theoretically extract the optimal control sequence, by increasing the grid closer to its continuous limit, the error between the two methods should shrink. To sum up, as it currently stands, the proposed method should not be used in the CDP algorithm, since the error it introduces is larger than the current method's, especially in smaller grids. Improving the selection of the dual grids during the conjugation steps of the algorithm could lead to significant improvements which could make the proposed method more advantageous, should that happen.

5-2 Future Research

Several improvements could be carried out in the works presented in the current thesis. These suggestions are pertaining to the work conducted during this thesis. For further research ideas one could read the relevant section of the original CDP paper.

Identification of system matrices. One possible research idea to extend the capabilities of the original algorithms is to introduce an identification technique that is able to generate a model. Since the original algorithms are model based, by employing an identification technique we can solve a larger number of problems, not necessarily model based. Especially important is the identification when dealing with noisy data, as the performance of the algorithm depends on the accuracy of the model.

Preserving Convexity. As neither the current method, LERP, nor the proposed one, Kuhn triangulation, can preserve convexity when extending a convex extensible set of points, further research is required in finding a computationally efficient interpolation method that could guarantee that.

Choosing the dual grid. As mentioned above the choice of the dual grid greatly impacts the error of the proposed method for extracting the optimal control input. By choosing more fitting grids the proposed method could result in significantly lower errors, which could make the method more attractive than the currently used one since its computational load is effectively non existent.

Appendix A

CDP Algorithms

In this appendix we present the CDP algorithms that were developed in [1].

Setting 1

Algorithm 1 Implementation of the d-CDP operator for Setting 1 [1]

Input: dynamics $F_s : \mathbb{R}^n \to \mathbb{R}^n, F_i : \mathbb{R}^n \to \mathbb{R}^{nxm}$; cost-to-go (at t+1) $J : \mathbb{X}_g w R$; conjugate of stage cost $C_x^* : \mathbb{X}_g x \mathbb{R}^m \to R$

Output: cost-to-go (at t) $\hat{\mathcal{T}}[J](x) : \mathbb{X}_g \to R$

1: construct the grid $\mathbb{Y}_g$;
2: use LLT to compute $J^{d*} : \mathbb{Y}_g \to R$;
3: **for** each $x \in \mathbb{X}_g$ **do**
4: $\quad \phi_x(y) \leftarrow C_x^*(-F_i(x) + J^{d*}(y))$ for $y \in \mathbb{Y}_g$
5: $\quad \hat{\mathcal{T}}[J](x) \leftarrow \max_{y \in \mathbb{Y}_g}\{\langle F_s(x), y\rangle - phi_x(y)\}$
6: **end for**

Setting 2

Algorithm 2 Implementation of the d-CDP operator for Setting 2 [1]

Input: dynamics $F_s : \mathbb{R}^n \to \mathbb{R}^n, B :\in \mathbb{R}^{nxm}$; cost-to-go (at t+1) $J : \mathbb{X}_g \to R$; stage cost (state) $C_s(x) : \mathbb{X}_g x \mathbb{R}^n \to R$; conjugate of stage cost (input) $C_i^* : \mathbb{R}^m \to R$; grid $\mathbb{Z}_g \subset R$

Output: cost-to-go (at t) $\hat{\mathcal{T}}_d^m[J](x) : \mathbb{X}_g \to R$

1: construct the grid $\mathbb{Y}_g$;
2: use LLT to compute $J^{d*} : \mathbb{Y}_g \to R$ from $J : \mathbb{X}_g \to \mathbb{R}$;
3: $\phi(y) \leftarrow C_i^*(-B^T y) + JJ^{d*}(y)$ for $y \in \mathbb{Y}_g$
4: use LLT to compute $\phi^{d*} : \mathbb{Z}_g \to \mathbb{R}$ from $\phi : \mathbb{Y}_g \to \mathbb{R}$
5: **for** each $x \in \mathbb{X}_g$ **do**
6: use LERP to compute $\overline{\phi^{d*d}}(F_s(x))$ from $\phi^{d*} : \mathbb{Z}_g \to \mathbb{R}$
7: $\hat{\mathcal{T}}_d^m[J](x) \leftarrow C_s(x) + \overline{\phi^{d*d}}(F_s(x))$
8: **end for**

Bibliography

[1] M. A. S. Kolarijani and P. M. Esfahani, "Fast approximate dynamic programming for input-affine dynamics," 2021.

[2] R. E. Bellman, *Dynamic Programming*. USA: Dover Publications, Inc., 2003.

[3] M. B. Haugh and L. Kogan, "Chapter 22 duality theory and approximate dynamic programming for pricing american options and portfolio optimization," in *Financial Engineering* (J. R. Birge and V. Linetsky, eds.), vol. 15 of *Handbooks in Operations Research and Management Science*, pp. 925–948, Elsevier, 2007.

[4] M. Uddin, M. Romlie, M. Abdullah, S. Abd Halim, A. Bakar, and T. Kwang, "A review on peak load shaving strategies," *Renewable and Sustainable Energy Reviews*, vol. 82, pp. 3323–3332, 11 2017.

[5] A. Oudalov, R. Cherkaoui, and A. Beguin, "Sizing and optimal operation of battery energy storage system for peak shaving application," in *2007 IEEE Lausanne Power Tech*, pp. 621–625, 2007.

[6] Q. Wei, G. Shi, R. Song, and Y. Liu, "Adaptive dynamic programming-based optimal control scheme for energy storage systems with solar renewable energy," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 7, pp. 5468–5478, 2017.

[7] R. Martins, H. C. Hesse, J. Jungbauer, T. Vorbuchner, and P. Musilek, "Optimal component sizing for peak shaving in battery energy storage system for industrial applications," *Energies*, vol. 11, no. 8, 2018.

[8] F. Borrelli, M. Baoti, A. Bemporad, and M. Morari, "Dynamic programming for constrained optimal control of discrete-time linear hybrid systems," *Automatica*, vol. 41, no. 10, pp. 1709–1721, 2005.

[9] Q. Wei, D. Liu, and H. Lin, "Value iteration adaptive dynamic programming for optimal control of discrete-time nonlinear systems," *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 840–853, 2016.

[10] A. Kleywegt, V. Nori, and M. Savelsbergh, "Dynamic programming approximations for a stochastic inventory routing problem," *Transportation Science*, vol. 38, pp. 42–70, 02 2004.

[11] D. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*, vol. 27. 01 1996.

[12] D. Bertsekas, *Dynamic programming and optimal control: Volume I*, vol. 1. Athena scientific, 2012.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[14] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[15] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE transactions on automatic control*, vol. 42, no. 5, pp. 674–690, 1997.

[16] B. V. R. D.P. de Farias, "The linear programming approach to approximate dynamic programming," *Operations Research*, vol. 51, no. 6, pp. 850–865, 2003.

[17] F. Bach, "Max-plus matching pursuit for deterministic markov decision processes," 2019.

[18] E. Berthier and F. Bach, "Max-plus linear approximations for deterministic continuous-state markov decision processes," *IEEE Control Systems Letters*, vol. 4, no. 3, pp. 767–772, 2020.

[19] R. T. ROCKAFELLAR, *Convex Analysis*. Princeton University Press, 1970.

[20] Y. Lucet, "Faster than the Fast Legendre Transform, the Linear-time Legendre Transform," *Numerical Algorithms*, vol. 16, pp. 171–185, Mar. 1997.

[21] S. Davies, "Multidimensional triangulation and interpolation for reinforcement learning," in *Advances in Neural Information Processing Systems* (M. C. Mozer, M. Jordan, and T. Petsche, eds.), vol. 9, MIT Press, 1997.

[22] D. W. Moore, *Simplicial Mesh Generation with Applications*. PhD thesis, USA, 1992. UMI Order No. GAX93-00795.

[23] K. Murota, "Discrete convex analysis," *Mathematical Programming*, vol. 83, no. 1, pp. 313–371, 1998.

[24] K. Murota, *Recent Developments in Discrete Convex Analysis*, pp. 219–260. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[25] K. Murota, *Discrete Convex Analysis: Monographs on Discrete Mathematics and Applications 10*. USA: Society for Industrial and Applied Mathematics, 2003.