UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Value Function Estimation in Optimal Control via Takagi-Sugeno Models and Linear Programming

PhD Dissertation

Author: Henry Paúl Díaz Iza

Supervisors: D. Antonio Sala Piqueras

D. Leopoldo Armesto Ángel

February, 2020

Instituto Universitario de Automática e Informática Industrial

Departamento de Ingeniería de Sistemas y Automática

Universitat Politècnica de València

This work was supported by the National Department of Higher Education, Science, Technology and Innovation of Ecuador (SENESCYT), and the Spanish ministry of Economy and European Union, grant DPI2016-81002-R (AEI/FEDER, UE). The author also received the grant for a predoctoral stay, Programa de Becas Iberoamérica- Santander Investigación 2018, of the Santander Bank.

Acknowledgements

Sin duda alguna el primer agradecimiento es para Dios, que sin su bendición de día tras día esta meta no podría haber sido lograda.

Un agradecimiento especial a mis directores Antonio Sala y Leopoldo Armesto por haberme permitido realizar mi tesis bajo su tutela y guiarme en cada paso dado, gracias por contribuir en mi crecimiento tanto profesional como personal.

Tengo que expresar mi más profundo agradecimiento a todas las personas que han estado allí para apoyarme, personas que me han ofrecido su amistad y me han hecho sentir como en casa. A mis amigos del máster Luis, Javier y Andrés. A mis compañeros que están y han pasado por *La Sala*: Ale, Yadira, Vanessa, Alberto, Nobel, Juanfer, Clara, Iván, Frank, Gloria, Eslam, Carlos, Michelle, Manuel, Temo, Rubén, Julio y Victor. A mis compañeros de piso Anita, Andrés y Marcelo. Ha sido un placer conocerlos a todos ustedes y espero que podamos vernos de nuevo en cualquier parte del mundo.

Agradezco a Elizabeth por su apoyo incondicional y por sus palabras de aliento y optimismo.

Finalmente, quiero expresar mi gratitud a mi madre, padre y hermano que siempre han estado allí apoyándome todo este tiempo. Ahora, después de todos estos años, puedo volver a casa.

Abstract

The present Thesis employs dynamic programming and reinforcement learning techniques in order to obtain optimal policies for controlling nonlinear systems with discrete and continuous states and actions. Initially, a review of the basic concepts of dynamic programming and reinforcement learning is carried out for systems with a finite number of states. After that, the extension of these techniques to systems with a large number of states or continuous state systems is analysed using approximation functions. The contributions of the Thesis are:

- A combined identification/Q-function fitting methodology, which involves identification of a Takagi-Sugeno model, computation of (sub)optimal controllers from Linear Matrix Inequalities, and the subsequent data-based fitting of Q-function via monotonic optimisation.

- A methodology for learning controllers using approximate dynamic programming via linear programming is presented. The methodology makes that ADP-LP approach can work in practical control applications with continuous state and input spaces. The proposed methodology estimates a lower bound and upper bound of the optimal value function through functional approximators. Guidelines are provided for data and regressor regularisation in order to obtain satisfactory results avoiding unbounded or ill-conditioned solutions.

- A methodology of approximate dynamic programming via linear programming in order to obtain a better approximation of the optimal value function in a specific region of state space. The methodology proposes to gradually learn a policy using data available only in the exploration region. The exploration progressively increases the learning region until a converged policy is obtained.

Resum

La present Tesi empra tècniques de programació dinàmica i aprenentatge per reforç per al control de sistemes no lineals en espais discrets i continus. Inicialment es realitza una revisió dels conceptes bàsics de programació dinàmica i aprenentatge per reforç per a sistemes amb un nombre finit d'estats. S'analitza l'extensió d'aquestes tècniques mitjançant l'ús de funcions d'aproximació que permeten ampliar la seua aplicabilitat a sistemes amb un gran nombre d'estats o sistemes continus. Les contribucions de la Tesi són:

Es presenta una metodologia que combina identificació i ajust de la funció Q, que inclou la identificació d'un model Takagi-Sugeno, el càlcul de controladors subòptims a partir de desigualtats matricials lineals i el consegüent ajust basat en dades de la funció Q a través d'una optimització monotónica.

Es proposa una metodologia per a l'aprenentatge de controladors utilitzant programació dinàmica aproximada a través de programació lineal. La metodologia fa que ADP-LP funcione en aplicacions pràctiques de control amb estats i accions continus. La metodologia proposada estima una cota inferior i superior de la funció de valor òptima a través de aproximadores funcionals. S'estableixen pautes per a les dades i la regularització de regresores amb la finalitat d'obtenir resultats satisfactoris evitant solucions no fitades o mal condicionades.

Es planteja una metodologia sota l'enfocament de programació lineal aplicada a programació dinàmica aproximada per a obtenir una millor aproximació de la funció de valor òptima en una determinada regió de l'espai d'estats. La metodologia proposa aprendre gradualment una política utilitzant dades disponibles només a la

regió d'exploració. L'exploració incrementa progressivament la regió d'aprenentatge fins a obtenir una política convergida.

Resumen

La presente Tesis emplea técnicas de programación dinámica y aprendizaje por refuerzo para el control de sistemas no lineales en espacios discretos y continuos. Inicialmente se realiza una revisión de los conceptos básicos de programación dinámica y aprendizaje por refuerzo para sistemas con un número finito de estados. Se analiza la extensión de estas técnicas mediante el uso de funciones de aproximación que permiten ampliar su aplicabilidad a sistemas con un gran número de estados o sistemas continuos. Las contribuciones de la Tesis son:

- Se presenta una metodología que combina identificación y ajuste de la función Q, que incluye la identificación de un modelo Takagi-Sugeno, el cálculo de controladores subóptimos a partir de desigualdades matriciales lineales y el consiguiente ajuste basado en datos de la función Q a través de una optimización monotónica.

- Se propone una metodología para el aprendizaje de controladores utilizando programación dinámica aproximada a través de programación lineal. La metodología hace que ADP-LP funcione en aplicaciones prácticas de control con estados y acciones continuos. La metodología propuesta estima una cota inferior y superior de la función de valor óptima a través de aproximadores funcionales. Se establecen pautas para los datos y la regularización de regresores con el fin de obtener resultados satisfactorios evitando soluciones no acotadas o mal condicionadas.

- Se plantea una metodología bajo el enfoque de programación lineal aplicada a programación dinámica aproximada para obtener una mejor aproximación de la función de valor óptima en una determinada región del espacio de estados. La metodología propone aprender gradualmente una política utilizando datos disponibles sólo en la región de exploración. La exploración incrementa progresivamente la región de aprendizaje hasta obtener una política convergida.

Contents

Chapter 1

Introduction

1.1 Motivation and objectives

Dynamic programming and reinforcement learning are an active research area that has generated a large number of new algorithms and applications in recent years. Many of the reinforcement learning algorithms are based on the working principles of dynamic programming. Despite the progress made in recent years, there are still aspects that limit the applicability of these techniques to complex problems. These aspects are related to the limited scalability to problems in continuous spaces or problems defined by a large number of states or control actions. In order to face this problem, the use of function approximations is necessary. It implies a correct selection of the approximation method to make the learning process converges to useful solutions. Another problem to overcome is a large amount of data needed to learn useful policies, depending on the systems obtaining these data can be difficult or expensive; however, the availability of real data makes the identification of models possible. These models can be used in the learning process to obtain simulated data, thus preventing that the real systems reach dangerous conditions or suffer damages. These aspects have motivated us to propose new methodologies that allow the use of these techniques for the control of robotics and electromechanical systems.

The main objective of this Thesis is to develop new techniques of nonlinear optimal control using approximate dynamic programming. In order to get this objective, we use a Takagi-Sugeno model parametrisation for modelling nonlinear systems and using this parametrisation compute a fuzzy LMI-based control to initialise the reinforcement learning algorithm. This parametrisation avoids that learning algorithm can not converge or even provide a non-stabilising controller.

Additionally, the classical algorithms for solving the dynamic programming and reinforcement learning are iterative, and they can have convergence problems when the value function is approximated. So, we propose a methodology inspired in the relaxation of the Bellman equation and the linear programming approach in order to provide lower and upper bounds in problems that compute the approximated value function.

In real applications a sweep of all states is impossible. However, through the exploration using a predefined controller is possible to get a dataset that allows us to apply a learning technique. In this context, we propose a methodology that explores the state space using a learned/predefined controller in order to get a dataset to apply approximated dynamic programming by linear programming. We get a lower bound and controller defined only in the region where there are samples, and the exploration continues until converges a useful policy.

1.2 Structure of the Thesis

This Thesis is divided in two parts:

- Part I summarises the fundamental concepts related to dynamic programming and reinforcement learning. Chapter 2 provides the theoretical foundations to describe the rest of the concepts introduced throughout the Thesis. It presents the framework of Markov Decision Processes to present the elements of dynamic programming and reinforcement learning formally. The fundamental algorithms of dynamic programming and reinforcement learning are introduced, assuming that the space of states and actions is discrete and small enough to be enumerated and stored in tables.

  Chapter 3 extends the classical dynamic programming and reinforcement learning algorithms to the more general case where state and action are continuous variables, which is essential to face real problems. The approximation functions techniques are used as a compact representation of the value functions.

Chapter 4 reviews concepts about Policy Search methods, providing a brief description of the main algorithms that use the model-free and model-based approaches.

- Part II contains the contributions of this work. The first contribution is described in Chapter 5, where it presents a methodology that involves identification of a Takagi-Sugeno model, computation of (sub)optimal controllers from Linear Matrix Inequalities, and subsequent data-based fitting of the Q-function via monotonic optimisation.

  Chapter 6 presents a methodology that makes approximate dynamic programming via linear programming to work in practical control applications with continuous state and input spaces. Lower and upper bounds of the value function are computed, the gap between them provides an idea of the suitability of the regressors for a particular application.

  Chapter 7 illustrates an extension of approximate dynamic programming solved with linear programming in order to get a better approximation of the optimal policy only in the region where the data are available. The learning process is done incrementally, using a learned/predefined controller for the exploration process.

- The Thesis ends in Chapter 8; this chapter describes the conclusions resulting from the work carried out and identifies opportunities for future research.

  The methodology proposed in Chapter 6 assumes that any state can be known throughout the operating space of the system, which in real systems is not practical. For this reason, Chapter 7 proposes previous ideas to extend the application of the methodology of Chapter 6 to realistic problems in which data are only available in some areas of the system's operating space.

1.3 Publications

The research work during the doctoral period has led to several publications, which are listed below. Publications [3, 7] on policy search are close to the main research topic of the Thesis (optimal control, reinforcement learning), but outside of the core contributions (chapters 5 to 7) on dynamic programming. Given that policy search was the thesis topic of J. Pastor, the main findings of the two referred works are included in Chapter 4 in the state-of-the-art part.

Journal papers

1. Díaz, Henry, Leopoldo Armesto and Antonio Sala (2019) "Metodología de programación dinámica aproximada para control óptimo basada en datos". In: Revista Iberoamericana de Automática e Informática industrial, [S.l.], v. 16, n. 3, p. 273-283, jun. 2019. ISSN: 1697-7920. DOI:10.4995/riai.2019.10379.

2. Díaz, Henry, Leopoldo Armesto and Antonio Sala (2018) "Fitted Q-Function Control Methodology Based on Takagi-Sugeno Systems". In: IEEE Transactions on Control Systems Technology, vol. 28, no. 2, pp. 477-488, March 2020. ISSN:2374-0159. DOI: 10.1109/TCST.2018.2885689.

Conference papers

3. Pastor, Jose, Henry Díaz, Leopoldo Armesto, Alicia Esparza and Antonio Sala (2018) "Learning Upper-Level Policy using Importance Sampling-based Policy Search Method" In: 2018 7th International Conference on Systems and Control,pp. 188-193. DOI: 10.1109/ICoSC.2018.8587772.

4. Díaz, Henry, Leopoldo Armesto and Antonio Sala (2017) "Improving LMI Controllers for Discrete Nonlinear Systems using Policy Iteration" In: 2017 21th International Conference on Systems Theory, Control and Computing (ICSTCC), pp. 833-838. DOI: 10.1109/ICSTCC.2017.8107140.

5. Díaz, Henry, Antonio Sala and Leopoldo Armesto (2017). "Aprendizaje por Refuerzo para sistemas lineales discretos con dinámica desconocida: Simulación y Aplicación a un Sistema Electromecánico". In: Actas de las XXXVIII Jornadas de Automática, pp. 360-367.

6. Díaz, Henry, Leopoldo Armesto and Antonio Sala (2016) "Improvement of LMI controllers of Takagi-Sugeno models via Q-learning". In: IFAC PaperOnLine 49.5. 4th IFAC International Conference on Intelligent Control and Automation Science (ICONS 2016), pp. 67-72. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2016.07.091.

7. Pastor, Jose, Henry Díaz, Leopoldo Armesto and Antonio Sala (2016). "Aprendizaje por Refuerzo con Búsqueda de Políticas: Simulación y Aplicación a un Sistema Electromecánico". In: Actas de las XXXVII Jornadas de Automática, pp.597-604.

Contributions under review/in elaboration

8. Díaz, Henry, Antonio Sala, and Leopoldo Armesto (2019). " A Linear Programming Methodology for Approximate Dynamic Programming". Submitted.

Part I

State of the art

Chapter 2

Dynamic Programming and Reinforcement Learning

Dynamic programming and reinforcement learning are algorithmic methods used for finding optimal solutions in sequential decision systems. The dynamic programming has a robust mathematical framework to solve the sequential decision systems when the dynamics model of the system is available. Reinforcement learning is considered an extension of dynamic programming that provides solutions without the need to know the dynamics model of the system. Many of the algorithms in reinforcement learning come from dynamic programming. Under the control systems approach, reinforcement learning combines some features of optimal control and adaptive control for the design of feedback controllers. This chapter describes the fundamentals of dynamic programming and reinforcement learning, as well as their basic algorithms.

2.1 Introduction

Dynamic programming (DP) (Bellman 1957) and reinforcement learning (RL) (Sutton and Barto 2018) are methods developed to compute optimal solutions in sequential decision problems (Busoniu et al. 2010; Lewis and Liu 2013), and they may be part of the stochastic optimisation methods (Powell 2019). In order to compute the optimal solution, the DP methods need to know the behaviour of the system, that means the dynamic model should be known *a priori*. On the other hand, RL calculates the optimal solution without the need of model. DP has an essential theoretical and mathematical development, and RL can be considered an extension of the applicability of DP (Sutton and Barto 2018; Bertsekas 2019). The DP and RL solve sequential decision problems; therefore, they exist at the intersection of various fields of science such as operational research, engineering, artificial intelligence, neuroscience, psychology, economics, mathematics, computer science among others.

DP is a fundamental part of optimal control (Bertsekas 2012; Bertsekas 2017). DP has its origin and development in the 1950s by Richard Bellman (Bellman 1957; Dreyfus and Bellman 1962). The objective of DP is to compute an optimal policy that minimises a cost index over time to achieve a specific system behaviour, such as minimum fuel, minimum energy, minimum risk, etc. The main idea behind DP is to divide or fragment the main problem into smaller subproblems and apply the *Bellman optimality principle* (Powell 2011; Bertsekas 2012). DP has been successfully applied to several problems where an exact system model is available.

RL can be considered an extension of the DP because it computes the optimal solution of the sequential decision problems without the model of the system (Sutton and Barto 2018; Bertsekas 2019). RL has its origins in the artificial intelligence (AI), and it is inspired in the learning process of animal learning (Niv 2009; Sutton and Barto 2018). People and animals learn through the process of interaction or exploration of the environment/system. They modify their actions in order to improve the reward obtained by observing responses to the interaction (Wiering and Otterlo 2012; Sutton and Barto 2018). So actions with positive rewards tend to repeat themselves, and actions with negative rewards tend to disappear (Skinner 1938). For example, Ivan Pavlov, in his experiments, used elementary rewards or punishments to modify the behaviour patterns of his dogs through conditioned reflexes (or stimulus-response learning) (Pavlov 1927). In the neuroscience field, the neurotransmitter dopamine can act as a reward signal that favours learning at the neuronal level (Doya, Kimura, and Kawato 2001; Lewis and Vrabie 2009). Most of the RL increase the practical applicability of DP algorithms by eliminating the need to have an in-depth knowledge of the system.

In this chapter, we will describe the DP/RL fundamentals in problems where the state and action spaces are finite and can be stored in tables. Throughout the Thesis, the information will be presented using notation and terminology of the control systems approach. This chapter is organised as follows. Section 2.2 provides a background of concepts related to Markov decision process, optimality criteria, policies, value functions, evaluation, and policy improvement. Once these previous concepts have been introduced, Section 2.3 and Section 2.4 explain the basic principles and algorithms of DP and RL, respectively. Finally, a summary is described in Section 2.5.

2.2 Background

2.2.1 Markov Decision Processes

The DP and RL methods for finding an optimal *policy* take the Markov decision processes (MDP) as a mathematical framework (Busoniu et al. 2010; Bertsekas 2017; Sutton and Barto 2018). Formally, a finite MDP is defined as the tuple $\{\mathbb{X}, \mathbb{U}, f, L\}$, where $\mathbb{X}$ is a finite set of states , $\mathbb{U}$ is a finite set of actions , the transition function is f, and the reward function L. We briefly describe the elements of the finite MDPs and their interaction, for more details is suggested to see (Sigaud and Buffet 2013; Puterman 2014; Bertsekas 2017).

- The finite set of states is $\mathbb{X}$; each state $x \in \mathbb{X}$ is the minimal set of variables with the minimum dimension to describe how the system evolves. In each time step, the system in a specific state x captures and summarises all information of the previous states (Powell 2011; Albertos and Sala 2006).
- The finite set of actions (control signals) is $\mathbb{U}$. The action $u \in \mathbb{U}$ is used to control the transitions made by the system in each time step. The selection of the control action in each state is generated through policy π. A policy π is a mapping of each state $x \in \mathbb{X}$ into a control action $u \in \mathbb{U}$.
- The transition function f defines the change from the state x to the state x'; it characterises the dynamics of the system.
- The reward function L, which is called a *cost function* in the control systems context, is a scalar function that provides the immediate cost r of applying an action u in a state x at time step t.

In general terms, the interaction of the elements in an MDP is described as follows. Applying a control action u in the state x at time step t, the state changes from x to x' based on the system transition function f. The action taken receives the scalar reward r, it is calculated with the reward function L that evaluates the immediate effect of the action u. Control actions are taken according to policy π of the controller (Busoniu et al. 2010). This interaction is repeated continuously in the case of *continuous problems* or the case of *episodic problems*;

the initial and final states are clearly defined. The policy, transition function and the reward function can be deterministic or stochastic. MDPs satisfy the Markov property, which establishes that applying a control action, the resulting state only depends on the current state and action. This property is fundamental because it provides important theoretical convergence guarantees for the DP and RL algorithms (Busoniu et al. 2010; Bertsekas 2017).

2.2.2 Optimality Criteria

The optimality criteria are the mathematical representations of the objective that we want to optimise. The DP and RL algorithms find a policy that optimises the cost obtained by the controller over time; this cost is known as *return* (Sutton and Barto 2018; Busoniu et al. 2010). The cost is quantified by the cumulative sum of immediate costs resulting from applying the control actions under policy π. The optimality criteria that represent most of the problems are: finite horizon, infinite horizon and averaged optimality models (Wiering and Otterlo 2012).

The *finite horizon* model $\left(\sum_{t=0}^{h} r_t\right)$ accumulates the cost of a fixed-length trajectory, and h is a finite horizon. The second optimality model is an *infinite horizon* $\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$. It takes into consideration a long-term reward. The future rewards that are received from the future are discounted according to how far in time they would be received by a *discount factor* γ $(0 \leqslant \gamma < 1)$. A third model of optimality is the model of *averaged return* $\left(\lim_{h\to\infty} \frac{1}{h} \sum_{t=0}^{h} r_t\right)$. This model also takes into consideration the long-term reward. Note that only returns for the deterministic case are defined. For generalisation to stochastic systems see (Busoniu et al. 2010; Wiering and Otterlo 2012; Bertsekas and Tsitsiklis 1996). The choice of the optimality criteria depends on the learning problem to be solved. For example, if the duration of the episode is known, the finite horizon model is the best option. On the other hand, if the length is not known or if the problems are continuous, the infinite horizon model is the most appropriate choice.

Most DP and RL algorithms use the infinite horizon discounted model because it has theoretical properties that make it more suitable for mathematical analysis (Bertsekas and Tsitsiklis 1996). The discount factor γ is a way of considering the uncertainty associated with future costs, and mathematically it ensures that the return is a finite quantity (Busoniu et al. 2010). The selection of the discount factor γ influences in both the quality of the policy obtained and the convergence speed. Usually, in the AI, the value of the discount factor is less than unity in order to bound the rewards. On the other hand, in control systems point of view, the discount factor is equal to 1. That means that the rewards are not bounded and it is necessary to use additional stability conditions in the problem to be solved. (Busoniu et al. 2018; Postoyan et al. 2017).

2.2.3 Value functions

A value function indicates the goodness (or badness) of the controller to be in a given state when it follows a specific policy (Sutton and Barto 2018). Value functions are the link between optimality criteria to policies (Wiering and Otterlo 2012). There are two types of value functions *state-value* and *action-value* functions. A state-value function V (V-function) estimates the goodness of being in a state. An action-value function Q (Q-function) estimates the goodness of applying an action in a state. Note that, when we use the term *value function* is to refer to function V-function and Q-function in a general way.

The value of a state x under the policy π, denoted as $V^\pi(x)$ is the *return* from starting in the state x and following the policy π. The value function in the deterministic case using the discounted infinite horizon model is:

$$V^\pi(x) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2.1}$$

Equation (2.1) can be defined recursively in terms of the *Bellman equation* (Bellman 1957). Then for any policy π and any state x is defined by

$$V^\pi(x) = r_t + \gamma V^\pi(x_{t+1}) \tag{2.2}$$

Equation (2.2) shows that the value function $V^\pi(x)$ for the policy π in the state x is defined with r_t (immediate reward) and the value function of the next state $V^\pi(x_{t+1})$. Note that, the rewards of the following states are represented with $V^\pi(x_{t+1})$, the discount factor γ weighs it.

The aim is to find a policy that minimises the accumulated cost. For this purpose, the equation (2.1) will be optimised for all states $x \in \mathbb{X}$, and the optimal solution $V^* = V^{\pi^*}$ satisfies the equation (2.3) that is the *Bellman Optimality Equation.* In order to select an optimal action given V^*, we can apply a *greedy policy* equation (2.4) that selects the best action using the V-function.

$$V^*(x) = \min_{u \in U} \left(L(x,u) + \gamma V^*(x') \right) \tag{2.3}$$

$$\pi^*(x) = \operatorname*{argmin}_{u} \left(L(x,u) + \gamma V^*(x') \right) \tag{2.4}$$

Similarly, using the Q-function can be defined as the return starting from the state x, taking an action u and following the policy π.

The Bellman equation and the Bellman optimality equation with action-value function as defined in the equations (2.5) and (2.6), respectively.

$$Q^\pi(x,u) = r_t + \gamma Q^\pi(x',u') \tag{2.5}$$

$$Q^*(x,u) = L(x,u) + \gamma \min_{u'} Q^*(x',u') \tag{2.6}$$

The relationship between Q^* and V^* is $V^*(x) = \min_u Q^*(x,u)$ and the selection of the optimal action is $\pi^*(x) = \operatorname*{argmin}_u Q^*(x,u)$.

In other words, the best action is the action that has the lowest expected cost based on possible next states resulting from taking that control action. Note that the introduced concepts focus on the deterministic case for the generalisation to the stochastic case can be referenced in (Kaelbling, Littman, and Moore 1996; Busoniu et al. 2010; Szepesvári 2010).

2.2.4 Model-based and model-free

In essence, the DP and RL algorithms compute an optimal policy π^* given a sequential decision problem. The main distinction between DP and RL algorithms is the need to know the model of the system. Therefore, DP algorithms are generally known *model-based*, and RL algorithms are known as *model-free* (Powell 2011; Sutton and Barto 2018; Bertsekas 2017). Model-based algorithms need to know the model in advance, and they can be used to determine value functions and policies using the Bellman equations. The model of the system can be estimated using identification techniques (Ljung 2001). In many practical applications, a simulation model is easier to find than an analytical expression (Sutton and Barto 2018; Bertsekas and Tsitsiklis 1996). Model-free algorithms do not depend on the availability of a model. Instead, these algorithms depend on the *interaction* with the system. For example, a policy simulation can generate samples of state transitions and rewards, and these samples are used to estimate action-state value functions (Q-functions). Since the model is unknown, the controller explores the state space for information. It naturally involves performing a balanced *exploration-exploitation* to obtain an optimal policy.

2.2.5 General policy iteration principle

An essential mechanism in the DP and RL algorithms is the principle called *generalised policy iteration* (GPI), it consists of two processes of interaction: *policy evaluation* and *policy improvement* (Wiering and Otterlo 2012; Sutton and Barto 2018). Policy evaluation or *prediction problem* computes the value function of the current policy. The policy improvement evaluates the action values for each state in order to find possible improvements (Szepesvári 2010; Sutton and Barto 2018). The main objective of policy evaluation is to gather information about the policy for the computation of policy improvement. Both evaluation and improvement steps can be implemented and interleaved in several different ways. The DP and RL algorithms can be classified according to how the optimal policy is obtained in policy iteration, value iteration and policy search (Busoniu et al. 2010).

Policy iteration algorithm (Howard 1960) evaluates policies by building their value functions (rather than the optimal value function) and use these value functions to find new and better policies. Value iteration algorithms look for the optimal value of the value function, which consists of the minimum cost in each state or each state-action pair. The optimal value function is used to calculate the optimal policy. Policy search algorithms (Deisenroth, Neumann, Peters, et al. 2013; Sigaud and Stulp 2019) use optimisation techniques to search the optimal policy directly.

2.3 Dynamic Programming

DP refers to a set of techniques to obtain the solution to a sequential decision problem. The DP is based on the Bellman principle of optimality (Bellman 1957), and DP techniques limited their applicability to the availability of the system behaviour model that, in many cases is a complex or very costly task to obtain (Lewis and Vrabie 2009; Powell 2011; Bertsekas and Tsitsiklis 1996).

On the other hand, the theoretical and algorithmic analyses of these techniques are essential because they provide the foundation for the RL techniques. RL can be considered a theoretical extension of the DP to achieve the same results without the condition of known system behaviour model and also reducing the computational load (Sutton and Barto 2018).

2.3.1 Policy iteration and Value iteration algorithms

The two main DP algorithms are policy iteration and value iteration. The description of these methods is limited to finite and sufficiently small sets of states and control actions that they can store in tables. The algorithms will be described using Q-functions, but the analysis and description are also valid for V-functions.

Policy iteration

The policy iteration (PI) algorithm (Howard 1960) has two stages that are repeated alternately until the algorithm converges to the optimal policy. In the first stage, known as policy evaluation, the value function of the current policy is computed. In the second stage, known as policy improvement, an improved policy is computed using the value function obtained in the previous stage.

The policy evaluation is the first step in the PI algorithm, and it is a fundamental step in the PI algorithm because if the evaluation is exact in the next step policy improvement to find a better policy is guaranteed (Wiering and Otterlo 2012). The set of all Q-functions is $\mathscr{Q}$, and the Bellman operator is $T^\pi : \mathscr{Q} \rightarrow \mathscr{Q}$ that in the deterministic case is:

$$[T^\pi(Q)](x,u) = L(x,u) + \gamma Q(x',u') \tag{2.7}$$

The Bellman operator T^π is a contraction if $\gamma < 1$ (Szepesvári 2010). For example, the Bellman operator T^π applied to functions Q and Q' with infinity norm:

$$||T^\pi(Q) - T^\pi(Q')||_\infty \leq \gamma ||Q - Q'||_\infty \tag{2.8}$$

T^π has a unique fixed point that is $Q^\pi = T^\pi(Q^\pi)$ and policy evaluation (PE) asymptotically converges to Q^π. Consequently, in practical situations, it is necessary to define a threshold $\varepsilon_{PE} > 0$ in order to stop the algorithm when $||Q^\pi_{\tau+1} - Q^\pi_\tau||_\infty \leq \varepsilon_{PE}$. One characteristic in the deterministic case of T^π is the linearity in the Q-values, that means, the Q^π can be calculated in an easy way solving a linear system of the equations.

The result of the policy evaluation stage is the value function Q^π of a policy π. The next step is trying to improve the policy. The current policy can be improved by selecting a better action in a specific state. The *greedy* policy π' is calculated with:

$$\pi'(x) = \underset{u}{\operatorname{argmin}}\, Q^\pi(x,u) \tag{2.9}$$

When the policy cannot be improved in this way, it means that the policy is already optimal, and its value function satisfies the Bellman equation for an optimal value function. In summary, the PI generates an alternate sequence of policies and value functions $\pi_0 \rightarrow Q^{\pi_0} \rightarrow \pi_1 \rightarrow Q^{\pi_1} \rightarrow \pi_2 \rightarrow Q^{\pi_2} \rightarrow ... \rightarrow \pi^*$

Value iteration

The algorithm where policy evaluation stage stops after a one-iteration is called the value iteration (VI) (Bellman 1957; Sutton and Barto 2018). At the evaluation stage, the value function is computed at the limit. However, it is not necessary to wait for a total convergence, but it is possible to stop the evaluation prematurely and improve the policy based on the evaluation (Wiering and Otterlo 2012). Thus the algorithm focuses on directly estimating the value function. The Bellman optimality operator as mapping $T : \mathscr{Q} \rightarrow \mathscr{Q}$. T computes the right-hand side of the Bellman optimality equation (2.6) for any Q-function, and in the deterministic case is:

$$[T(Q)](x,u) = L(x,u) + \gamma \min_{u'} Q(x',u') \tag{2.10}$$

The algorithm can start with an arbitrary Q_0 and each iteration updates the Q-function. T is contractive if the discount factor $\gamma < 1$ and will asymptotically converge to the unique fixed point $Q^* = T(Q^*)$ (Szepesvári 2010). For any pair of functions Q and Q':

$$||T(Q) - T(Q')||_\infty \leq \gamma ||Q - Q'||_\infty \tag{2.11}$$

The optimal solution can be reached after a sufficient number of iterations. We can define a threshold $\varepsilon_{QI} > 0$ in order to stop a Q-iteration algorithm when the difference between two consecutive Q-functions satisfies $||Q_{i+1} - Q_i||_\infty \leq \varepsilon_{QI}$. The Q-iteration algorithm return Q^* and the corresponding greedy policy. Therefore, each iteration is calculated a new value function until the stop criterion is met, and the optimal value function is reached $Q_0 \rightarrow Q_1 \rightarrow Q_2 \rightarrow ... \rightarrow Q^*$. Note that, the operator T is highly non-linear due to the minimisation on the set of actions, which implies some difficulty in solving the Bellman optimality equation compared to the Bellman equation with operator T^π used in the policy iteration algorithm (Busoniu et al. 2010).

A disadvantage in DP methods is that they require complete sweeping of the state, and the computational cost in large state space is considerable. One of the methods to alleviate this is the application of the asynchronous dynamic programming (Barto, Bradtke, and Singh 1991; Sutton and Barto 2018) that proposes flexibility in the update of the states. These algorithms update the values of the states in any order, using the values of other available states. The intention is to make a mixture between PI and VI updates and some other combinations of these methods. The implementation of PI and VI algorithms and a more extensive description about their deterministic and stochastic versions can be found in (Szepesvári 2010; Busoniu et al. 2010).

2.4 Reinforcement Learning

RL pursues the same objective as DP, which is to find an optimum that minimises the cost obtained by the controller with the difference that RL methods are model-free (Sutton and Barto 2018; Bertsekas 2019). The family of algorithms that describes in this section is called *temporal difference* (TD), that are the most extended methods[1] (Bertsekas and Tsitsiklis 1996; Lewis and Vrabie 2009; Kiumarsi et al. 2014; Sutton and Barto 2018; Bertsekas 2019).

2.4.1 Temporal difference

Temporal difference (TD) is a family of methods for estimating V-function or Q-function of a fixed policy (Sutton 1984; Sutton 1988); they learn from interactions with the system. In TD methods the value function is estimated based on previous estimates, with a technique known as *bootstrapping* (Sutton and Barto 2018). TD updates are made along the trajectory without waiting for the end of the episode. These methods are online and incremental, which is an advantage over methods that require a full sweep of the state space (Busoniu et al. 2018). For example, in the case of a V-function each time the controller performs a control action, the TD method uses the generated reward and the current estimate of V-function to make a new estimate according to the expression:

$$V_{k+1}(x_k) = V_k(x_k) + \alpha_k \left[r_{k+1} + \gamma V_k(x_{k+1}) - V_k(x_k) \right] \tag{2.12}$$

where $\alpha_k \in [0, 1]$ is the learning rate that determines the amount with V-function is updated in the state the x_k. The term in square brackets is known as the temporal difference and gives the method its name. In order to ensure the convergence of the algorithm, the learning rate α_k should satisfy Robbins-Monro conditions,

[1]There is another model-free method called Monte-Carlo (MC) RL methods. These methods are outside of the scope of this Thesis, and we do not discuss them. For details about MC methods, see, e.g., (Sutton and Barto 2018).

and all states should be visited infinitely (Szepesvári 2010; Busoniu et al. 2010). In practice, in order to obtain a suitable solution using TD is necessary an ad-hoc adjustment of the learning rate α_k.

When TD is applied to Q-functions and combined with the GPI principle, the resulting algorithms can solve the more general problem of learning that is to get an optimal policy. A requirement for such algorithms in order to converge is updating all action-state pairs of the Q-function indefinitely (Jaakkola, Jordan, and Singh 1994; Singh et al. 2000; Watkins and Dayan 1992). To satisfy this condition, the policy used by the controller should select in each state all possible actions with a probability greater than zero. The policy should explore all the state and action spaces. At the same time, the controller also needs to use the acquired knowledge to act appropriately and obtain the highest possible reward or, in other words, to exploit the knowledge it possesses. Both exploration and exploitation are contrary and give rise to the so-called exploration-exploitation dilemma (Sutton 1988). In practice, a compromise is required between the two requirements. The most widely and successful exploration strategies are $\epsilon - greedy$, Boltzmann exploration (softmax), and uncertainty optimism technique (Sutton, Barto, and Williams 1992). TD methods can classify in *Q-Learning, Sarsa and actor-critic* algorithms.

2.4.2 Q-learning and Sarsa

Q-learning (Watkins and Dayan 1992; Werbos 1992) is part of the TD methods, and it is considered a classical algorithm of reinforcement learning. Using Q-functions implies not requiring a model, and it iteratively estimates the Q-function in a similar way to algorithm VI from from DP. The Q-learning uses the tuple $(x_k, u_k, x_{k+1}, r_{k+1})$ to update the Q-function estimate using:

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k \left[r_{k+1} + \gamma \min_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k) \right] \tag{2.13}$$

where $\alpha_k \in (0.1]$ the sequence of learning rates. The Q-learning is a version online and incremental Q-iteration algorithm (Busoniu et al. 2018). The Q-learning is an *off-policy* algorithm since it converges towards an optimal policy independently of the policy that the controller uses to interact with the system. In order to converge, Q-learning requires that all the action-state pairs of the Q-function be updated indefinitely and that the sequence of learning rates be adjusted so that they decrease appropriately (Watkins and Dayan 1992; Bertsekas and Tsitsiklis 1996; Szepesvári 2010).

Sarsa (Rummery and Niranjan 1994; Sutton 1996) is another possible extension of the TD algorithm for estimating the optimal Q-function and obtaining the optimal policy. This algorithm takes its name from the elements used for updating the value function estimate: **S**tate(x_k), **A**ction (u_k), **R**eward (r_{k+1}), **S**tate (x_{k+1}), and **A**ction(u_{k+1}). The operating principle is the same as the PI algorithm from the DP. Unlike the PI algorithm, Sarsa does not need to know the MDP model but learns a policy *online* when the controller interacts with the system. Following the GPI principle, Sarsa begins with an arbitrary Q_0 that is updated after each interaction through the expression:

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k[r_{k+1} + \gamma Q_k(x_{k+1}, u_{k+1}) - Q_k(x_k, u_k)] \quad (2.14)$$

where α_k is the sequence of learning rates. The Sarsa operating principle is to evaluate the current policy and then find an improved policy for the new estimate of Q, and the process is repeated iteratively. For the convergence of the algorithm towards the optimal Q, the same conditions of Q-Learning must be met, and additionally, the exploration policy applied asymptotically should be greedy (Singh et al. 2000). Algorithms that learn from data acquired from the MDP need to sample state space and actions thoroughly. The policy used by the controller must choose all actions in all states with a probability greater than zero. It can be achieved by exploring techniques such as $\epsilon - greedy$ or Boltzmann. Sarsa is an *on-policy* algorithm because it requires that the policy the controller is learning is the same as the policy the controller uses to interact with the environment (Sutton and Barto 2018). In order to reduce the variance of the original Sarsa is possible to use the expected value of the next state-action pairs, this idea is implemented in the called *Expected Sarsa* algorithm (Van Seijen et al. 2009).

In general, the TD methods can improve their efficiency by using mechanisms like n-step bootstrapping or *eligibility traces*. Both n-step bootstrapping and eligibility traces generalise the one-step TD and MC learning methods creating intermediate methods. In the case of the n-step method, the update of temporal difference is done after n time steps. In the eligibility traces (Sutton 1988), a memory vector is used to estimate the value function. For further details, we recommended seeing (De Asis et al. 2018; Seijen 2016; Sutton and Barto 2018).

Actor-Critic methods

The actor-critic methods (Witten 1977; Barto, Sutton, and Anderson 1983; Konda and Tsitsiklis 2000) are on-policy learning methods. The main characteristic of these class of algorithms is that the policy is separated from the value function. The elements of this structure are the *critic* and the *actor*. The critic evaluates the actual policy using the temporal difference error, and the actor generates the control action. That means the value function is the critic and the policy is the actor. One advantage that presents these algorithms is with problems with many control actions or continue action in order to avoid to calculate all Q-values for only choose one. Another advantage is that these algorithms can learn stochastic policies.

2.5 Summary

In this Chapter, the basic concepts of dynamic programming and reinforcement learning were briefly reviewed, considering that the space of states and actions is discrete and small enough that it can be stored in tables. The DP and RL are methods that compute optimal solutions in sequential decision problems. The DP/RL framework is the Markov decision process, whose elements are: the finite set of states, the finite set of actions, the transition function and the reward function. The interaction of these elements makes to pose the sequential decision problem that DP and RL intend to solve. The objective of DP/RL is to minimise the long-term reward that is mathematical represent using optimality criteria. The optimal criteria are linked with the policy by the value function. The value function represents the goodness of the controller to be in a specific state. There are two kinds of value functions: V-functions and Q-functions.

The main difference between DP and the RL is the use or not of the model of the system. The DP methods use the model, so they are model-based. The main DP algorithms are policy iteration and value iteration that is based on the general policy iteration principle that consists in the interaction of two processes: the policy evaluation and the policy improvement. On the other hand, RL methods find the optimal solution without the dynamic model of the system; they need to explore/interact with the system in order to learn a policy. This characteristic is crucial in systems where the model is difficult or costly to obtain. The popular RL algorithms use the temporal difference method that estimates the value function using a bootstrapping technique; the most important algorithms are Q-learning, Sarsa and Actor-Critic.

These DP/RL methods described in this Chapter have a limited application, and they can not apply to systems with a large number of state and action spaces or continuous state systems. Therefore the use of function approximation techniques to represent value functions is necessary in order to extend these ideas to more complex problems. These techniques will be described in Chapter 3.

Chapter 3

Approximate Dynamic Programming

Dynamic programming techniques have a robust mathematical background, but their applications are limited to problems with a finite set of spaces and actions with small dimensions. A way to generalise this approach is using approximation functions. Approximation functions can use parametric and nonparametric methods to represent value functions. The generalisation using function approximation of the classical dynamic programming algorithms are presented. Besides, batch algorithms are briefly described. They combine the approximation of functions with efficient data handling. Finally, a brief explanation of the linear programming approach applied to approximate dynamic programming problems is given.

3.1 Introduction

In Chapter 2, the description of the fundamental algorithms of DP and RL was made. The conditions that we assume to apply these algorithms included that the state and action spaces must be discrete, finite, and small enough to be stored in tables (Bertsekas and Tsitsiklis 1996). They are also called *exact* or *tabular* algorithms, and under certain conditions, they asymptotically converge to the exact optimal solution. The disadvantage of this type of algorithms is that they cannot apply to realistic problems where the number of states and control actions is so large or continuous that the state cannot be stored in tables. Therefore, in this type of problem, it is unavoidable that the representation of state variables and actions must be carried out using approximation techniques.

Additionally, the use of approximation techniques not only emerges from problems of large dimensions or continuous variables that cannot be stored in tables but also in order to obtain a compact representation with few parameters that will be an approximate estimate of the complete function (Bertsekas and Tsitsiklis 1996; Bertsekas 2019). Algorithms described in Chapter 2 converge under the condition that all elements of the value function should be updated infinitely (Wiering and Otterlo 2012). When the systems are continuous, it is not possible to update the states infinitely. Therefore, the use of approximation methods is an advantage to generalise the controller behaviour in states that have not been visited before.

After the value function update is performed, much of the algorithms based on the value function approach require optimisation over control actions. It implies that if the control actions are continuous, this optimisation could be not convex and one solution is to discretise the control action space and find a solution more easily by enumeration (Busoniu et al. 2010). Another alternative in the case that the action space is continuous and discretisation is not an option is to use architectures that represent control actions explicitly as a policy search methods[1] (Deisenroth, Neumann, Peters, et al. 2013; Sigaud and Stulp 2019).

The fundamentals of RL for continuous spaces are introduced. The use of approximation methods for value function or policy extends the capabilities of the algorithms studied in previous chapters. Several algorithms based on the value function approach will be introduced. Mainly both policy iteration and value iteration algorithms, and some modification of them in order to improve the efficient data management are studied. The complete MDP model is assumed not to be known. Also, the linear programming approach (LP) is used to solve the approximate dynamic programming (ADP) (De Farias and Van Roy 2003). Using this approach, the solution that is obtained is a lower bound of the optimal solution because the Bellman equation is relaxed.

This Chapter is organised as follow; Section 3.2 describes function approximation and a brief description of parametric and nonparametric methods. Section 3.3 introduces the policy iteration and value iteration algorithms in continuous spaces: Sarsa and the Q-learning like representative algorithms to these methods. The batch RL algorithms are briefly described in Section 3.4. Section 3.6 describes the linear programming approach to approximate dynamic programming.

[1]Policy search methods will be discussed in Chapter 4 of this work

Finally, Section 3.7 summarises the fundamental concepts that we describe in this Chapter.

3.2 Function approximation for reinforcement learning

Commonly, any approximator can be considered as a mapping between the parameters space and space of functions to be represented (Bertsekas and Tsitsiklis 1996; Busoniu et al. 2010; Sutton and Barto 2018). Specifically, in RL, functions to represent are value or policy functions. Depending on the algorithm used, the value function, the policy or both can be approximated. The approximation of a function must adjust/train the approximator parameters to obtain a proper representation of the function (Bertsekas and Tsitsiklis 1996; Bertsekas 2019). The symbol that we will use to indicate that the function is approximate is $\hat{\cdot}$, for example, $\hat{V}, \hat{Q}$ represent approximated functions to estimate value functions:

$$\hat{V}(x;\theta) \approx V(x) \tag{3.1}$$

$$\hat{Q}(x,u;\theta) \approx Q(x,u) \tag{3.2}$$

where θ is the vector of parameters resulting from the approximation method used.

3.2.1 Parametric and nonparametric methods

The function approximation can be classified in parametric and nonparametric methods (Busoniu et al. 2010; Powell 2011). This classification is based on the numbers of parameters to adjust. Parametric approximator has a fixed number of parameters of adjusting, and the approximator does not depend on the numbers of samples or data. On the other hand, the nonparametric approximators adapt the number of parameters based on the samples or data (Busoniu et al. 2018).

In parametric methods, the number of parameters is defined in advance, and there is no dependence on the amount of data available. One advantage of approximation function is the compact representation. Instead of storing a value for each (x,u) pair, it is only necessary to store n parameters of the vector θ. Logically, when the state space is discrete, the number of parameters should be much less than $|X| \times |U|$, so that the approximate representation is more compact (Bertsekas and Tsitsiklis 1996). In function approximation, an *approximation error* is assumed.

A subgroup of approximators whose parametrisation is linear has particular relevance in DP and RL because they simplify the analysis of the theoretical properties of the algorithms (Bertsekas 2019). The output of an approximator can be calculated as a linear combination of the parameter vector. The approximate Q-function representation for a given pair (x, u) is:

$$\hat{Q}(x, u; \theta) = \sum_{l=1}^{n} \phi_l(x, u)\theta_l = \phi^T(x, u)\theta \tag{3.3}$$

where θ corresponds an n-dimensional vector of parameters, and $\phi(x, u)$ is a vector formed by the fixed functions $\phi_1, ..., \phi_n$, usually known as base functions(BF) (Busoniu et al. 2010).

The base functions can be constructed in different ways, some examples in RL are the binary base functions used in the tile coding, method also known formerly as CMAC (cerebellar model articulation controller) (Albus 1975), the radial base functions (Gaussian mainly) used in the fixed base RBF (radial basis function) networks, state aggregation (Singh, Jaakkola, and Jordan 1995; Bertsekas and Tsitsiklis 1996), multilinear interpolation (Davies 1997) and Kuhn triangulation (Munos and Moore 2002). In general, the approximator may be nonlinear in the parameters such as neural networks and deep networks are widely used type of nonlinear approximator (Bertsekas and Tsitsiklis 1996; Riedmiller 2005; Mnih et al. 2015).

On the other hand, the nonparametric methods (despite their name) also use a set of parameters to perform the approximation. The difference is the number of parameters and the approximator structure. They are not chosen by the user but are obtained from the training dataset. This type of method can be changed and adapted depending on the available data (Alpaydin 2014). Nonparametric methods in combination with RL are used and include *k-nearest* neighbours, the kernel methods (Shawe-Taylor, Cristianini, et al. 2004; Ormoneit and Sen 2002), among which the vectors machines support (Schölkopf, Burges, Smola, et al. 1999), Gaussian processes (Rasmussen 2003; Engel, Mannor, and Meir 2005), regression trees (Breiman 2017; Ernst, Geurts, and Wehenkel 2005), kernel methods with value function iteration (Farahmand et al. 2009) and policy iteration (Bethke, How, and Ozdaglar 2008) or regression trees with policy evaluation (Jodogne, Briquet, and Piater 2006). The main advantages of nonparametric approximators are high flexibility and their ability to adapt to the complexity of the approximator according to the amount of data available. On the other hand, the disadvantages are convergence problems and the computational load when the amount the data increase. Convergence problems occur due to modification in the structure of the approximator during optimal policy learning; it is difficult to obtain theoretical guarantees of convergence (Busoniu et al. 2010). Besides, if the amount of data acquired increases rapidly, the computational load and the amount of memory required by the approximator also tends to increase proportionally. Therefore, nonparametric methods are inappropriate to be used together with RL algorithms whose learning is online, since the amount of data sampled by the controller can increase indefinitely.

Although the flexibility of nonparametric approximators usually requires to be nonlinear, RL algorithms combined with nonlinear approximators are difficult to deal with theoretically and in practice often behave unsteadily. For this reason, and despite their limitations, linear approximators are used in most of the literature. There are also linear approximators that increase their flexibility by allowing the introduction of new base functions as needed (Munos and Moore 2002; Szepesvári and Smart 2004; Waldock and Carse 2008). In this class of algorithms, the approximation method cannot be considered purely parametric, since the structure of the approximator changes as a function of the data, one of the properties that characterise nonparametric methods.

3.3 Policy iteration and Value iteration algorithms in continuous spaces

The PI and VI with approximation functions is a natural extension of algorithms from DP. When the state space is large, or continuous, both the value functions and the policies cannot be represented accurately, so it is necessary to use an approximate representation. The big problem with function approximation is that the convergence to the optimal policy is not guaranteed (Sutton and Barto 2018). Therefore, special conditions need to be taken in order to get suitable solutions.

3.3.1 Policy iteration

In PI algorithm with function approximation methods is necessary that the initial policy should be admissible (Lewis, Vrabie, and Syrmos 2012; Liu et al. 2017). In general, the approximated PI algorithm starts with an initial admissible policy for which its value function is estimated; then this value function is used to find a new policy better than the old one. This process is repeated until it converges to the optimal policy.

The evaluation stage of the policy consists of solving the Bellman equation approximately. Usually, it is possible to avoid explicitly representing the policy, so it is only necessary to approximate the Q-function. The minimisation carried out on actions can be easily resolved when the space for actions is discrete. Otherwise, it can be an added difficulty (Busoniu et al. 2010). The representative example of the PI with approximate function is a version of the Sarsa algorithm (Busoniu et al. 2018).

The approximation methods generally establish a cost function to minimise, i.e., $J_{Q^\pi}(\theta) = ||Q^\pi - \hat{Q}||$. The methods used for minimisation are generally the stochastic gradient descent (SGD) or recursive least-squares (Geist and Pietquin 2013). In the case of SGD the cost function can be expressed as:

$$\hat{J}_{Q^\pi} = \sum_k \left(q_k^\pi - \hat{Q}(x_k, u_k) \right)^2 \tag{3.4}$$

Suppose that, at the instant k, we have access to an observation of Q denoted as q_k^π. When the controller performs a transition from the state x_k applying the control u_k is possible to adjust the approximator parameters with an amount proportional to the gradient of the empirical cost function that is evaluated only in the pair (x, u). The parameters are updated using Widrow-Hoff equation (Geist and Pietquin 2013):

$$\theta_{k+1} = \theta_k - \frac{1}{2}\alpha_k \frac{\partial}{\partial \theta_k} \left[q_k^\pi - \hat{Q}(x_k, u_k) \right]^2 \tag{3.5}$$

$$= \theta_k + \alpha_k \left[q_k^\pi - \hat{Q}(x_k, u_k) \right] \frac{\partial}{\partial \theta_k} \hat{Q}_k(x_k, u_k) \tag{3.6}$$

where α_k is a sequence of learning rates that satisfies the Robbins-Monro conditions (Szepesvári 2010; Powell 2011). One of the conditions that the approximator must meet is that it should be derivable with respect to the parameters. However, the updating rule defined in Equation (3.5) cannot be applied in practice since it is not possible to observe the value q_k^π. Instead, an estimate of that value can be calculated by applying the sampled Bellman operator:

$$r_{k+1} + \gamma \hat{Q}_k(x_{k+1}, u_{k+1}) \tag{3.7}$$

Including this estimate in the previous update rule gives the equation used by the Sarsa algorithm:

$$\theta_{k+1} = \theta_k + \alpha_k \left[r_{k+1} + \gamma \hat{Q}(x_{k+1}, u_{k+1}) - \hat{Q}(x_k, u_k) \right] \frac{\partial}{\partial \theta_k} \hat{Q}_k(x_k, u_k) \tag{3.8}$$

Where the term in square brackets corresponds to an approximation of the temporal difference used in the TD algorithm. When the linear approximators in the parameters, it is much easier to calculate the gradient. Applying the Equation (3.3) on the update rule is obtained:

$$\theta_{k+1} = \theta_k + \alpha_k \left[r_{k+1} + \gamma \phi^T(x_{k+1}, u_{k+1})\theta_k - \phi^T(x_k, u_k)\theta_k \right] \phi(x_k, u_k) \tag{3.9}$$

Also, the fact that the approximator is linear implies that the function $\hat{J}$ has a single global minimum, which improves the convergence properties. As with the exact version of the Sarsa algorithm, in the approximate version, it is also necessary that the controller incorporates some exploration technique to obtain samples (x, u) with $u \neq \pi(x)$. A detailed analysis of the convergence properties of the linearly approximated TD algorithm can be found in (Tsitsiklis and Van Roy 1997) and can be extended to the Sarsa algorithm (Szepesvári 2010).

3.3.2 Value iteration

When an algorithm based on VI is applied to a problem whose state space is continuous, the same difficulties appear as in the case of algorithms based on PI. The techniques for dealing with these difficulties also consist of representing value functions using approximators. Let us remember that the operating principle of VI algorithms consisted in calculating a sequence of value functions that converged towards the optimal value function, from which an optimal policy was obtained (see subsection 2.3.1).

The most studied algorithm of VI is the Q-learning algorithm (Horiuchi et al. 1996; Jouffe 1998; Fernández and Borrajo 1999; Glorennec 2000; Murphy 2005; Sherstov and Stone 2005); this algorithm is used as an illustrative example of the methods of VI in continuous spaces. The Q-learning algorithm with function approximation can be obtained following a procedure similar to the one previously carried out with the Sarsa algorithm. In this case, the objective is to estimate Q directly, so the theoretical cost function to be minimised is given by the difference between the optimal value function and the estimated value function:

$$J_{Q^*}(\theta) = ||Q^* - \hat{Q}|| \tag{3.10}$$

This theoretical cost function results in an empirical cost function where, at every instant k, the observations q_k^* and $\hat{Q}$ are used. The value q_k^* is not directly observable but is estimated by the sampled operator of optimality of Bellman:

$$r_{k+1} + \gamma \min_{u'} \hat{Q}_k(x_{k+1}, u') \tag{3.11}$$

Combining this estimate with the Widrow-Hoff rule gives the equation that updates the approximator parameters in the Q-learning algorithm. Also, if the approximator is linear in the parameters, this equation can be simplified by obtaining:

$$\theta_{k+1} = \theta_k + \alpha_k \left[r_{k+1} + \gamma \min_{u'} \phi^T(x_{k+1}, u')\theta_k - \phi^T(x_k, u_k)\theta_k \right] \phi(x_k, u_k) \tag{3.12}$$

When the controller uses the equation to update Q^*, the policy must perform exploration.

Q-learning is an off-policy algorithm; the convergence properties of TD cannot be extended to it. In fact, despite being one of the most widely used algorithms in practice, it is known that it can diverge when no restriction is applied to it. The only convergence test is provided by (Melo, Meyn, and Ribeiro 2008), where substantial restrictions are assumed on the distribution of the sampled state-action pairs. More recently, the appearance of the family of algorithms gradient temporal difference (GTD) (Sutton, Maei, and Szepesvári 2009; Sutton et al. 2009) has led to a new version of Q-learning, known as greedy GD (Maei et al. 2010), whose convergence is assured. However, even in the case of using a linear approximator, the J cost function is not convex so it can converge to local minima (Maei 2011). In general, the risk of divergence is present if the algorithms combine function approximation, bootstrapping and off-policy learning (Sutton and Barto 2018).

3.4 Batch reinforcement learning

The batch RL algorithms solve the RL problem using the *batch* idea in order to minimise the cost and find an optimal policy. They increased in importance a few years ago because the idea of using batch in learning has two main benefits: data-efficient and stability (Wiering and Otterlo 2012). In the beginning, batch RL algorithms used a fixed data set to learn a policy, but modern algorithms introduce the idea of the *growing batch*. The growing batch learning consists of altering between the exploration and learning phases and increasing the batch of transitions using learned intermediate policies (Ernst, Geurts, and Wehenkel 2005; Lange, Gabel, and Riedmiller 2012).

The model-free online methods in problems with a large or continuous spaces are limited, and there are problems like slow learning, inefficiencies for stochastic approximation and stability issues related to function approximation (Wiering and Otterlo 2012). So the batch RL algorithms overcome theses issues introducing in their algorithms ideas like experience replay (Lin 1992), supervised learning in order to fit the value function approximation (Gordon 1995) and use efficient stochastic approximation (Ormoneit and Sen 2002). In general, the batch RL algorithms have three phases: the exploration and collecting data, policy learning and applying the learned policy (Wiering and Otterlo 2012). The most popular algorithms of batch RL are fitted Q-iteration and least squares policy iteration.

Fitted Q-iteration (FQI) (Ernst, Geurts, and Wehenkel 2005) is an algorithm based on VI, and it can be considered a version of Q-Learning of batch RL. The approximators that can be used are non-linear and non-parametric. Instead of applying the update rule on all pairs (x, u), only applies on the pairs contained in the set of experiences. Subsequently, the set of updated values is used as a training set for a regression method. The result of this regression is a rough estimate of the Q-function in the entire state space. FQI convergence is guaranteed only for those approximators that perform a non-expansive mapping between their input and output space (Ernst, Geurts, and Wehenkel 2005) such as K-Nearest Neighbours regression, linear interpolation, and kernel averaging, among others. Additionally, using neural networks to approximate value functions have obtained successful results in real-world problems with the algorithm *Neural Fitted Q-Iteration* (Riedmiller 2005; Riedmiller, Montemerlo, and Dahlkamp 2007) or using deep neural networks with the algorithm *Deep Fitted Q-Iteration* (Lange and Riedmiller 2010). The implementation and further description can be found at (Ernst, Geurts, and Wehenkel 2005; Szepesvári 2010; Wiering and Otterlo 2012).

The *least squares policy iteration* (LSPI) algorithm (Lagoudakis and Parr 2003) has the corresponding steps of policy evaluation and improvement of the classical PI algorithm (Sutton and Barto 2018). The evaluation step is an extension to Q-function of the algorithm *least squares temporal difference* (LSTD) (Bradtke and Barto 1996; Boyan 2002). The objective of LSTD is to find V-functions; therefore, the MDP model is known. In LSPI, the idea of finding a value function by directly solving a system of equations is extended to the case of Q-functions by LSTD-Q (Lagoudakis and Parr 2002). When this method of evaluating policies is combined with a stage of policy improvement, the resulting algorithm, known as LSPI, converges asymptotically towards the optimal policy.

3.5 Deep reinforcement learning

In recent years, the use of deep neural networks has increased in reinforcement learning. The main algorithms that combine deep NN and reinforcement learning are Deep Q-Networks (DQN) (Mnih et al. 2015) that uses Convolutional NNs and successfully applied to vision problems in Atari games. There are many variations of DQN algorithm like Double DQN (Van Hasselt, Guez, and Silver 2016), Double DQN with proportional prioritisation (Schaul et al. 2015). The previous algorithms use a generic neural network for regression of the Q-function, Dueling Network Architectures (Wang et al. 2015) implement a deep architecture with the use of the called Advantage function. The work in (Hessel et al. 2018) presents a comparative analysis of some extensions of DQN algorithms and their combinations.

3.6 Linear programming approach to ADP

DP and ADP can be posed as a linear optimisation problem. LP can, indeed, be used to get solutions for both *exact* DP in discrete state and input spaces (Manne 1960; Sutton and Barto 2018) and *approximate* DP with large dimensions or continuous variables (De Farias and Van Roy 2003; Wang, O'Donoghue, and Boyd 2015; Beuchat, Georghiou, and Lygeros 2016; Beuchat, Georghiou, and Lygeros 2020). Linear programming is a well-known optimisation tool that makes it suitable for use in ADP problems.

The LP approaches are based on two main properties (Busoniu et al. 2018) *monotonicity* and *value iteration convergence*. The authors of (De Farias and Van Roy 2003) prove that, any $V(x,\theta)$ fulfilling (3.14) is a lower bound of the optimal cost function, i.e., $V(x,\theta) \leq V^*(x)$ in the case $\gamma < 1$; indeed, T is a monotonic and contractive operator, so it fulfils the value iteration convergence $V \leq TV \leq T^2V \leq \cdots \leq T^\infty V = V^*$. The problem to solve is:

$$\begin{aligned} &\max EV \\ &\quad \text{s.t.} \quad V \leq TV \end{aligned} \tag{3.13}$$

In ADP, to avoid PI/VI convergence problems, in (De Farias and Van Roy 2003) authors provide a lower bound estimation of the value function that verifies:

$$V(x,\theta) \leq L(x,u) + \gamma V(f(x,u),\theta) \quad \forall (x,u) \in \mathbb{X} \times \mathbb{U} \tag{3.14}$$

The equation (3.14) is equivalent to:

$$V(x,\theta) \leq \min_{u \in \mathbb{U}} \left(L(x,u) + \gamma V(f(x,u),\theta)\right) \tag{3.15}$$

i.e., $V(x,\theta) \leq TV(x,\theta)$ for all $x \in \mathbb{X}$, which is consistent with $V^*(x) = TV^*(x)$, but replacing equalities by inequalities. As $\mathbb{X} \times \mathbb{U}$ is finite, we can write (3.14) as a finite set of inequalities.

If the function approximator is linear in parameters, then maximisation of a weighted average of V over the state space, subject to (3.14), can be solved as an LP problem because the inequality constraints can be written as:

$$\Big(\phi^T(x) - \gamma\phi^T(f(x,u))\Big) \cdot \theta \leq L(x,u) \quad \forall (x,u) \in \mathbb{X} \times \mathbb{U} \tag{3.16}$$

Note that, we are working with value function approximation so the base functions (BFs) for approximation can be, e.g. Gaussian neurons, triangular neurons, etc. The constraint inequality (3.15) represents the relaxation on the Bellman equation. In (Wang, O'Donoghue, and Boyd 2015) use the iterative Bellman inequalities $V \leq T^K V$,$K > 1, K \in \mathbb{N}$ instead of $V \leq TV$ in order to improve the results obtained using only a single Bellman inequality condition (De Farias and Van Roy 2003). In this context, LP approaches can be extended to Q-functions (Beuchat, Georghiou, and Lygeros 2016), online variants of equation (3.13) (De Farias and Van Roy 2003; Wang, O'Donoghue, and Boyd 2015; Beuchat, Georghiou, and Lygeros 2016; Beuchat, Georghiou, and Lygeros 2020) and model-based ADP (Beuchat, Georghiou, and Lygeros 2020).

3.7 Summary

In this Chapter, an extension of the tabular DP/RL methods is presented. The algorithms described in this Chapter using value function approximation techniques. The use of approximation techniques in order to represent the value function allows increasing the applicability of the DP/RL methods to problems with large or continuous state and action spaces. Additionally, the function approximation presents a compact representation using a few parameters in order to represent a complete value function.

The function approximation can use parametric and nonparametric methods. The two methods have a parameter vector that needs training; the main difference is that the parametric methods state the parameter vector previously, and it is independent of the samples. On the other hand, in the nonparametric methods, the parameter vector depends on the samples, that means that the size of the vector is variable. The use of the two methods has successful applications in approximate dynamic programming.

The classical algorithms of the DP/RL in exact approach can be extended in continuous spaces. The representative algorithm of policy iteration using value function approximation is the Sarsa algorithm, and in the case of value iteration is the Q-learning algorithm. The approximation process usually minimises a cost function, in order to minimise several methods can be used, i.e., gradient descent methods.

In order to improve the data-efficient, the batch algorithms are used. The main idea behind these algorithms is to use a fixed or growing batch in the exploration stage and supervised learning in order to fit the value function. The fitted Q-iteration and least-squares policy iteration algorithms are the main algorithms that use the batch idea.

The use of the deep neural networks in combination with reinforcement learning arises in the last years, the most popular algorithm with this approach is Deep Q-Networks. There are some variations of these algorithms that have successful application in the vision problems.

The use of the linear programming approach in adaptive dynamic programming is presented in the last part of this Chapter. The main idea is to relax the Bellman equation in order to compute a lower bound of the optimal value function. Applying this approach the avoid the use of the classical iterative algorithms.

Chapter 4

Policy Search

The policy search methods are a part of the RL methods to find the policy directly optimising in the policy space. These methods can be applied to high-dimensional state and action spaces. In a robot learning context, policy search methods can be classified in model-free and model-based methods. Model-free methods use exploration strategies in order to get sampled trajectories, with the dataset, the policy applied is evaluated and improved. On the other hand, methods based on models have a previous stage to identify the model and with that model obtain data through simulation.

The contents of Section 4.4 are based on publications [3, 7] listed on page 4 in which the author has collaborated.

4.1 Introduction

The applications of electromechanical systems and specifically robotic systems nowadays are very diverse, for example, robots in the automotive industry, domestic robots, robots for the medical purpose, among others. The specific tasks performed by these systems are programmed taking into account certain constraints and assumptions, but if conditions or the environment change it is necessary to modify the program or even make a new program, but using learning techniques the system, i.e., a robot, can be adapted to these new conditions autonomously.

In this context, the aim of reinforcement learning (RL) techniques is to solve sequential decision problems (Busoniu et al. 2010; Lewis and Liu 2013). For example, in previous chapters this aim is to get using value functions and the Bellman equations (Sutton and Barto 2018). Policy Search (PS) methods can achieve the same objective. PS is a subfield of RL which focuses on finding the parametrised optimal policy π_θ^* , where $\theta \in \mathbb{R}^n$, optimising directly on parameter space without computing the value function, thus finding an optimal policy π_θ^* is equivalent to finding the optimal vector of parameters θ^* (Deisenroth, Neumann, Peters, et al. 2013; Busoniu et al. 2018; Sigaud and Stulp 2019).

The parametrised policies have some advantages like they can permit to use prestructured policies using configurations like dynamic movement primitives (DMP), imitation learning, central pattern generators among other representations that allow having robustness and stability for coping robot learning problems (Schaal et al. 2005; Endo et al. 2008; Heidrich Meisner and Igel 2009; Kober et al. 2010). Additionally, policy search methods have better convergence properties, the task can be conditioned the policy subspace, and the exploration can be directly controlled. On the other hand, the main disadvantages are related to the convergence to a local optimum and the data inefficient and high variance.

PS can be divided into model-free and model-based methods. Model-free PS methods are the most widely used, and they use sampled trajectories and immediate rewards to learn the policy without the model of the process (Deisenroth, Neumann, Peters, et al. 2013; Kober and Peters 2016). On the other hand, model-based use the sampled trajectories to build a model initially, evaluate and learn the policy (Abbeel, Quigley, and Ng 2006; Deisenroth, Neumann, Peters, et al. 2013). Model-free methods are used more frequently than model-based ones in the robotics field because learning policy is often easier than learning a model of the robot and its environment (Deisenroth and Rasmussen 2011). PS methods have been successfully applied not only in robotics but also in many scientific fields such as medicine, economy or artificial intelligence (Kober, Bagnell, and Peters 2013; Lewis and Vrabie 2009).

The structure of the Chapter is as follows: Section 4.2 describes model-free policy search methods. Model-based methods are presented in Section 4.3. Section 4.4 summarises the main findings of the work in (Pastor et al. 2018). Finally, a summary is drawn in Section 4.5.

4.2 Model-free policy search

Model-free methods, as their name indicate, do not need the system model in order to find an optimal policy (Peters and Schaal 2006; Kober and Peters 2009). These methods learn the optimal policy using a dataset that consists of sampled trajectories. The iterative steps of the model-free PS methods are exploration, policy evaluation and policy update (Deisenroth, Neumann, Peters, et al. 2013). The exploration is to gather the samples of the system, and its strategies generate trajectories using a policy. The next step is the policy evaluation. The policy applied generated a reward; this reward is used to know the quality of the policy in the policy evaluation step. After the policy evaluation, an essential step is the the policy update. The policy updated calculates the new parameters of the policy based on the policy evaluation step. These steps are repeated iteratively until the policy converge (Deisenroth, Neumann, Peters, et al. 2013).

4.2.1 Exploration and evaluation

The objective of the exploration is taking samples and generating trajectories (Rückstieß, Felder, and Schmidhuber 2008; Kober and Peters 2009). There are several ways to explore; for example, adding an exploration noise to the action control is possible to explore the action space (Baxter and Bartlett 2000; Peters, Vijayakumar, and Schaal 2003). In contrast, if the perturbation is applied in the parameter vector θ, the exploration is done in the parameter space (Kober and Peters 2009; Rückstieß, Felder, and Schmidhuber 2008). The exploration strategies in the action or the parameters spaces are generalised in the hierarchical setting of learning upper-level, and lower-level polices (Kober and Peters 2009; Theodorou, Buchli, and Schaal 2010). For example, consider the following nonlinear stochastic system:

$$x_{t+1} \sim p(x_{t+1}|x_t, u_t) \tag{4.1}$$

where the control actions are computed by a low-level deterministic policy defined as $u_t := \pi_\theta(x_t)$. When the control action u_t is applied the system changes from the state x_t to the state x_{t+1} using the probability of the transition $p(x_{t+1}|x_t, u_t)$. The lower-level policy is parametrised by θ. So, the aim of the model-free PS methods is to find the parameter vector θ such that maximises J_θ using the available sampled trajectories.

$$J_\theta := \mathbb{E}_{p_\theta}[R(\tau)] = \int R(\tau)p_\theta(\tau)d\tau \tag{4.2}$$

where $R(\tau)$ is the accumulated reward for a system trajectory $\tau := \{x_{0:T}; u_{0:T}\}$, $p_\theta(\tau) := \prod_{t=0}^{T-1} p(x_{t+1}|x_t, \pi_\theta(x_t))$ is the probability density function and $\mathbb{E}$ is the expectation operator.

In the upper-level case, the parameter vector for the low-level policy π_θ is sampled from the upper-level policy as $\theta \sim \pi_\omega(\theta)$, and the optimisation problem for learning upper-level policies $\pi_\omega(\theta)$ is the maximisation of:

$$J_\omega := \mathbb{E}_{\pi_\omega}[R(\theta)] = \int_\theta \pi_\omega(\theta) R(\theta) d\theta \tag{4.3}$$

Additionally, the exploration strategies can apply the exploration noise at the beginning of each experiment/trajectory and evaluated in each episode (episode-based) or considering each time step (step-based) or even considering additional features like correlated and uncorrelated exploration when the policies are represented with Gaussian distributions (Stulp and Sigaud 2012). The different configurations of exploration essentially seek to perform efficient exploration that is essential in policy search.

The policy evaluation uses the dataset in order to asses the quality of the policy used to explore (Deisenroth, Neumann, Peters, et al. 2013). The evaluation can be step-based or episode-based. The step-based assesses the quality of the policy in single state-action, and in order to avoid the bias, the Monte-Carlo estimations are used (Peters and Schaal 2006; Williams 1992a). Algorithms using step-based evaluation include REINFORCE, GPOMDP, NAC, eNAC, PoWER, PI^2. The episode-based methods assess the quality of the parameter vector θ, the main algorithms that use this evaluation are PEPG, NES, CMA-ES, RWR, and episode-based versions of the REPS, and PI^2. The choice of the type of evaluation depends on the problem we are solving.

4.2.2 Update the policy

The policy update uses the outcome of the policy evaluation in order to improve the policy used to explore. The main methods to update the policy are policy gradient methods, expectation-maximisation, information-theoretic, path integral theory, among others (Deisenroth, Neumann, Peters, et al. 2013).

Policy gradient methods

Policy gradient methods use the gradient ascent to maximise the return. So, the update the parameter vector using the gradient update is:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J_\theta \tag{4.4}$$

There are many ways to estimate the gradient $\nabla_\theta J_\theta$, i.e., likelihood policy gradients (Williams 1992a), natural gradients (Amari 1998). α is a step-size parameter. In general, the learning process is stable with a smooth policy update. For instance, in the upper-level policy the gradient estimation is computed in order to update:

$$\omega_{i+1} = \omega_i + \alpha \nabla_\omega J_{\omega_i} \tag{4.5}$$

being the sub-index i represents an algorithm iteration and α is defined as a gradient-step update parameter. The policy update uses the gradient estimation to improve (4.3), and the gradient can be computed using likelihood-ratio policy gradients:

$$\begin{aligned}\nabla_\omega J_\omega &= \int \nabla \pi_\omega(\theta) R(\theta) d\theta \\ &= \int \pi_\omega(\theta) \nabla_\omega \log \pi_\omega(\theta) R(\theta) d\theta \\ &= \mathbb{E}_{\pi_\omega}[\nabla_\omega \log \pi_\omega(\theta) R(\theta)]\end{aligned} \tag{4.6}$$

Additionally, the baseline b is computed in order to reduce the variance of the gradient estimation (Williams 1992a; Deisenroth, Neumann, Peters, et al. 2013). For detail about variance reduction, see (Greensmith, Bartlett, and Baxter 2004). This approach is known as Parameter Exploring Policy Gradient (PEPG) method (Sehnke et al. 2010) and the resulting gradient to be computed an optimal baseline:

$$\nabla_\omega J_\omega = \mathbb{E}_{\pi_\omega}[\nabla_\omega \log \pi_\omega(\theta)(R(\theta) - b)] \tag{4.7}$$

where the other h-th element of the baseline can be obtained as:

$$b_h := \frac{\mathbb{E}_{\pi_\omega}[(\nabla_{\omega_h} \log \pi_\omega(\theta))^2 R(\theta)]}{\mathbb{E}_{\pi_\omega}[(\nabla_{\omega_h} \log \pi_\omega(\theta))^2]} \tag{4.8}$$

After the gradient update, a new exploration is performed to obtain new data and recalculate the policy distribution, this process is repeated throughout the learning process.

The most relevant algorithms that use the likelihood-ratio are REINFORCE (Williams 1992b) and GPOMDP/Policy gradient theorem (Sutton et al. 2000; Baxter and Bartlett 2001). In addition, by using the past data efficiently through importance sampling (IS) methods, the estimation of the expected return can be improved (Jie and Abbeel 2010; Pastor et al. 2018).

The step size is an essential issue in the likelihood-ratio methods because the gradient is a local approximation. For instance, if the step is too far, the objective does not improve, or even it might do worse. So, an efficient gradient is the natural gradient (Amari 1998). The natural gradient corresponds to the most significant ascent in the policy space and not in the parameter space with the proper step size. The main idea of the natural gradients in the policy search method is to limit a step-width in the trajectory distribution (Deisenroth, Neumann, Peters, et al. 2013). The main algorithms that use the approach are Natural Actor-Critic (NAC), episodic NAC, for details see, e.g., (Peters and Schaal 2008a; Peters and Schaal 2008b; Wierstra et al. 2008; Sun et al. 2009).

A simple step-sizing would be to do a line search in the direction of the gradient. The line search tries different step-sizes and sees the performs in the roll-outs and then take the best one. An advanced technique to define the step-sizing is through of *trust regions*. Trust Region Policy Optimisation (TRPO) (Schulman et al. 2015b) maximise parameters that modify the policy increasing advantage for old policy in order to avoid too large step size. Inspired in TRPO, a proximal policy optimisation (PPO) (Schulman et al. 2017) simplifies the computational load.

Expectation-maximisation approach

Another way to update the policy is by using the expectation-maximisation (EM) algorithm. EM-algorithm avoids slow convergence problems and unstable learning (Deisenroth, Neumann, Peters, et al. 2013). The idea for using the EM-algorithm is to pose the PS problem as an inference problem. There are some variations of the applications of these algorithms that are most efficient as Monte-Carlo expectation maximisation, that can use considered episode-based or step-based.

Information-theoretic approach

The main idea behind the use of the information-theoretic approach is that the policy updated should be kept close to the dataset (Deisenroth, Neumann, Peters, et al. 2013). This approach ensures that the policy update does not move away from the data of the previous policy by delimiting the distance between the current and new policy distributions. The representative algorithm that uses this approach is the Relative Entropy Policy Search (REPS) (Peters, Mulling, and Altun 2010). REPS pose the policy search problem as an optimisation problem where the distance measure used is the Kullback-Leibler divergence in order to bound the distance between the new and the old policy distributions. Initially, the REPS was posed in a step-based formulation (Peters, Mulling, and Altun 2010), but it was extended to others formulations like episode-based (Peters, Mulling, and Altun 2010), episode-based to multiple contexts (Daniel, Neumann, and Peters 2012) and learning multiples solutions (Kober and Peters 2009).

Stochastic optimisation

Other ways to compute the gradient is using parameter perturbation or evolutionary strategies. For example, the simplest way to compute the gradient is to use finite difference methods; the main idea is to compute the gradient perturbing the parameter vector θ with positive and negative disturbances (Kim et al. 2004). Versions of evolutionary strategies are Cross-Entropy methods (Rubinstein and Kroese 2013); these methods have been successfully applied games (Szita and Lörincz 2006; Gabillon, Ghavamzadeh, and Scherrer 2013) and Covariance Matrix Adaptation (CMA)(Hansen and Ostermeier 1996). There is a whole family of algorithms that have this approach like *Reward Weighted Regression* (RWR) (Dayan and Hinton 1997; Peters and Schaal 2007) ,*Policy Improvement with Path Integrals* (PI2) (Kappen, Wiegerinck, and Broek 2007; Theodorou, Buchli, and Schaal 2010; Stulp and Sigaud 2012) *Covariance Matrix Adaptation Evolutionary Strategy* (CMA-ES) (Hansen and Ostermeier 1996; Hansen, Muller, and Koumoutsakos 2003), *Policy learning by Weighting Exploration with Returns* (PoWER) (Kober and Peters 2009). The main advantage of these methods is a simple and easy way to implement. These methods work with arbitrary parametrisation, even nondifferentiable. Besides, they are easy to parallelise. On the other hand, they are not sampling efficient, and it is the main caveat in problems with high-dimensional space that optimising over. If the simulator is being used, sampled inefficient methods become a minor problem. In general, the stochastic optimisation is a black box; this feature permits to apply in policy search problems. For more details about CMA-ES, we refer to (Heidrich Meisner and Igel 2009).

Actor-Critic Architectures

In the actor-critic architectures, the processes of evaluation and improvement are independent. The critic evaluates the current policy and the actor implements the policy and can be trained with policy gradients methods. The methods with this approach are one step actor-critic, advantage actor-critic (A2C). *Asynchronous Advantage Actor-Critic* (A3C) (Mnih et al. 2016) is a policy gradient method that is a variant of actor-critic that work in the forward view n-step returns to update the policy and the value function. If the idea of eligibility traces (Sutton and Barto 2018) is combined with A3C the algorithm resulting is Generalised Advantage Estimation (GAE) (Schulman et al. 2015a).

In recent years, with the use of deep neural networks, new algorithms have been developed that combine these neural networks with reinforcement learning techniques. For example, deep deterministic policy gradient(DDPG) is an extension of Q-Learning for continuous spaces and follows the gradient in the policy network to increase future reinforcements (Lillicrap et al. 2015). Twin Delayed DDPG (TD3) (Fujimoto, Hoof, and Meger 2018) improves the learning speed and performance of the DDPG by mitigating overestimation using pessimistic Double Q-Learning idea. An algorithm based on the maximum entropy reinforcement learning framework is Soft Actor-Critic (SAC) (Haarnoja et al. 2018).

4.3 Model-based policy search

The model-based PS methods use the dataset to learn a dynamics model of the system, and this model is used to do internal simulations in order to learn an optimal policy π_θ^* without the use of the real system (Deisenroth, Neumann, Peters, et al. 2013). In general, the use of the model-based approach avoids performing a large number of direct iterations on the system; therefore, the iterations are fewer than would be done in the case of the model-free approach (Atkeson and Santamaria 1997). The loop of the model-based PS methods consists in: to learn a model using the real data, to simulate experiments or trials using the learned model, to learn a new policy. Then the learned policy is applied in the real system in order the get a new real dataset, the new dataset is used to improve the estimation of the dynamics model of the system, and these steps are repeated iteratively (Deisenroth, Neumann, Peters, et al. 2013).

The model learning step is essential in this approach. The estimation of the model is not perfect, and the quality of the model affects directly to the learning process (Abbeel, Quigley, and Ng 2006), i.e., degrading policies or large errors in the policy. Therefore, the representations used in order to deal with uncertain models by reducing the effects of the model errors are the probabilistic distributions (Schneider 1997; Deisenroth and Rasmussen 2011; Deisenroth, Rasmussen, and Fox 2011). The most important probabilistic models used in model-based methods in a context of robotics are Locally Weighted Bayesian regression (LWBR) (Cleveland and Devlin 1988) and Gaussian Process (GP) regression (Rasmussen 2003).

The other aspect is the long term prediction using this model. These predictions can be implemented methods like stochastic inferences (Monte-Carlo (MC) method) or deterministic approximation inferences (Deisenroth, Neumann, Peters, et al. 2013). The trajectory sampling using the MC is implemented, i.e., in the PEGASUS (Ng and Jordan 2000) algorithm. The main idea of this algorithm is to change the approach from a stochastic MDP to a modified deterministic MDP (Ng and Jordan 2000; Deisenroth, Neumann, Peters, et al. 2013). This modification approach of the stochastic MDP can permit to reduce the sampling variance. PEGASUS is relative efficiently and can be used even in model-free methods. On the other hand, the most usual deterministic approximation methods used are the linearisation, moment matching, unscented transformation (Deisenroth, Neumann, Peters, et al. 2013).

The policy update can be done with the gradient-free or gradient-based methods. The gradient-free methods (Nelder and Mead 1965) are easy to implement, but the disadvantage is the slow convergence rate. On the other hand, the gradient-based policy update is faster than gradient-free methods, but they need additional computational resources. The main methods to compute the gradient-based are sample-based estimation and analytic computation.

4.4 Learning Upper-Level Policy using Importance Sampling-based Policy Search Method

Policy search methods allow to learn upper-level policies whose main advantage is that these distributions explore directly in the parameter space. In (Pastor et al. 2018) is proposed an algorithm based on importance sampling methods and local linear regression that uses the samples in an efficient way. To achieve this objective, the authors propose to include information of all the past samples in the learning process using importance sampling methods. Additionally, the gradient direction of the linear local model reward to explore regions where the prediction of the reward could be better.

4.4.1 Policy Search via Importance Sampling (IS) and local regression for upper level policy

Considering the system in the equation (4.1), and it is controller by a low-level deterministic policy $u_t := \pi_\theta(x_t)$. The exploration is done in the parameter space (upper-level ones). So, the optimisation problem for learning upper-level policies $\pi_\omega(\theta)$ can be formalised as the maximisation of equation (4.3), repeated here for easy reference:

$$J_\omega := \mathbb{E}_{\pi_\omega}[R(\theta)] = \int_\theta \pi_\omega(\theta) R(\theta) d\theta \tag{4.9}$$

where the parameter vector for the low-level policy $u_t := \pi_\theta(x_t)$ is sampled from the upper-level policy as $\theta \sim \pi_\omega(\theta)$.

It should be noted that the equation (4.6) implies that the evaluation of the reward $R(\theta)$ and its gradient needs multiple roll-outs on each algorithm iteration because it is a stochastic process. This can be costly and the work (Jie and Abbeel 2010) proposes using IS to reuse previous experiments to provide a better estimation of the cost. In this respect, the equation (4.9) can be approximated by re-weighting the cost of past experiments via IS.

Importance Sampling

Using IS methods it is possible to include information of past samples, IS methods are used to estimate the characteristics of a specific distribution using samples generated by different distributions (Thrun, Burgard, and Fox 2005). So, if the objective distribution is f, and the proposed distribution is g the importance weights $q^{[n]}$ are:

$$q^{[n]} = \frac{f(x^{[n]})}{g(x^{[n]})} \tag{4.10}$$

The importance weights are computed for each indexed sample ($n = 1, ..., N$). The $q^{[n]}$ can be interpreted as the importance of each sample in the distribution g on the distribution f.

In this context, the IS sampling idea can be used in the PS methods in order to reduce the number of experiments needed to find an optimal policy. For instance, if the dataset has the following structure:

$$\mathcal{D} = \{\{\theta^{[1]}, R^{[1]}, \pi_{\omega_1}(\theta^{[1]})\}, \dots, \{\theta^{[k]}, R^{[k]}, \pi_{\omega_k}(\theta^{[k]})\}\}, \tag{4.11}$$

the expected accumulated reward using the IS approach and the given dataset (equation (4.11)) is defined as

$$\hat{J}_\omega \approx \frac{1}{\sum_{i=1}^{k} q_\omega^{[i]}} \sum_{i=1}^{k} q_\omega^{[i]} R^{[i]} \tag{4.12}$$

and the importance weights:

$$q_\omega^{[i]} = \frac{\pi_\omega(\theta^{[i]})}{\pi_{\omega^{[i]}}(\theta^{[i]})} \tag{4.13}$$

The upper level policy $\theta^{[i]} \sim \pi_{\omega^{[i]}}(\theta)$ can be modelled like any distribution, i.e., Gaussian distribution $\pi_\omega(\theta) := \mathcal{N}(\theta|\omega)$, with $\omega := [\omega_\mu, \omega_\Sigma]$. The optimal policy is obtained using:

$$\pi_{\omega^*}(\theta) := \underset{\omega}{\operatorname{argmax}}\, \hat{J}_\omega \tag{4.14}$$

In order to properly formulate the optimisation problem, it is desirable to determine a minimum number of effective samples N_{eff} to estimate (4.12) and should be considered during the optimisation process (Kong 1992). Thus the policy update problem statement (4.14) has to be modified to:

$$\pi_{\omega^*}(\theta) := \arg\max_{\omega} \hat{J}_\omega, \text{ s.t. } \mathrm{ESS} := \frac{\left(\sum_{i=0}^{k} q_\omega^{[i]}\right)^2}{\sum_{i=0}^{k} \left(q_\omega^{[i]}\right)^2} \geq N_{eff} \tag{4.15}$$

where N_{eff} is the minimum number of effective samples, set to be a design parameter in the implementation.

The gradient-based optimisation should calculate the gradient update using IS; this can be described through an equation such as the following:

$$\nabla_\omega J_\omega = \frac{1}{\sum_{i=1}^k q_\omega^{[i]}} \sum_{k=1}^k q_\omega^{[i]} [\nabla_\omega \log \pi_\omega(\theta^{[i]})(R^{[i]})] \\ - \frac{1}{\left(\sum_{i=1}^k q_\omega^{[i]}\right)^2} \left(\sum_{k=1}^k q_\omega^{[i]} [\nabla_\omega \log \pi_\omega(\theta^{[i]})]\right) \left(\sum_{k=1}^k q_\omega^{[i]} R^{[i]}\right) \quad (4.16)$$

Cost-surface approximation by local regression

Additionally, it is possible to integrate improvements in the exploration phase to increase the speed of convergence. This improvement can be implemented by integrating cost-surface approximation through local regressions, e.g. Generalised linear models (Hardin and Hilbe 2001), Neural-networks (Haykin 1998), Support Vector Regression Machines (SVRM) (Drucker et al. 1997; Smola and Scholkopf 2004), Gaussian processes regression (GPR) (Rasmussen and Williams 2005) and Locally Weighted Projection Regression (LWPR) (Vijayakumar and Schaal 2000), among others. The idea is to provide a local estimation of the cost-surface using the given dataset $\mathcal{D}$:

$$\hat{R}_{\mathcal{D}}(\theta) \approx R(\theta) \quad (4.17)$$

It should be noted however that the methods mentioned above are mainly designed to perform a regression considering the whole cost-surface, while we are mostly interested in predicting the local behaviour at the point where the maximisation finishes (our best belief of the most appropriate policy for the given data). Thus, our choice of using local linear regression (LLR) appears to be reasonable in terms of just using with the minimum set of necessary points to provide a reliable estimation.

The LLR is a nonparametric method to approximate nonlinear functions. One characteristic of the nonparametric methods is that try to fit the function as much as possible with the available data or a subset of the data. The way in what we use this approximation is:

1. Solve (4.15), being $\omega^* := [\omega_\mu^*, \omega_\Sigma^*]$ the best upper-level policy parameters found with IS.

2. Compute the Mahalanobis distance $\mathcal{M}_\omega(\theta) := (\theta - \omega_\mu)^T \left[\omega_\Sigma\right]^{-1} (\theta - \omega_\mu)$ from each parameter in the dataset $\mathcal{D}$ to the actual upper-level policy parameters

found with IS and create a list $\mathcal{H}$ with the parameters satisfying the following criteria:

$$\mathcal{H} := \left\{ k \in \mathbb{Z}^{+} \middle| \mathcal{M}_{\omega^*}(\theta^{[k]}) < h \right\}, \tag{4.18}$$

with h the selection criterion, typically $h = 9$ to consider only points lying within the $\pm 3\sigma$ of the current policy variance.

3. Compute a linear model using least squares method:

$$\hat{R}_{\omega_\mu^*} := \xi \omega_\mu^*, \tag{4.19}$$

with,

$$\xi = \mathcal{I}_Y \mathcal{I}_X^T \left(\mathcal{I}_X \mathcal{I}_X^T \right)^{-1}, \tag{4.20}$$

and

$$\mathcal{I}_X = \begin{bmatrix} \theta^{[\mathcal{H}_1]} & \theta^{[\mathcal{H}_2]} & \dots & \theta^{[\mathcal{H}_m]} \\ 1 & 1 & \dots & 1 \end{bmatrix} \tag{4.21}$$

$$\mathcal{I}_Y = \begin{bmatrix} R^{[\mathcal{H}_1]} & R^{[\mathcal{H}_2]} & \dots & R^{[\mathcal{H}_m]} \end{bmatrix}, \tag{4.22}$$

being m the number of indices in $\mathcal{H}$. An illustrative example of this step is given in Figure 4.1.

4. Update the mean of the policy based on the gradient of $\hat{R}_{\omega_\mu^*}$:

$$\omega_\mu^* \leftarrow \omega_\mu^* + \alpha_R \nabla_{\omega_\mu^*} \hat{R}_{\omega_\mu^*} = \omega_\mu^* + \alpha_R \xi, \tag{4.23}$$

where α_R is a gradient-step update parameter. Note that the gradient being now evaluated, i.e., that of $\hat{R}_{\omega_\mu^*}$ is different (regression-based) to the gradient used in the importance-sampling maximisation of the expected reward (4.16). This combination of two different gradients is the key ingredient of our proposal.

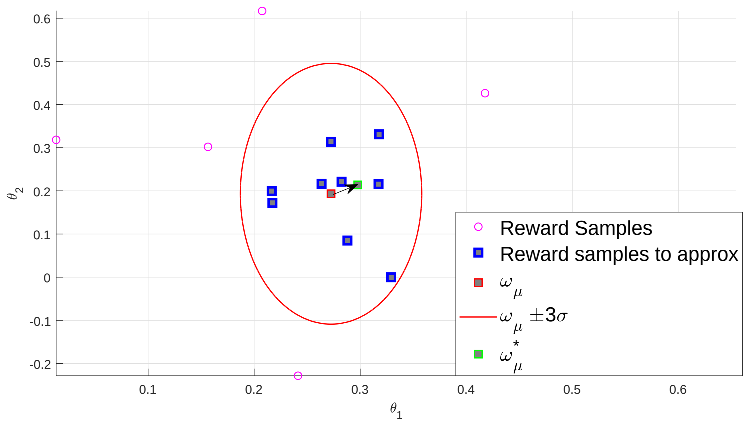


Figure 4.1: Two-dimensional example of local linear regression. The linear local regression uses samples of past rewards that are closed to our interest point(ω_μ). The gradient of local linear model is used to calculate the mean of a new policy(ω^*_μ).

Proposed algorithm

The inputs in the Algorithm 1 are: initial policy (π_{ω_0}) and initial state (x_0) to run the experiments. The main parameter of the algorithm is the effective number of samples (N_{eff}), while an additional parameter h is required which is fixed to $h = 9$ to consider only points lying within the $\pm 3\sigma$ boundaries of the IS computed upper-level Gaussian policy.

Numerical Example

In a linear second-order discrete-time model given $x_{t+1} = Ax_t + Bu_t$ with

$$A = \begin{bmatrix} 0.9 & 0.1 \\ 0 & 0.8 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The control problem will be stated using the quadratic instantaneous reward $r_t = x_t^T H_x x_t + u_t^T H_u u_t$ with:

$$H_x = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix} \qquad H_u = \begin{bmatrix} 2 \end{bmatrix}$$

Algorithm 1 Learning Upper-Level Policy using Importance Sampling-based Policy Search Method

```
1: Input: $\pi_{\omega^{[0]}}, x_0$
2: Parameters: $N_{eff}, h = 9, \alpha_R$
3: for $i = 0$ to Max Iterations do
4:     Sample a new low-level policy parameter $\theta^{[i]} \sim \mathcal{N}(\omega_\mu^{[i]}, \omega_\Sigma^{[i]})$ and run a trial under the policy $\pi_{\theta^{[i]}}$ starting from $x_0$ and obtain $R^{[i]}$.
5:     if $i > N_{eff}$ then
6:         Solve (4.15) to compute a new upper-level policy $\pi_{\omega^*}(\theta)$ maximising $\hat{J}_\omega$ satisfying ESS $\geq N_{eff}$, by using gradient ascent (4.16) until the ESS bound is hit.
7:         Compute $\mathcal{H} := \left\{k \in \mathbb{Z}^+ \middle| \mathcal{M}_{\omega^*}(\theta^{[k]}) < h \right\}$.
8:         Compute $\xi$ based on (4.20) and $\mathcal{H}$.
9:         Update policy mean $\omega_\mu^* \leftarrow \omega_\mu^* + \alpha_R \xi$
10:        Update policy $\omega^{[i+1]} \leftarrow \omega^*$
11:    else
12:        Update policy $\omega^{[i+1]} \leftarrow \omega^{[i]}$
13:    end if
14: end for
```

A linear parameterised policy with parameter $\theta \in \mathbb{R}^2$ is used, given by $u_t = -\theta x_t$. The initial state $x_0 = [10 \quad 10]^T$, and return function as an exponential transformation $R^{[i]} = 10^4 \cdot exp(\beta \sum r_t)$ with $\beta = 0.05$. The high-level policy is defined as:

$$\theta \sim \mathcal{N}\left(\begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}, \begin{bmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{bmatrix}\right) \tag{4.24}$$

where the adjustable parameters are $\omega_\mu := (\mu_1, \mu_2)$ and the variance-related ones $\omega_\Sigma := (\sigma_1, \sigma_2)$. The initial Gaussian policy is defined as $\mu_1 = 0.1$, $\mu_2 = 0.25$, $\sigma_1 = \sigma_2 = 0.2$. The settings of the algorithm are $N_{eff} = 3$, reward step $\alpha_R = 1.8 \times 10^{-1}$. Additionally, we added 1% white noise to the reward function when sampling in the data simulation. The learning algorithm cannot make use of the simulation dynamics except by gathering trials.

Each algorithm has been executed 20 times, in each execution, 100 iterations (tests) have been performed. Thus, it is possible to determine the influence of noise on the system and the variability in the algorithms. The maximum, minimum and mean values in each experiment have been calculated for both policy

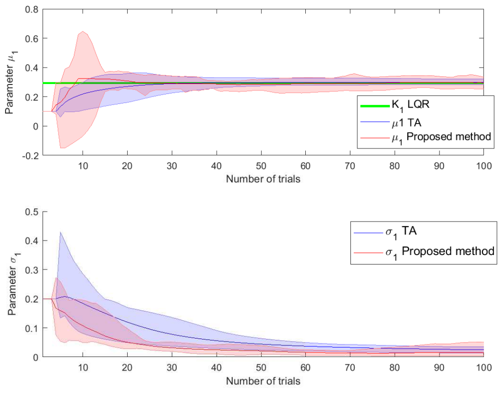


Figure 4.2: Controller exploration parameters $\pi_{1\omega} = \mathcal{N}(\mu_1, \sigma_1^2)$. The algorithms considered are the Tang and Abbel algorithm (TA), our approach (Proposed method) and the optimal LQR(OPT).

and average reward. Figure 4.2 and Figure 4.3 represent the evolution of the exploration parameters in the parameter space of the policy. Figure 4.4 represents the average reward obtained by number of trials. The evaluation included comparison between results of this proposed algorithm and the Tang and Abbel algorithm (Jie and Abbeel 2010) in upper-level version.

The shaded region in all figures depicts the max and min values collected from the 20 runs of each algorithm. The proposed algorithm improved the speed of convergence to the optimal values with the same quantity of samples. The outcome of simulation lead to the conclusion that the extrapolation of the reward permitted to explore areas with the highest reward.

4.5 Summary

The policy search methods solve the RL problem, that is maximising the cost in a long period of time. The PS methods find the policy directly optimising in the parameter space. It permits to extended to high-dimensional or continuous problems. The PS methods can be classified considering if they need or not the model. In the case that PS methods do not use the model are called model-free PS methods. The model-free PS methods have three iterative steps: the exploration, policy evaluation and policy update. The exploration step gathers the sampled trajectories; the exploration strategies usually use the exploration

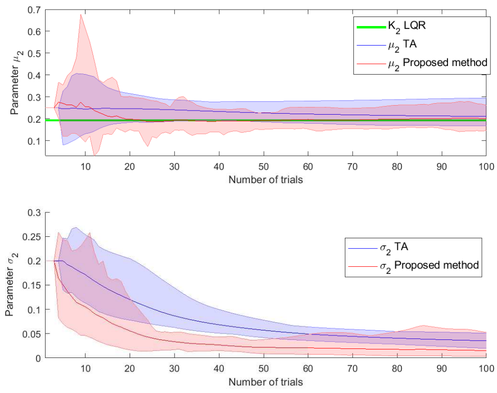


Figure 4.3: Controller exploration parameters $\pi_{2\omega} = \mathcal{N}(\mu_2, \sigma_2^2)$. The algorithms considered are the Tang and Abbel algorithm (TA), our approach (Proposed method) and the optimal LQR(OPT).

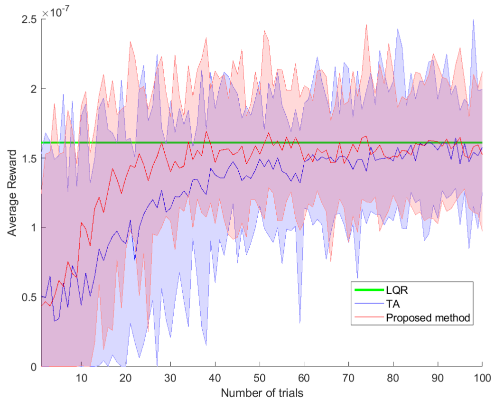


Figure 4.4: Performance evaluation on 2nd order linear system the algorithms considered are the Tang and Abbel algorithm (TA), our approach (Proposed method) and the optimal LQR(OPT).

noise in the action space or in the parameter space. The policy evaluation step provides information about the quality of the policy used in the exploration step. The exploration and evaluation of the policy can be episode-based or step-based. The crucial step in the PS methods is the policy update, which can be done using a simple method like a finite difference to estimate the gradient or using more complex methods like stochastic optimisation. The steps are repeated iteratively until the policy converges.

Model-based PS methods find the optimal policy, but they need a model of the system. The model estimation is obtained using the dataset collected from the interactions with the real system. The model estimation of the system is an essential step in the learning process in these kinds of algorithms. The model is not perfect, and the errors in the model affect directly in the learning process. In the initial stage, the dynamics of the model is learned. This model is used to simulate experiments or trials of the system. These experiments are evaluated and used to update the policy. Finally, the policy is used in the real system in order to get more real data that are used to refine the model estimation and is repeated continuously until the policy do not change. These methods are data-efficient.

Finally, the results obtained from a proposed algorithm inspired in the use of IS methods and model-free PS methods in upper-level policies are presented. The use of the IS approach and the LLR cost approximation improved the speed convergence of a proposed policy search algorithm at optimal values. The IS method allowed us to use information from past samples to estimate a reward model that can predict regions where the reward may be high. Information from these regions makes it possible to propose a new exploration policy. As a result, more efficient data management can be achieved, which is essential for robotic learning tasks, where trials are expensive or difficult to achieve.

The family of the algorithms of PS methods have a lot of successful applications in the robots' systems.

Part II

Contributions

Chapter 5

Fitted Q-function Control Methodology based on Takagi-Sugeno Systems

This chapter presents a combined identification/Q-function fitting methodology, which involves identification of a Takagi-Sugeno model, computation of (sub)optimal controllers from Linear Matrix Inequalities, and subsequent data-based fitting of the Q-function via monotonic optimisation. The LMI-based initialisation provides a conservative solution but it is a sensible starting point to avoid convergence/local-minima issues in raw data-based fitted Q-iteration or Bellman residual minimisation. An inverted-pendulum experimental case study illustrates the approach.

The contents of this chapter appeared in the journal article:

- H. Díaz, L. Armesto, and A. Sala (2018). "Fitted Q-Function Control Methodology Based on Takagi-Sugeno Systems". In: IEEE Transactions on Control Systems Technology, vol. 28, no. 2, pp. 477-488, March 2020. ISSN:2374-0159. DOI: 10.1109/TCST.2018.2885689.

5.1 Introduction

Optimal controllers minimising an infinite-time cost index are of interest in many fields. Dynamic programming (DP) (Bertsekas 2017; Zhang et al. 2012) and reinforcement learning (RL) (Sutton and Barto 2018) are powerful paradigms to obtain them, used in many applications (Wei et al. 2015; Pomprapa, Leonhardt, and Misgeld 2017). DP and RL pursue computing value functions $V(x)$, or action-value functions $Q(x, u)$ (Sutton and Barto 2018) in order to yield optimal controllers (also denoted as policies[1]).

Policy iteration (PI) and value iteration (VI) are widely-used techniques to iteratively compute such optimal value functions and associated policies (Lewis and Vrabie 2009; Lewis, Vrabie, and Vamvoudakis 2012; Lewis and Liu 2013). Based on the principles of fixed-point theorem, PI and VI converge to the optimal value and policy under mild conditions ensuring a contraction mapping (Powell 2011; Busoniu et al. 2010). However, these mild conditions are actually so only in systems with a finite number of states and control actions. In continuous-valued settings, some approximation is needed to map a maybe complex controller/value function; the exact Bellman equation gets now converted to a Bellman *residual minimisation* problem (Antos, Szepesvári, and Munos 2008) (also called Bellman *error* in (Sutton et al. 2009)) . Such approximation may spoil the contractive nature of the iteration steps and, hence, convergence may be lost (Fairbank and Alonso 2012). Only in some quite restrictive settings, for instance, fuzzy Q-iteration (Busoniu et al. 2010) convergence can be guaranteed.

A way to avoid PI/VI divergence is trying to minimise the Bellman error via gradient descent (Munos, Baird, and Moore 1999; Baird and Moore 1999; Geist and Pietquin 2013) or other monotonic optimisation methods (Lagarias et al. 1998). However, such methods may get caught on local minima if not properly initialised.

In a model-based approach to optimal control, many non-linear systems can be modelled as the so-called fuzzy Takagi-Sugeno (TS) systems (Takagi and Sugeno 1985), via the well-known sector-nonlinearity approach (Tanaka and Wang 2001). These TS models express the nonlinearity as a convex combination of linear systems. Some convex conditions on the vertex linear models can easily obtain quadratic value function bounds by solving the so-called linear matrix inequalities (LMIs). The LMIs were introduced by the seminal book (Boyd et al. 1994) and

[1]Another option to look for optimal controllers is the so-called “policy search” techniques, in which value functions are avoided and only its gradient is estimated with respect to some controller parameters. These techniques are out of the scope of this work, see (Deisenroth, Neumann, Peters, et al. 2013) and references therein for ample detail.

exploited for (sub)optimal nonlinear control in many works, for instance (Tanaka and Wang 2001; Wu and Cai 2004; Ariño, Pérez, and Sala 2010; Guerra, Sala, and Tanaka 2015) and references therein. Nonlinear optimal control using linear-like techniques can be also approached via Jacobian linearisation at several points (Armesto et al. 2015); nevertheless, these developments are related to predictive control, and out of the scope of our proposal.

The objective of this chapter is to propose a methodology for nonlinear optimal control applications bridging the DP/RL and the model-based LMI approaches: given a nonlinear system in Takagi-Sugeno form, the LMI solution and the fuzzy controller structures associated to them inspire a particular parametrisation of the Q-function so that fitting algorithms can be initialised with the LMI solution. Monotonic/gradient-descent setups from such an initial solution provide controllers with lower Bellman residual than the parameters they were initialised at, without the risk of divergence of traditional PI/VI (PI/VI can also be tested under the proposed parametrisation). Preliminary versions of some ideas in this respect appear in (Díaz, Armesto, and Sala 2016; Díaz, Sala, and Armesto 2017). In a somehow similar philosophy, the paper (Radac, Precup, and Roman 2017) also bridges control-theoretic virtual-reference tuning and Q-learning.

The structure of the chapter is as follows: Section 5.2 introduces necessary preliminaries of the chapter and states the problem. Section 5.3 describes our methodology proposal. Section 5.4 discusses the proposals on Takagi-Sugeno model identification and guaranteed-cost LMI controllers. Section 5.5 proposes a fuzzy Q-function parametrisation arising from the LMIs and an improved one-step controller. The Q-function parametrisation is generalised in Section 5.6. Section 5.7 discusses the generic optimisation approach to temporal-difference error minimisation and presents a summary of previous ideas onto a methodology proposal. Section 5.8 evaluates such methodology both in a simulation case study and an experimental inverted pendulum setup. Some conclusions are given in Section 5.9.

5.2 Preliminaries and problem statement

This chapter will consider nonlinear discrete-time systems

$$x_{t+1} = f(x_t, u_t), \tag{5.1}$$

with $x_t \in \mathbb{X} \subset \mathbb{R}^{n_x}$ and $u_t \in \mathbb{U} \subset \mathbb{R}^{n_u}$ being the model validity and input constraint region, where n_x and n_u are the number of states and inputs, respectively.

Without loss of generality, we will assume that $(x, u) = (0, 0)$ is an equilibrium point, i.e., $f(0, 0) = 0$. By assumption, the above system will be controlled by a state-feedback policy $u = \pi(x)$.

Let us define the value of a policy $\pi(x)$ as:

$$V^{\pi}(x) := \sum_{t=0}^{\infty} \gamma^t r(x_t, \pi(x_t)) \tag{5.2}$$

where $x_0, x_1, \ldots,$ is the trajectory of (5.1) under $u_t = \pi(x_t)$ with initial condition $x_0 = x$. In (5.2), $0 \le \gamma \le 1$ is a discount factor, and $r(\cdot, \cdot)$ is a function $r : \mathbb{R}^{n_x+n_u} \mapsto \mathbb{R}^+$ of state and input known as *immediate cost*.

The control objective will be driving the system towards the origin so, by, assumption, $r(0, 0) = 0$ and $r(x, u) > 0$ for any $(x, u) \neq (0, 0)$. $V^{\pi}(x)$ can be seen as the expected return (cost) if policy $\pi(x)$ were used at all times from initial condition x_0.

Optimal control problems for the above system are usually stated as obtaining an optimal policy $u = \pi^*(x)$ such that, given x, the following cost index is minimised:

$$\pi(x)^* := \arg\min_{\pi \in \Pi} V^{\pi}(x) \tag{5.3}$$

In general, the above problem is solved by searching for the optimal π in a suitable set of functions Π, usually a parameterised function approximator $\Pi := \{\phi(x, \theta) : \theta \in \Theta\}$ so the optimal θ^* is sought in a given parameter set Θ.

5.2.1 Dynamic programming and Q-function fitting

From (5.1) and (5.2), we can write the well-known Bellman equation,

$$V^{\pi}(x) = r(x, \pi(x)) + \gamma V^{\pi}(f(x, \pi(x))). \tag{5.4}$$

In addition to the value of a policy, the so-called *action-value* function $Q^{\pi}(x, u)$, also known as Q-function, is defined in (Watkins 1989) as the return for a given state and action if policy π where applied from the next instant onwards:

$$Q^{\pi}(x, u) := r(x, u) + \gamma V^{\pi}(x_+), \tag{5.5}$$

being $x_+ := f(x, u)$.

For a given policy $\pi(x)$, holds $V^\pi(x) = Q^\pi(x, \pi(x))$; thus, the Q-function fulfils the Bellman's equation:

$$Q^\pi(x,u) - (r(x,u) + \gamma Q^\pi(x_+, \pi(x_+))) = 0, \tag{5.6}$$

and the optimal Q-function fulfils:

$$Q^{\pi^*}(x,u) - \left(r(x,u) + \gamma \min_{u_+} Q^{\pi^*}(x_+, u_+) \right) = 0 \tag{5.7}$$

In the literature, there are quite a few iterative algorithms to estimate the optimal policy, based on dynamic programming (Bellman 1957) and reinforcement learning (Sutton and Barto 2018) such as value iteration, policy iteration, actor-critic setups, etc. Most of them reduce to Riccati equations (or iterations converging to the Riccati solution) for the linear case in model (5.1), see (Lewis, Vrabie, and Vamvoudakis 2012). Let us outline the basic ideas of some of these algorithms which will be later used and compared.

Policy iteration (PI) can be proved that, given a suboptimal policy $\pi(x)$ and its action-value function $Q^\pi(x,u)$, we can define an improved policy given by

$$\hat{\pi}(x) := \arg\min_u Q^\pi(x,u), \tag{5.8}$$

Also we can prove that (Liu et al. 2017):

$$Q^{\hat{\pi}}(x, \hat{\pi}(x)) \leq Q^\pi(x, \pi(x)) \tag{5.9}$$

The next step in the referred algorithms is obtaining an expression $Q^{\hat{\pi}}(x, \hat{\pi}(x))$, denoted as *policy evaluation*, understanding (5.6) as a system of equations that the Q-function must fulfil for all (state, input, successor state) triplets. The interleaved policy evaluation and policy improvement steps are denoted as *policy iteration* (Lewis, Vrabie, and Vamvoudakis 2012) algorithm.

If the set of states and control actions are finite, then a mere lookup-table implementation of $Q^\pi(x,u)$ can be used (Bertsekas and Tsitsiklis 1996; Powell 2011). In continuous-valued state and input spaces, function approximators $Q^\pi(x,u) \equiv Q(x,u,\theta^\pi)$ are needed. Usually, with $Q(x,u,\theta^\pi)$, equation (5.6) cannot be fulfilled and minimising the norm (integral over state+action space) of its (squared) left-hand side is denoted as Bellman residual minimisation (BRM)

(Antos, Szepesvári, and Munos 2008; Sutton et al. 2009). As the parametrised version[2] of policy improvement (5.8) ends up being:

$$\hat{\pi}(x, \theta^\pi) := \arg\min_u Q(x, u, \theta^\pi) \tag{5.10}$$

the so-called *fitted* policy iteration is described in Algorithm 2: from an initial stabilising policy $\hat{\pi}(x, \theta_0)$, its BRM policy-evaluation (5.11) yields θ_1, so $\hat{\pi}(x, \theta_1)$ is obtained, and so on. The norm notation $\|\cdot\|$ indicates the integral of the square over $\mathbb{X} \times \mathbb{U}$.

Algorithm 2 Approximate (fitted) policy/value iteration

```
1: Set an initial parameter $\theta_0$.
2: Using the policy-improvement definition (5.10), solve one of the optimisation
   problems*:
   [policy iteration ]
       $\theta_{i+1} := \arg\min_{\tilde{\theta}} \|Q(x, u, \tilde{\theta}) - r(x, u) - \gamma Q(x_+, \hat{\pi}(x_+, \theta_i), \tilde{\theta})\|$   (5.11)
   The initial parameter in the policy iteration case must yield a stabilising policy $\hat{\pi}(x, \theta_0)$, ensuring
   that $Q(x, u, \theta_1) \geq 0$ and finite for all $(x, u) \in \mathbb{D}$, see (Liu et al. 2017).
   [value iteration]
       $\theta_{i+1} := \arg\min_{\tilde{\theta}} \|Q(x, u, \tilde{\theta}) - r(x, u) - \gamma Q(x_+, \hat{\pi}(x_+, \theta_i), \theta_i)\|$   (5.12)
3: If $\|\theta_{i+1} - \theta_i\| \geq \varepsilon$, set $i = i + 1$ and go to step 2; otherwise STOP.
```

*Note: in exact policy/value iteration algorithms (Sutton and Barto 2018; Busoniu et al. 2010) the minimum achieved norm would be zero. Note also, that the policy has been expressed as the parametrised expression (5.10), even if such parametrisation may be implicit, see footnote 2.

Value iteration (VI) is another alternative to find an approximately optimal Q-function by iterating (5.12). The difference with (5.11) is that both Q-function and policy are using the parameter from previous iteration.

Fitted PI/VI are well-developed techniques; the reader is referred to (Busoniu et al. 2010; Riedmiller 2005; Munos and Szepesvári 2008; Antos, Szepesvári, and Munos 2008; Sutton et al. 2009) for further detail. However, it is well known, too, that these iterative approaches do not converge in a general case (Fairbank

[2]Note that the policy in (5.10) has been intentionally expressed as a parametrised expression depending on θ^π. Actually, in many cases, such parametrisation is implicit, i.e., $\hat{\pi}$ is obtained as a numerical solution of the optimisation problem at the right-hand side, instead of a closed-form expression in x and θ^π. The parametrised notation is of interest for later developments in this work.

and Alonso 2012), and only very restrictive lookup-table-like parametrisations guarantee convergence (Busoniu et al. 2010).

When converged, both algorithms would end up in the iteration fixed point given by the following Bellman residual minimisation:

$$\theta^* := \arg\min_{\theta} \|Q(x,u,\theta) - r(x,u) - \gamma Q(x_+, \hat{\pi}(x_+,\theta), \theta)\|. \tag{5.13}$$

So, conceivably, it might be carried out with any technique, not necessarily fitted PI/VI. This option is not without problems, as the chosen algorithm and initialisation may influence convergence, or getting stuck in spurious local minima. These issues may arise even in the case when the ground-truth optimal value function can be parametrised by $Q(x,u,\theta)$, see Section 5.8.1 for an example of such behaviour with a linear system.

If $Q(x,u,\theta) := \Phi(x,u)\theta$, i.e., the value function approximator is linear in parameters θ, then problems (5.11) and (5.12) can be easily stated as standard linear least squares optimisation. However, this will not hold, in general, with (5.13).

5.2.2 Problem statement

As above discussed, fitted PI/VI or generic minimisation (5.13) need a careful choice of approximator structure (regressors) and initial parameters; otherwise, the learning algorithm may not converge or, even if it does, a suboptimal or even non-stabilising controller can be obtained.

The objective of this chapter is to propose a methodology, based on fuzzy optimal control for TS systems, to address the above issues in applications of dynamic programming or reinforcement learning.

In addition to this, it would be of interest in applications to choose a parametrisation of $Q(x,u,\theta)$ such that an explicit closed-form solution of $\hat{\pi}$ can be derived, to avoid the need of real-time optimisation or large memory requirements and, too, speeding up the computations.

Our proposal will use data-based Takagi-Sugeno identification, fuzzy LMI-based control, and BRM to achieve the mentioned goals.

5.3 Methodology proposal

This chapter will present a series of ingredients to build up a methodology proposal mixing model-based and data-based Q-function fitting approaches, in the line expressed in the above problem statement.

The said ingredients will be as follows:

1. Fuzzy TS modelling and identification, with the objective of building a model so that (sub)optimal fuzzy controllers can be build upon it.
2. The referred fuzzy controllers will be computed with the so-called *guaranteed cost* solutions in literature which, using Linear Matrix Inequalities (LMI) can provide an *upper* bound on the value function Q.
3. The above LMIs can only optimise over Q-function parametrisations in quadratic form. An improved controller will be found by solving one step of the Bellman equation.
4. A more general fuzzy parametrisation of the Q-function will be proposed in Section 5.6, and its optimisation via a generic (monotonic descent) optimisation algorithm will be discussed. In this way, conservatism over LMI results can be reduced.

With all these tools, we can craft a methodology in which an initial model-based approach based on fuzzy-TS models can be a very good starting point for data-based fitted Q-function approaches: with good LMI-based initialisation, the second stage can avoid getting caught at spurious local minima and, also, if monotonic optimisation is used, the LMI result can be improved even in the case PI/VI were not convergent.

Remark: Instead of the actual norm in (5.11)–(5.13), minimisation of these Bellman residuals will be carried out using the finite sum over a set of points in a dataset $\mathcal{D}$ composed of triplets (x, u, x_+), being x_+ the successor state. The dataset can be obtained either by simulation or by experimentation. So we need exciting the system with an input sequence $\{u_0, u_1, \ldots, u_N\}$ and collect $\{x_0, \ldots, x_{N+1}\}$. The dataset will be arranged as :

$$\mathcal{D} := \{(x_0, u_0, x_1), (x_1, u_1, x_2), \ldots, (x_N, u_N, x_{N+1})\} \tag{5.14}$$

Experiment design in such situation may be an important issue, as the relevant region of the state and action space should be well explored. Nevertheless, these issues are out of scope of the present work. Note that the collected data do not need to be generated with any of the policies involved in Algorithm 2, i.e., the proposal is an *off*-policy learner (Sutton and Barto 2018).

5.4 TS models and guaranteed cost controllers

This section will review prior results which will be part of our proposed methodology for engineering applications of fitted Q-function algorithms.

5.4.1 Takagi-Sugeno modelling and identification

If $f(x,u)$ in (5.1) can be expressed as $f(x,u) = h(x) + g(x)u$, and $h(x)$ has continuous first derivatives, then the nonlinear system can be **exactly** modelled using sector nonlinearity based on Takagi-Sugeno fuzzy models (Tanaka and Wang 2001; Sala 2009):

$$x_{t+1} = \sum_{i=1}^{\rho} \mu_i(x_t)(A_i x_t + B_i u_t) = A_{[\mu]} x_t + B_{[\mu]} u_t \tag{5.15}$$

being $\mu(x_t) := \{\mu_1(x_t), \ldots, \mu_\rho(x_t)\}$ a set of $\rho = 2^p$ membership functions with p nonlinearities, where

$$\sum_{i=1}^{\rho} \mu_i(x_t) = 1, \quad 0 \le \mu_i(x_t) \le 1 \tag{5.16}$$

and the notation $A_{[\mu]} := \sum_{i=1}^{\rho} \mu_i(x_t) A_i$, and similarly for $B_{[\mu]}$ has been introduced for compactness.

The above technique is a model-based one, however in the learning approach considered in this manuscript, we pursue a data-based approach. Hence, the TS models will be identified from experiments.

We will assume that a preliminary theoretical model exists for the process under control, so that the membership functions can be extracted from it. If such memberships are known, then the dataset $\mathcal{D}$ can be used to identify the vertices of the TS model, because (5.15) can be written as:

$$x_{t+1} = \begin{pmatrix} A_1 & B_1 & \dots & A_\rho & B_\rho \end{pmatrix} \cdot \begin{pmatrix} \mu_1(x_t)x_t \\ \mu_1(x_t)u_t \\ \vdots \\ \mu_\rho(x_t)x_t \\ \mu_\rho(x_t)u_t \end{pmatrix} \tag{5.17}$$

Now, if we form matrices:

$$\mathcal{X} := \begin{pmatrix} x_1 & x_2 \dots & x_{N+1} \end{pmatrix}_{n_x \times (N+1)} \tag{5.18}$$

$$\Gamma := \begin{pmatrix} \mu_1(x_0)x_0 & \dots & \mu_1(x_N)x_N \\ \mu_1(x_0)u_0 & \dots & \mu_1(x_N)u_N \\ & \vdots & \\ \mu_\rho(x_0)x_0 & \dots & \mu_\rho(x_N)x_N \\ \mu_\rho(x_0)u_0 & \dots & \mu_\rho(x_N)u_N \end{pmatrix}_{(n_x+n_u)\rho \times (N+1)} \tag{5.19}$$

the least-squares estimate of the model matrices is:

$$\begin{pmatrix} \hat{A}_1 & \hat{B}_1 & \dots & \hat{A}_\rho & \hat{B}_\rho \end{pmatrix} = \mathcal{X}\Gamma^\dagger \tag{5.20}$$

where $\Gamma^\dagger$ denotes the Moore-Penrose pseudo-inverse.

In the case where no preliminary model exists and there is no prior insight on which membership functions are the involved, we have a pure black-box identification problem that might need clustering, non-linear model fitting, etc. These issues are, intentionally, out of the scope of this work; the reader is referred to (Hellendoorn and Driankov 1997; Lughofer 2011; Lovera et al. 2011; Škrjanc 2015) and references therein for details.

5.4.2 Guaranteed-cost control

It is well known that a suboptimal control policy for a TS model and an upper bound of its value function can be numerically found in a very efficient way (convex optimisation) using LMIs (Tanaka and Wang 2001). The fact that these LMIs provide only an upper bound of the cost motivates that these algorithms are known in literature as *guaranteed-cost* design methods.

These methods, derived from the LQR techniques, require a quadratic cost index:

$$r(x,u) := x^T H_x x + u^T H_u u \tag{5.21}$$

thus, this immediate cost structure will be assumed in the sequel.

The aim of LMI approaches is to find a positive-definite matrix X that overbounds the optimal *value* function, with guaranteed-cost:

$$\bar{V}^{\pi_{LMI}}(x) := x^T X^{-1} x \geq V^{\pi_{LMI}}(x) \geq V^{\pi^*}(x). \tag{5.22}$$

where the LMI policy $\pi_{LMI}(x_t)$ is fixed to the following structure, known as parallel-distributed compensator (PDC):

$$\pi_{LMI}(x) := -F_{[\mu]} X^{-1} \cdot x := -\sum_{i=1}^{\rho} \mu_i(x) F_i X^{-1} x, \tag{5.23}$$

F_i, X being the decision-variable matrices to be found by solving a set of LMIs:

$$\mathcal{L}(i,j) \geq 0 \quad \forall j = i \tag{5.24}$$

$$\frac{2\mathcal{L}(i,i)}{\rho - 1} + \mathcal{L}(i,j) + \mathcal{L}(j,i) \geq 0 \quad \forall j > i \tag{5.25}$$

with,

$$\mathcal{L}(i,j) = \begin{pmatrix} X & X & F_j & (A_i P - B_i F_j) \\ X & H_x^{-1} & 0 & 0 \\ F_j & 0 & H_u^{-1} & 0 \\ (P A_i^T - F_j^T B_i^T) & 0 & 0 & \gamma^{-1} X \end{pmatrix}$$

where (5.25) is a relaxation to avoid double summation of the Lyapunov equations (Tuan et al. 2001).

The proof of the above assertion appears in the Appendix of this chapter. In fact, it is a straightforward adaptation of the well-known undiscounted $\gamma = 1$ expressions in literature. Furthermore, generalisations to non-quadratic cost bounds $\bar{V}(x) = x^T X_{[\mu]}^{-1} x$ can be also thought of (Márquez et al. 2013; Tanaka, Ohtake, and Wang 2009) but, for brevity, they are left to the reader. The goal of the TS framework will be providing a reasonable initialisation for further data-based learning improvements; actually, the final result will have membership-dependent value functions and non-PDC controllers even if the initial LMIs have membership-independent Lyapunov functions.

5.5 Improved TS guaranteed-cost controllers

Now we will present a so-called one-step controller which will improve the performance bound from LMIs based on the evaluation of the Q-function arising from the LMI solution.

Let us first discuss how the action-value function relates to Lyapunov functions in a linear discrete-time case. Consider $f(x_t, u_t)$ in (5.1) be:

$$x_{t+1} = Ax_t + Bu_t,$$

under a linear state feedback law $u_t = \pi(x_t) = -K^\pi x_t$. It is well known that the value function V^π is quadratic, in the form $V^\pi(x) = x^T P^\pi x$, where P^π is the solution to the Lyapunov equation associated to the feedback gain:

$$P^\pi - H_x - (K^\pi)^T H_u K^\pi - \gamma (A - BK^\pi)^T P^\pi (A - BK^\pi) = 0 \tag{5.26}$$

coming from the Bellman equation (5.4). From V^π, it can be proved that the Q-function in a linear case is, too, quadratic, because replacing the model and value functions in (5.5) we get:

$$\begin{aligned} Q^\pi(x,u) &= \begin{bmatrix} x \\ u \end{bmatrix}^T \left[\begin{pmatrix} H_x & 0 \\ 0 & H_u \end{pmatrix} + \gamma \begin{pmatrix} A^T \\ B^T \end{pmatrix} P^\pi \begin{pmatrix} A & B \end{pmatrix} \right] \begin{bmatrix} x \\ u \end{bmatrix} \\ &= \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} H_x + \gamma A^T P^\pi A & \gamma A^T P^\pi B \\ \gamma B^T P^\pi A & \gamma B^T P^\pi B + H_u \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \\ &:= \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} S_{xx} & S_{ux}^T \\ S_{ux} & S_{uu} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \end{aligned} \tag{5.27}$$

As (5.27) is quadratic in u, straightforward differentiation obtains the explicit solution for the improved policy $\hat{\pi}$ in (5.8):

$$\hat{\pi}(x) = -S_{uu}^{-1} S_{ux} \cdot x := -K^{\hat{\pi}} x. \tag{5.28}$$

Alternatively, a matrix-based proof of the fact that $\hat{\pi}(x)$ improves over $\pi(x)$ appears in the Appendix.

Obviously, if $K^\pi = K^{\hat{\pi}}$ we have converged to the optimal controller. Thus, policy iteration evaluates repeatedly (5.26), (5.27) and (5.28) until convergence.

5.5.1 Fuzzy Q-function

Based on (5.27), we can extend to a TS case the above argumentation. Instead of a Lyapunov equation, in a TS case (5.22) provides an *upper* bound $\bar{V}^{\pi_{LMI}}(x)$ of the cost function of a fuzzy controller $\pi_{LMI}(x)$ given in (5.23) obtained from LMIs. Analogously to the linear case, we can assert an upper bound of the Q-function given by:

$$\bar{Q}^{\pi_{LMI}}(x,u) = \begin{bmatrix} x \\ u \end{bmatrix}^T S_{[\mu^2]} \begin{bmatrix} x \\ u \end{bmatrix} \tag{5.29}$$

being $S_{[\mu^2]}$ a matrix, depending on membership functions, which is partitioned analogously to (5.27) with:

$$S_{[\mu^2],xx} := H_x + \gamma \sum_{i=1}^{\rho} \mu_i(x) A_i^T \cdot X^{-1} \cdot \sum_{i=1}^{\rho} \mu_i(x) A_i \tag{5.30}$$

$$S_{[\mu^2],ux} := \gamma \sum_{i=0}^{\rho} \mu_i(x) B_i^T \cdot X^{-1} \cdot \sum_{i=1}^{\rho} \mu_i(x) A_i \tag{5.31}$$

$$S_{[\mu^2],uu} := \gamma \sum_{i=1}^{\rho} \mu_i(x) B_i^T \cdot X^{-1} \cdot \sum_{i=1}^{\rho} \mu_i(x) B_i + H_u \tag{5.32}$$

Note that elements of matrix $S_{[\mu^2]}$ are quadratic polynomials in the membership functions, which motivates the introduced subscript $_{[\mu^2]}$ notation.

5.5.2 One-step controller

Now, using (5.28) and replacing the $S_{[\mu^2]}$ matrix from (5.29), the resulting controller is a rational function in the membership functions (with degree-2 numerator and denominator terms):

$$\hat{\pi}_1(x) := -(S_{[\mu^2],uu})^{-1} S_{[\mu^2],ux} \cdot x. \tag{5.33}$$

Invertibility of the required matrices is guaranteed if $H_u > 0$.

We can prove that the worst-case performance (upper bound) of $\hat{\pi}_1(x)$ improves over $\pi_{LMI}(x)$. Indeed, an upper bound of the value function of $\hat{\pi}_1(x)$ can be computed as:

$$\bar{V}^{\hat{\pi}_1}(x) := x^T \cdot \left[S_{[\mu^2],xx} - S^T_{[\mu^2],ux} S^{-1}_{[\mu^2],uu} S_{[\mu^2],ux} \right] \cdot x \tag{5.34}$$

The proof of the above is straightforward, by substitution of (5.33) in (5.29). From (5.34), a proof that $\bar{V}^{\hat{\pi}_1}(x) \leq \bar{V}^{\pi_{LMI}}(x)$ can be found in the Appendix. The feedback law (5.33) will be denoted as *one-step controller*. Thus, we have proof that the performance bound $\bar{V}^{\hat{\pi}_1}(x)$ is better than that of the PDC fuzzy controller (5.23), and it is directly obtained from the LMI decision variables and vertex model matrices.

Note that both the classic PDC and the one-step controllers are *shape-independent* (Sala 2009), in the sense that, at design-time, no specific knowledge about the shape of the $\mu_i(x)$ functions is needed. Thus, our first proposal is to actually use (5.33) in applications, instead of (5.23) as its implementation is straightforward and it provides an improved performance bound.

5.6 Q-function parametrisation for Takagi-Sugeno systems

Inspired in the polynomial-in-memberships expression (5.29) for the Q-function of the one-step controller, we will propose a generalisation of it in order to apply fitted Q-iteration and related algorithms to achieve further improvements.

In our proposal, we will set Q to be a polynomial-in-membership expression with arbitrary degree d (homogeneous, with no loss of generality). A monomial of degree d in variables $\mu \in \mathbb{R}^\rho$ will be represented as:

$$\mu^{\mathbf{a}} := \prod_{i=1}^{\rho} \mu_i^{a_i} \tag{5.35}$$

for any multi-index vector $\mathbf{a} := (a_1, \ldots, a_\rho)$ such that $|\mathbf{a}| := \sum_{i=1}^{\rho} a_i = d$ and each a_i is a non-negative integer. Dependence on x of the elements of μ has been omitted for notation compactness.

An homogeneous polynomial of degree d, denoted as $\Xi_{[\mu^d]}$, will be the sum of all these monomials multiplied by a real coefficient, i.e.:

$$\Xi_{[\mu^d]} := \sum_{|\mathbf{a}|=d} \mu^{\mathbf{a}} \xi_{\mathbf{a}} \tag{5.36}$$

where $\xi_{\mathbf{a}}$ conforms a d-dimensional coefficient tensor. Using a similar notation for monomials in x and u, we can express the value function bound (5.29) as:

$$\bar{Q}^{\pi_{LMI}}(x,u) = \Upsilon_{[\mu^2]}(x,u,\bar{q}_{LMI}) := \sum_{\substack{|\mathbf{a}|=2 \\ |\mathbf{b}+\mathbf{c}|=2}} \mu^{\mathbf{a}} x^{\mathbf{b}} u^{\mathbf{c}} q_{\mathbf{abc}} \tag{5.37}$$

for some values of the coefficients $q_{\mathbf{abc}}$ which can be obtained in a straightforward way from (5.30)–(5.32). Notation $\bar{q}_{LMI}$ denotes the vector containing all $q_{\mathbf{abc}}$ in any prescribed order.

Note that (5.37) is linear in the coefficients $q_{\mathbf{abc}}$, thus we can think on $\bar{q}_{LMI}$ as a parameter vector and Υ a linear-in-parameter approximator. Thus, the proposal at this stage is identifying the learnable parameter vector $\theta_0 \equiv \bar{q}_{LMI}$ thus initialising it at the LMI solution. With this initialisation, we may start an iterative Q-function optimisation procedure with the parametrisation $Q(x,u,\theta) \equiv \Upsilon_{[\mu^2]}(x,u,\theta)$, with a total of $\frac{1}{4}\rho(\rho+1)(n_x+n_u)(n_x+n_u+1)$ adjustable parameters.

In fact, the value function approximator can be generalised to a higher-degree homogeneous polynomial in the memberships:

$$\Upsilon_{[\mu^d]}(x,u,\theta) := \sum_{\substack{|\mathbf{a}|=d \\ |\mathbf{b}+\mathbf{c}|=2}} \mu^{\mathbf{a}} x^{\mathbf{b}} u^{\mathbf{c}} q_{\mathbf{abc}} \tag{5.38}$$

and the initial parameter value can be obtained from the coefficients resulting from multiplying the terms in (5.30)–(5.32) by $(\sum_i \mu_i)^{d-2}$ (or $(\sum_i \mu_i)^d$ in the case H_x and H_u).

As an example, in a system with $d = 3$, $\rho = 2$, $\Upsilon_{[\mu^d]}(x,u,\theta)$ would yield the following expression:

$$\begin{aligned}\Upsilon_{[\mu^3]}(x,u,\theta) = \begin{pmatrix} x^T & u^T \end{pmatrix} \big(&\mu_1^3 S_{[\mu^3],1} + \mu_1^2\mu_2 S_{[\mu^3],2} \\ &+\mu_1\mu_2^2 S_{[\mu^3],3} + \mu_2^3 S_{[\mu^3],4} \big) \begin{pmatrix} x \\ u \end{pmatrix}\end{aligned} \tag{5.39}$$

Partial derivatives of the memberships may be added to the parametrisation, too, (Díaz, Sala, and Armesto 2017). Of course, a last generalisation would be adding to $\Upsilon_{[\mu^d]}$ any approximator (a neural network $NN(x, u, \theta_{NN})$, for instance), with its parameters initialised to zero output. In this way, LMI-based initialisation would still be possible. Even if, in theory, such option would offer greater flexibility, this NN approach will not be further pursued because it would, in a generic case, hinder obtaining explicit expressions of the controller (5.10) in the policy-improvement step. The lack of an explicit controller would slow the learning algorithms due to nested numerical optimisation and possibly cause convergence issues. Also, given that the scope of this chapter is exploiting the partial knowledge of the nonlinearities (embedded in memberships) and LMI information for TS systems, we are, intentionally, leaving neural function approximators (or any other unstructured generic option) out of the scope of this work. For neural-network optimal-control applications of the concepts in this chapter, the reader is referred to (Bertsekas and Tsitsiklis 1996; Zhang et al. 2012), for instance. Kernel-based function approximators and Gaussian processes might also be used in reinforcement learning (Xu et al. 2014; De Paula et al. 2015).

5.7 A generic-optimisation approach

An alternative interpretation of the fitted Q-function objective would be, as discussed in Section 5.2, directly solving (5.13).

The above can be conceived as a generic optimisation problem: actually, Algorithm 2 can be interpreted as an iterative way of solving it, but any other iterative numerical optimisation algorithm in literature (gradient, Levenberg-Marquardt, Nelder-Mead, . . .) may be equally used. Thus, under this interpretation, the relevant result of learning is a parameter value achieving good accuracy in the cost index in (5.13), independently of the the actual algorithm used to compute it.

The Q-function parametrisation proposed in the previous section allows to obtain an explicit expression for $\hat{\pi}$, henceforth of the Bellman residual, that avoids nested optimisation and eases the optimisation steps.

For instance, the gradient with respect to θ of the Bellman residual inside the norm in (5.13), denoted as $e(x, u, \theta)$, can be computed as follows:

$$\frac{\partial e}{\partial \theta} = \frac{\partial Q}{\partial \theta}(x, u, \theta) - \gamma \frac{\partial Q}{\partial \theta}(x_+, \hat{\pi}, \theta) - \gamma \underbrace{\frac{\partial Q}{\partial u}(x_+, \hat{\pi}, \theta) \frac{\partial \hat{\pi}}{\partial \theta}}_{\Omega} \tag{5.40}$$

On the other hand, if no explicit expression for $\hat{\pi}$ were available, as $\hat{\pi}$ is optimal, the value function must fulfil $\frac{\partial Q}{\partial u}(x_+, \hat{\pi}, \theta) = 0$. Using a quadratic Taylor-series approximation of $Q(x_+, \hat{\pi} + \delta\hat{\pi}, \theta + \delta\theta)$ and taking derivatives of it with respect to $\delta\hat{\pi}$, and equating to zero (optimality of $\delta\hat{\pi}$), after some manipulations, the derivative of the optimal policy required in (5.40) is:

$$\frac{\partial \hat{\pi}}{\partial \theta} = -\left(\frac{\partial^2 Q}{\partial u^2}\right)^{-1} \frac{\partial^2 Q}{\partial u \partial \theta} \tag{5.41}$$

In this way, we have completed the computation of the gradient of the Bellman Residual.

Note that, in policy-evaluation BRM approaches Antos, Szepesvári, and Munos 2008, i.e., the gradient involved in (5.11), we have $\Omega \equiv 0$; thus, the policy-evaluation gradient does not move the parameters in the actual gradient-descent direction implied in (5.40).

In addition to this, it is well-known that convergence (numerical stability) of generic optimisation tools may be a problem; indeed, this is often the case with PI and VI which require restrictive contraction-related conditions recalled in Section 5.2.1. Contrarily, state-of-the-art optimisation techniques incorporate variable step-sizes, intermediate line search stages, etc. greatly improving its numerical stability (Chong and Zak 2013). In the examples in Section 5.8, Matlab has been used to carry out the optimisation, using the default quasi-Newton plus line-search algorithm of `fminunc`, as well as the simplex Nelder-Mead in `fminsearch`.

Nevertheless, when generic nonlinear optimisation is considered, it is well known that many of the algorithms in literature can easily get trapped in spurious solutions (local minima). A good initialisation is essential for succeeding in finding a good optimiser θ^* in this case; this is why our methodology proposes LMI-based initialisation[3].

[3] Obviously, global optimisers using, say, evolving algorithms or other population-based ideas (Pelikan, Goldberg, and Lobo 2002; Lughofer 2011) might avoid the need of a good initialisation. However, their computational efficiency with large parameter sets is usually worse than the Quasi-Newton or Nelder-Mead options, so these options will not be further discussed.

5.7.1 Summary

Given the above issues, our proposal outlined in Section 5.7.1 and developed in Section 5.4 onwards can be summarised as:

1. From preliminary theoretical insight, identify the most relevant nonlinearities and build the associated membership functions.
2. Identify a TS model, as proposed in Section 5.4.1.
3. Obtain a guaranteed-cost LMI controller, Section 5.4.2.
4. Build the fuzzy Q-function (5.29) based on the LMI solution.
5. Initialise a monotonic optimisation algorithm with the above Q-function decision variables, and perform the optimisation of (5.13).

The reason of proposing the last monotonic optimisation step is to ensure that the result of our proposal will, at least, improve over the conservative LMI solution in the cost index of (5.13) without convergence problems. Note that, however, PI/VI (Algorithm 2) using the initial LMI solution might also be a sensible option. Furthermore, greedy search or population-based directed random search (genetic, swarm, etc.) techniques may be used instead of the suggested monotonic optimisation. However, these algorithms are much slower than derivative-based options, so they are not proposed as a first-choice option.

5.8 Case study: inverted pendulum

This section will present a case study using an inverted pendulum where the previous proposals will be illustrated. First, simulation studies with a theoretical model will be addressed. Later on, experimental identification and learning on an actual pendulum will be tested.

The objective of these examples will be showing the advantages of our proposed methodology (LMI initialisation + monotonic optimisation) with respect to the other alternatives reviewed in previous sections. In particular, a simulation of an ideal linearised setup will show that VI/PI (Algorithm 2) may fail even if the function approximator includes the true value function (known to be quadratic in this case). In fact, even the BRM may fail if wrongly initialised, as shown there. Examples show, too, that with LMI initialisation, in case PI/VI are convergent they converge to basically the same solution as the monotonic BRM optimisation.

Table 5.1: Inverted pendulum model parameters

Model parameter	Symbol	Value	Units
Pendulum mass	M	2.40	kg
Center of gravity length	L	0.30	m
Friction Coefficient	β	0.35	$\mathrm{kg} \cdot \mathrm{m}^2 \cdot \mathrm{s}^{-1}$
Pendulum inertia	I	0.272	$\mathrm{kg} \cdot \mathrm{m}^2$
Gravity	g	9.81	$\mathrm{m} \cdot \mathrm{s}^{-2}$

5.8.1 Simulation examples

Consider an inverted pendulum model with 1 degree of freedom discretised with forward Euler approximation:

$$\alpha_{t+1} = \alpha_t + \delta\dot{\alpha}_t \tag{5.42}$$

$$\dot{\alpha}_{t+1} = \dot{\alpha}_t + \delta\frac{u_t - \beta\dot{\alpha}_t + MgL\sin(\alpha_t)}{I} \tag{5.43}$$

with $x_t = [\alpha_t \; \dot{\alpha}_t]^T$ where α_t is the beam angle and $\dot{\alpha}_t$ its velocity, M the mass, L length of the bar (and the position of the centre of gravity), β the friction coefficient, I the inertia and $\delta = 0.01$ s the sampling time; see Table 5.1 for numerical values of physical parameters.

Linearised case

Now, the linearisation of the inverted pendulum (5.42)-(5.43) system is considered. Specifically, we will linearise the equations around the vertically upward equilibrium position, $\alpha_t = 0$. So, the resulting linearised system is

$$x_{t+1} = Ax_t + Bu_t \tag{5.44}$$

with:

$$A = \begin{bmatrix} 1 & 0.0100 \\ 0.2598 & 0.9871 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0.0368 \end{bmatrix} \tag{5.45}$$

The control problem will be stated using the quadratic cost (5.21) with:

$$H_x = \mathrm{diag}([100,\ 10]),\ H_u = 1$$

and the discount factor will be set to $\gamma = 1$, so it is a standard LQR setup.

The optimal solution for the above-stated quadratic cost problem is well-known, as well as the fact that the true Q-function is quadratic in states and control inputs. The actual LQR solution is:

$$Q^*(x,u) = 6350.8x_1^2 + 1272.8x_1x_2 + 140.16x_2^2 + 42.88x_1u + 9.25x_2u + 1.165u^2 \tag{5.46}$$

Thus, if we define the following linear-in-parameters approximator:

$$Q(x,u,\theta) := [\varphi(x) \quad \varphi(x)u \quad \varphi(x)u^2] \cdot \begin{bmatrix}\theta_1\\ \theta_2\\ \theta_3\end{bmatrix} \tag{5.47}$$

for some unknown parameter vector $\theta := [\theta_1^T\ \theta_2^T\ \theta_3^T]$ and regressor vector $\varphi(x)$

$$\varphi(x) := [x_1^2 \quad x_1x_2 \quad x_2^2 \quad x_1 \quad x_2 \quad 1] \tag{5.48}$$

then, this choice of $Q(x,u,\theta)$ with 18 adjustable parameters can exactly fit the optimal LQR Q-function if the 6 parameters multiplying x_1^2, x_1x_2, x_2^2, x_1u, x_2u, u^2 are non-zero and the remaining 12 are zero.

The objective of this subsection is showing that even in this idealistic situation (linear process, true model of Q in the model set, i.e., the function approximator for Q includes the LQR solution), the PI/VI (Algorithm 2) or local optimisation approaches (Section 5.7) might fail.

In order to show these drawbacks, we have initialised the parameter vector θ to the following values:

$$\begin{aligned}
\theta_1 &= \left[6342.3\ 163.7\ 1329.4\ -0.216\ 0\ 0\right]^T\\
\theta_2 &= \left[0\ 0\ 0\ 42.3\ 9.27\ 0\right]^T\\
\theta_3 &= \left[0.051\ 0.002\ -0.061\ 0\ 0\ 0.359\right]^T
\end{aligned}$$

With these parameters, the resulting controller from (5.10) was slightly nonlinear but, nevertheless, it was shown to be stabilising when starting in several simulations with random initial conditions with $x_1 \in [-\pi, \pi]$, $x_2 \in [-15, 15]$.

A dataset (5.14) was generated from an equally-spaced grid in the above position/speed region, as well as a similar grid in the interval for control action $u \in [-120, 120]$.

Applying Algorithm 2 with the above dataset, approximator and initial (stabilising) parameters, it can be shown that neither VI nor PI versions of the algorithm converge to a stabilising controller (details omitted for brevity).

Additionally, using `fminsearch` or `fminunc` functions from Matlab Optimisation Toolbox Version 7.6 (R2017a) to solve problem (5.13), the objective function gets stuck on a local minimum and the resulting controller is not stabilising, even if the original one was.

In summary, PI and VI algorithms cannot provide a valid controller despite of having a linear system with an initial stabilising controller and a Q-function approximator able to represent the true LQR solution. Of course, such divergence can occur in many other situations (Fairbank and Alonso 2012). Also, generic optimisation may give useless results if not properly initialised.

Proposed fitted Q-function methodology

In the model (5.43), the nonlinearity is given by the sinusoidal function and the membership functions for the TS model are:

$$\mu_1(x) := \begin{cases} \frac{\sin(\alpha)/\alpha - \sin(\alpha_{max})/\alpha_{max}}{1-\sin(\alpha_{max})/\alpha_{max}} & \text{if } \alpha \neq 0 \\ 1 & \text{if } \alpha = 0 \end{cases}$$
$$\mu_2(x) := 1 - \mu_1(x)$$

with $\alpha_t \in [-\alpha_{max}, \alpha_{\max}]$ rad. and $\alpha_{max} = \pi$. Model vertex matrices are:

$$A_1 = \begin{bmatrix} 1 & \delta \\ \frac{\delta M g L}{I} & 1 - \frac{\beta\delta}{I} \end{bmatrix}, \qquad B_1 = \begin{bmatrix} 0 \\ \frac{\delta}{I} \end{bmatrix},$$
$$A_2 = \begin{bmatrix} 1 & \delta \\ 0 & 1 - \frac{\beta\delta}{I} \end{bmatrix}, \qquad B_2 = \begin{bmatrix} 0 \\ \frac{\delta}{I} \end{bmatrix}.$$

The LMI controller for the same problem as in the linearised case provides the following bound for the *value* function:

$$\bar{V}^{\pi_{LMI}}(x) = x^T \begin{bmatrix} 7598.37 & 588.28 \\ 588.28 & 130.05 \end{bmatrix} x \tag{5.49}$$

and a policy[4] from (5.23):

$$\hat{\pi}_{LMI}(x) = -\left[19.19972 \;\; 4.25971\right] x. \tag{5.50}$$

From the LMI output in (5.50), the resulting matrices for the one-step controller (5.33) are:

$$\begin{aligned} S_{[\mu^2],xx}(x) &= \begin{bmatrix} 7698.37 + 305.65\mu_1(x) + 8.777\mu_1^2(x) & 656.69 + 34.88\mu_1(x) \\ 656.69 + 34.88\mu_1(x) & 149.1 \end{bmatrix} \\ S_{[\mu^2],ux}(x) &= \begin{bmatrix} 21.63 + 1.24\mu_1(x) & 4.93 \end{bmatrix} \\ S_{[\mu^2],uu}(x) &= \begin{bmatrix} 1.17 \end{bmatrix} \end{aligned}$$

where μ_2 has been replaced by $1 - \mu_1$ so only μ_1 appears.

This provides an improved one-step controller yielding a fuzzy policy:

$$\hat{\pi}_1(x) = -\left[18.39977 + 1.05675\mu_1(x) \quad 4.19946\right] x$$

Figure 5.1 represents the ratio between the *actual* performance of the one-step and LMI controllers, evaluated by a simulation from a grid of different initial conditions. Note that the figure plots actual 300-step "measured" performance (5.2), and not the performance bounds $\bar{V}^{\pi_{LMI}}(x)$ or $\bar{V}^{\hat{\pi}_1}(x)$ in (5.34). Thus, the proposed one-step controller slightly improves performance up to 2% in some states over the LMI solution (5.50).

One-step controller will be further improved via fitted Q-function algorithms using a dataset $\mathcal{D}$, with $N = 1000$ samples equally spaced on the interval $\alpha_t \in [-\pi, \pi]$ rad, $\dot{\alpha}_t \in [-15, 15]$ rad/s and $u_t \in [-120, 120]$ N·m using the nonlinear model in (5.42)-(5.43).The fuzzy Q-function approximator (5.38) was set to $d = 2$, and initialised with (5.30)–(5.32).

[4]In this case, the particular model and cost index parametrisation results in a linear state-feedback controller, because the LMI solver outputs $F_1 = F_2$.

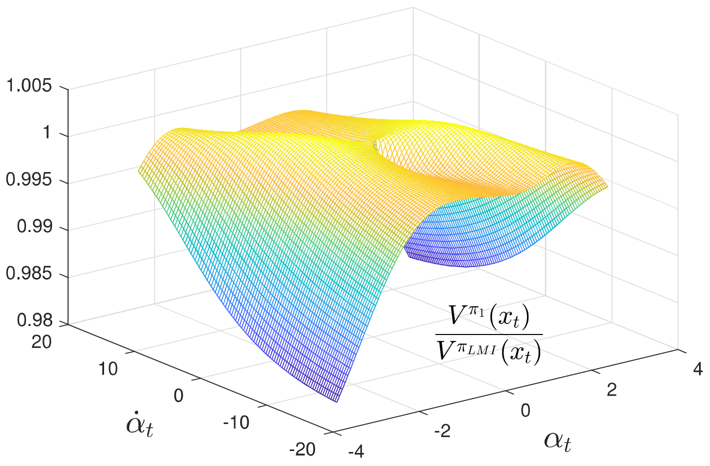


Figure 5.1: Ratio of measured performance of one-step controller with respect to the LMI controller.

Then, using `fminunc`[5]optimisation of (5.13) the Q-function approximation converges to:

$$Q(x,u) \approx \begin{pmatrix} x \\ u \end{pmatrix}^T (S_{[\mu^2]}) \begin{pmatrix} x \\ u \end{pmatrix} \tag{5.51}$$

with:

$$S_{[\mu^2],xx} = \begin{bmatrix} 4736.7 - 6887.6\mu_1(x) + & 319.4 - 586.6\mu_1(x) + \\ 2089.5\mu_1(x)^2 & 208.2\mu_1(x)^2 \\ 319.4 - 586.6\mu_1(x) + & 104.8 - 121.6\mu_1(x) + \\ 208.2\mu_1(x)^2 & 81.6\mu_1(x)^2 \end{bmatrix}$$

$$S_{[\mu^2],ux} = \begin{bmatrix} 10.6 - 17.2\mu_1(x) + 5.2\mu_1^2(x) & 3.5 - 3.6\mu_1(x) + 2.3\mu_1(x)^2 \end{bmatrix}$$

$$S_{[\mu^2],uu} = \begin{bmatrix} 1.1 - 0.4\mu_1(x) + 0.3\mu_1(x)^2 \end{bmatrix}$$

and, as a result, the controller (5.8) yields the following rational-in-memberships controller:

$$\hat{\pi}_{\mu^2}(x_t) = -\begin{bmatrix} \frac{5.2657\mu_1^2(x_t)+6.6635\mu_1(x_t)+10.643}{0.09026\mu_1^2(x_t)-0.048143\mu_1(x_t)+1.12789} \\ \frac{2.6398\mu_1^2(x_t)-1.3338\mu_1(x_t)+3.46478}{0.09026\mu_1^2(x_t)-0.048143\mu_1(x_t)+1.12789} \end{bmatrix}^T \cdot x_t.$$

[5] According to Matlab's documentation, `fminunc` implements a quasi-Newton plus line-search monotonic descent algorithm.

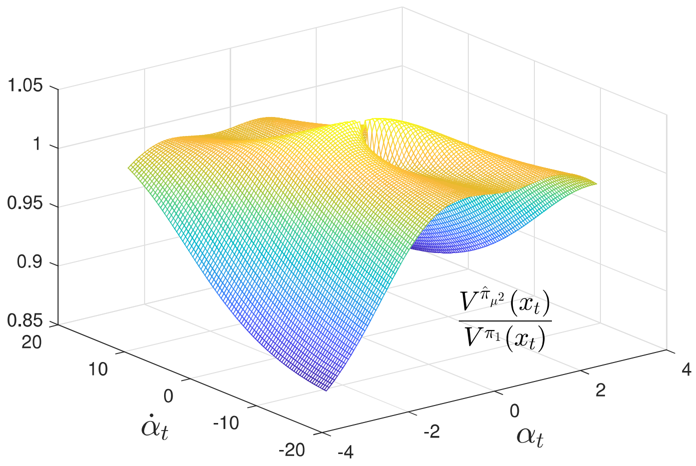


Figure 5.2: *Value* function ratio of shape-dependent fitted Q-function controller with $d = 2$ (with respect to the one-step controller).

If we compare this learnt controller against the initial one, we can see that there is a performance improvement of around 15% in some states, as shown in Figure 5.2.

If the procedure is done with a Q-function approximator of degree $d = 3$, the result is practically identical in this example as the one with $d = 2$, so a degree increase does not seem worthwhile for this particular application.

As an alternative, using policy iteration Algorithm 2 from the same initial parameters (i.e., obtained with LMI and one-step controller) yielded basically the same solution, as discussed in our previous work (Díaz, Armesto, and Sala 2016): LMI initialisation is a convenient option *per se*, whatever the later tuning steps are. However, our proposal has a *guaranteed* Bellman residual descent from an already sensible LMI solution, whereas PI does *not*.

Summarising, the proposed methodology for Q-function parametrisation and initialisation is able to achieve improved controllers with respect to the LMI solution in this process.

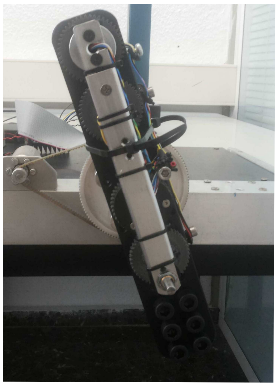


Figure 5.3: Inverted Pendulum for experiments

5.8.2 Experimental evaluation

Now, we will apply our proposed methodology in Section 5.7.1, from scratch, to a dataset obtained from the 1 DoF pendulum in Figure 5.3, including identification steps.

The hardware is an inverted pendulum mechanism consisting of an electric motor, which actuates a link with a bar. A power drive is able to operate the motor in the range ±12 V. A position sensor based on an Hall effect encoder is used for data acquisition. Control sampling period was set to $\delta = 10$ ms and the position sensor was operated at 1 ms sampling period. The controller was fed with filtered position/speed data (every 10 samples) coming from a Kalman filter for a double-integrator model. Dataset acquisition is performed in real-time using a NI myRIO-1900 embedded device by National Instruments and LabVIEW 2015. The learning algorithm phase and data processing were implemented in Matlab R2017a.

Also, as there was a significant actuator dead-zone due to Coulomb friction, such dead-zone was compensated as follows

$$
\begin{aligned}
u_a &= u + u_{comp} \\
u_{comp} &= \begin{cases} 0, & \text{if } |u| < k_1, \\ k_2 \cdot sign(u), & \text{otherwise} \end{cases}
\end{aligned}
$$

where k_1 and k_2 are 1% and 15% of input range respectively, and u is the output of the controllers to be designed whereas u_a is the actual voltage applied to the

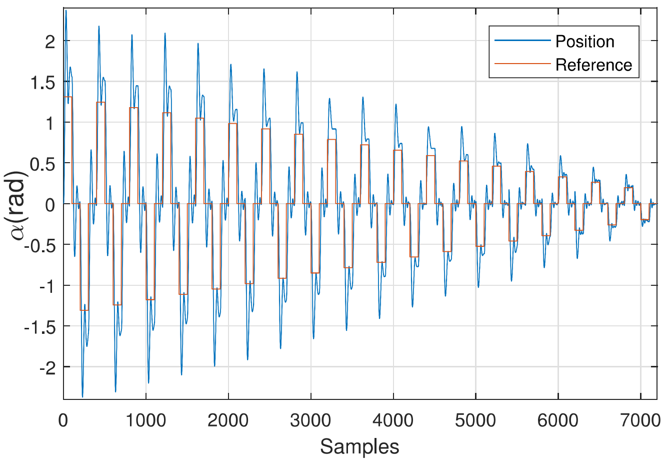


Figure 5.4: Position dataset for identification and learning.

motor. Note that, as intuitively expected, there will be some chattering in control input when the state is close to the equilibrium. The chattering appears due to the combined action of disturbances on an unstable system and the sign function in the above compensation.

As the pendulum is unstable, we need an initial stabilising controller to be able to collect any significant amount of data. Thus, the PD controller $u = -60(\alpha - \alpha_{ref}) - 1\dot{\alpha}$ was implemented to gather the dataset.

A dataset of 7200 data points (72 seconds of experiment) has been collected using the above PD controller and a varying set-point signal. In addition to the control action of this PD, we added a Gaussian noise with standard deviation 5% of the actuator range to generate some extra excitation, as typically required in identification and learning techniques. The sample time T_s is 10 ms. The set-point and actual pendulum angle is shown in Figure 5.4.

Thus, according to the proposed methodology, we will obtain first an identified TS model, second and an initial model-based stabilising control law using LMIs, for a later third stage of data based tuning.

Identification and model-based LMI controller Applying the identification method proposed in Section 5.4.1 the estimated matrices are:

$$A_1 = \begin{bmatrix} 1.0019 & 0.0097 \\ 0.3312 & 0.9718 \end{bmatrix} \qquad B_1 = \begin{bmatrix} 0.0001 \\ 0.0220 \end{bmatrix}$$
$$A_2 = \begin{bmatrix} 1.0002 & 0.0097 \\ -0.0156 & 0.9529 \end{bmatrix} \qquad B_2 = \begin{bmatrix} 0.0001 \\ 0.0131 \end{bmatrix}$$

The quadratic cost index was set with $H_x = \text{diag}([100,\ 2.5])$, $H_u = 0.15$ and $\gamma = 1$ for the discount factor. The LMIs provide the following bound for the *value* function and its associated state feedback policy:

$$\begin{aligned} \bar{V}^{\pi_{LMI}}(x) &\le x^T \begin{bmatrix} 4050.4 & 360.1 \\ 360.1 & 60.4 \end{bmatrix} x \\ \pi_{LMI}(x) &= -\left[41.1\ 5.8\right] x \end{aligned} \tag{5.52}$$

These computations with the identified TS model will be only used to initialise the function approximation for data-based fitted Q-function, to be discussed next.

Data-based fitted Q-function tuning After these model-based initial stages, finally, a data-based optimisation with `fminsearch` (which implements variations of simplex Nelder-Mead monotonic descent algoritm Lagarias et al. 1998) is carried out (obviously `fminunc` may also be used, as done in the simulation examples, but we chose `fminsearch` to illustrate another option).

The said optimisation minimises the sum of the squares of the Bellman residual, i.e., (5.13), over the same experimental dataset used for identification (Fig. 5.4) arranged as in (5.14), so the second phase of the Q-function fitting is model-free. The linear-in-parameter fuzzy Q-function approximator was (5.37), initialised with the LMI solution.

The resulting controller using the Fuzzy fitted Q-function algorithm converges to the following controller (rational in the membership functions):

$$\hat{\pi}_{[\mu^2]}(x) = -\begin{bmatrix} \frac{-9.17\mu_1^2(x)+6.19\mu_1(x)+7.91}{-0.36\mu_1^2(x)+0.33\mu_1(x)+0.18} \\ \frac{-1.12\mu_1^2(x)+0.94\mu_1(x)+0.98}{-0.36\mu_1^2(x)+0.33\mu_1(x)+0.18} \end{bmatrix}^T \cdot x \tag{5.53}$$

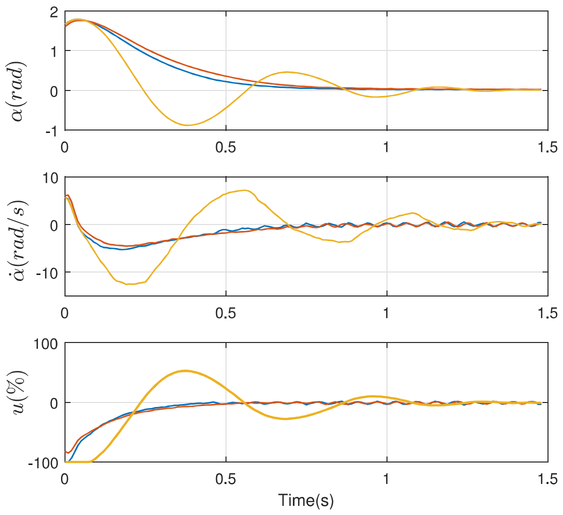


Figure 5.5: PD (yellow), LMI (blue) and learning (red) controller trajectories from $x_0 = [1.6, 5.2]^T$. Recall that position and speed figures have been obtained from a 10x oversampling Kalman filter which averages measurement noise.

From the initial state $x_0 = [1.6, 5.2]^T$, Figure 5.5 show the position, speed trajectories and control input. For the sake of comparison, the trajectories for the initial PD, the LMI controller and the final learnt one (5.53) are all plotted.

In order to show the actual experimental cost index performance, Figure 5.6 depicts the accumulated quadratic cost for the three compared controllers.

5.9 Conclusions

In this chapter we have presented a fuzzy fitted Q-function methodology to obtain approximately optimal controllers based on Takagi-Sugeno systems.

The method first proposes to identify a fuzzy model based on collected data, incorporating partial model information via the memberships arising from sector-nonlinearity TS theoretical models. Second, a guaranteed cost controller based on LMIs can be designed for the identified system. Last, based on this controller, we can use it to initialise a fuzzy function approximator of the Q-function and apply Q-function fitting algorithms from the same dataset used in the identification phase. The Q-function fitting step can be carried out with standard pol-

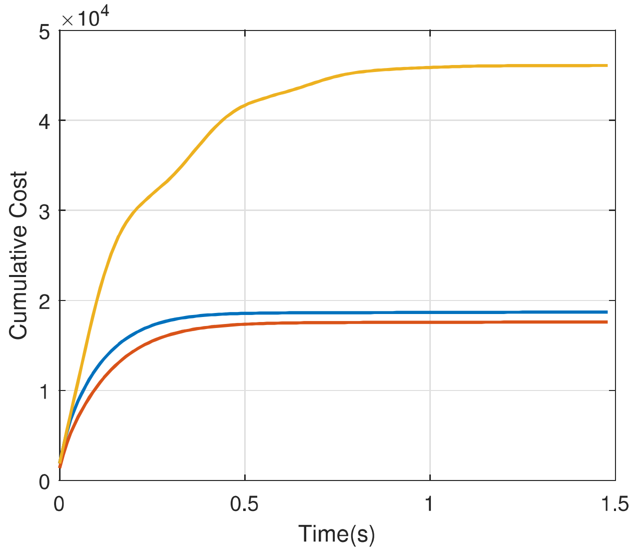


Figure 5.6: Cost performance of PD (yellow), LMI (blue) and learning (red) controllers.

icy/value iteration or, alternatively, if there are convergence problems, via generic (monotonic) optimisation algorithms. Examples illustrate that the methodology proposal (LMI initialisation plus monotonic optimisation) is advantageous with respect to standard PI/VI (which may have convergence issues) or BRM (which may converge to a local minima if not properly initialised).

This methodology combines model-based optimal control (via identification) with data-based learning. We have shown that the proposed initialisation (guaranteed-cost controller for the identified TS system) can be considered a good starting condition for most fitted Q-function algorithms. The chapter presents results both in simulation and with real experimental data from an inverted pendulum.

5.10 Appendix

5.10.1 Nonlinear Guaranteed-costs solutions

The LMIs (5.24) and (5.25) come from replacing the Bellman equation (5.4) by an inequality

$$\bar{V}^{\pi}(x) \geq r(x, \pi(x)) + \gamma \bar{V}^{\pi}(x_{+}) \tag{5.54}$$

so that the value functions $\bar{V}$ that fulfil it are only upper bounds of the optimal cost, i.e., $\bar{V}(x) \geq V^{\pi}(x)$.

With the proposed expression for $\bar{V}$ and π_{LMI}, the inequality gets:

$$\begin{aligned} x^T X^{-1} x - x^T H_x x - x^T X^{-1} F_{[\mu]}^T H_u F_{[\mu]} X^{-1} x \\ - \gamma x^T (A_{[\mu]} + B_{[\mu]} F_{[\mu]} X^{-1})^T X^{-1} (A_{[\mu]} + B_{[\mu]} F_{[\mu]} X^{-1}) x \geq 0 \end{aligned} \tag{5.55}$$

The change of variable $X^{-1}x = \psi$ transforms the above to:

$$\begin{aligned} \psi^T X \psi - \psi^T X H_x X \psi - \psi^T F_{[\mu]}^T H_u F_{[\mu]} \psi \\ - \gamma \psi^T (A_{[\mu]} X + B_{[\mu]} F_{[\mu]})^T X^{-1} (A_{[\mu]} X + B_{[\mu]} F_{[\mu]}) \psi \geq 0 \end{aligned} \tag{5.56}$$

So, application of Schur complement and double-sum relaxation yields the stated LMIs conditions.

5.10.2 Performance bound of the one-step controller

Consider a matrix

$$S := \begin{bmatrix} S_{xx} & S_{ux}^T \\ S_{ux} & S_{uu} \end{bmatrix} \tag{5.57}$$

and consider now the expression:

$$\Xi(S, \hat{K}) := \begin{pmatrix} I & \hat{K}^T \end{pmatrix} \begin{bmatrix} S_{xx} & S_{ux}^T \\ S_{ux} & S_{uu} \end{bmatrix} \begin{pmatrix} I \\ \hat{K} \end{pmatrix} \tag{5.58}$$

If we set $\hat{K} = -S_{uu}^{-1} S_{ux} + \Delta$, elementary algebraic manipulations yield:

$$\Xi(S, -S_{uu}^{-1} S_{ux} + \Delta) = S_{xx} - S_{ux}^T S_{uu}^{-1} S_{xu} + \Delta^T S_{uu} \Delta \tag{5.59}$$

thus, if S_{uu} is positive definite, we can ensure that

$$\begin{aligned}\Xi(S,\hat{K}) = \begin{pmatrix} I & \hat{K}^T \end{pmatrix} \begin{bmatrix} S_{xx} & S_{ux}^T \\ S_{ux} & S_{uu} \end{bmatrix} \begin{pmatrix} I \\ \hat{K} \end{pmatrix} \\ \geq \begin{pmatrix} I & -S_{ux}^T S_{uu}^{-1} \end{pmatrix} \begin{bmatrix} S_{xx} & S_{ux}^T \\ S_{ux} & S_{uu} \end{bmatrix} \begin{pmatrix} I \\ -S_{uu}^{-1} S_{ux} \end{pmatrix} \\ = S_{xx} - S_{ux}^T S_{uu}^{-1} S_{xu} = \Xi(S, -S_{uu}^{-1} S_{ux}) \quad (5.60)\end{aligned}$$

for any matrix $\hat{K}$ of compatible size.

Linear case In a linear case with controller $\pi(x) = -K^\pi x$, the Lyapunov equation (5.26) can be equivalently understood as $\Xi(S, K^\pi) = P^\pi$, thus from (5.60) we can assert $x^T P x \geq x^T (S_{xx} - S_{ux}^T S_{uu}^{-1} S_{xu}) x = Q^\pi(x, \hat{\pi}(x))$, and from standard dynamic-programming (5.9) argumentations, we can quantify a bound for the policy improvement $x^T P x \geq x^T (S_{uu} - S_{ux}^T S_{uu}^{-1} S_{xu}) x \geq Q^{\hat{\pi}}(x, \hat{\pi}(x))$.

TS case The guaranteed-cost LMIs (5.24) and (5.25) imply (5.56), and the latter equation implies $X^{-1} \geq \Xi(X^{-1}, F_{[\mu]})$, understanding the generic S in (5.57) to be $S_{[\mu^2]}$ in (5.29). Thus, we have proven that

$$\begin{aligned}\bar{V}^{\pi_{LMI}}(x) = x^T X^{-1} x \geq x^T \Xi(X^{-1}, F_{[\mu]}) x \\ \geq x^T \Xi(X^{-1}, -(S_{[\mu^2],uu})^{-1} S_{[\mu^2],ux}) x = \bar{V}^{\hat{\pi}_1}(x) \geq V^{\hat{\pi}_1}(x) \quad (5.61)\end{aligned}$$

thus the upper bound $\bar{V}^{\hat{\pi}_1}(x)$ is better than the LMI-proven bound $\bar{V}^{\pi_{LMI}}(x)$.

Chapter 6

A Linear Programming Methodology for Approximate Dynamic Programming

The linear programming approach to solve the Bellman equation in dynamic programming is a well-known option in finite state and input spaces. In such a case, an exact solution of an optimal control problem can be obtained; if a function approximator was used, a lower bound and an upper bound of the optimal value function would be the result. This paper presents a methodology to make approximate dynamic programming via LP work in practical control applications with continuous state and input spaces: there are some guidelines in data and regressor choice needed to obtain meaningful and well-conditioned value function estimates. The introduction of terminal ingredients (switching to an LQR setup when close to the origin; penalisation when hitting operational constraints) also improves the final controller performance. Additionally, lower and upper bounds of the value function are computed so that the gap between them provides an idea of the suitability of the regressors for a particular application. An experimental inverted-pendulum application will be used to illustrate the proposal and carry out suitable comparative analysis with alternative options in literature.

The content of this chapter appears in the journals articles:

- Díaz, Henry, Leopoldo Armesto and Antonio Sala (2019) "Metodología de programación dinámica aproximada para control óptimo basada en datos". In: Revista Iberoamericana de Automática e Informática industrial, [S.l.], v. 16, n. 3, p. 273-283, jun. 2019. ISSN: 1697-7920. DOI:10.4995/riai.2019.10379.
- H. Díaz, A. Sala, and L. Armesto (2019). "A Linear Programming Methodology for Approximate Dynamic Programming". Submitted.

6.1 Introduction

Optimal control is a relevant, widely-used strategy in many engineering applications (Bertsekas 2017). In process control, these problems are stated as minimising an infinite-time cost index or, in some cases, a finite-time approximation of it; these finite-time approximations are usually solved via the so-called model predictive control (MPC) methods (Camacho and Bordons 2013). The Linear Quadratic Regulator (LQR) is well known, and so it is the linear MPC. The nonlinear case is more demanding (Allgower and Zheng 2012); iterative LQR setups (Armesto et al. 2015) may be used to obtain solutions closed to the optimal one by successive linearisations, or guaranteed-cost LMIs can obtain upper cost bounds if a Takagi-Sugeno model of the nonlinear system is built (Tanaka, Ohtake, and Wang 2009; Ariño, Querol, and Sala 2017).

At the core of most of these techniques is Dynamic programming (DP) (Bertsekas 2017; Blackwell 1965; Lewis and Liu 2013). Indeed, DP obtains a solution of optimal control problems based on the Bellman equation, or the Hamilton-Jacobi-Bellman one in a continuous-time framework. However, in infinite state or action spaces, closed-form solutions to these optimal control problems are available only in a handful of cases, basically in linear processes via the celebrated LQR (Lewis and Liu 2013).

DP solutions and applications can either be off-line model-based ones, or on-line ones (with or without model) giving rise to *adaptive* DP or reinforcement learning (RL) paradigms (Liu et al. 2017; Radac, Precup, and Roman 2017; Liu and Yang 2018). DP estimates a value function $V(x)$ associated to the optimal control problem in consideration. In order to estimate it, policy iteration (PI) and value iteration (VI) are popular paradigms used to evaluate the expected value of a given policy and obtain an improved policy (Lewis and Liu 2013; Lewis, Vrabie, and Vamvoudakis 2012; Zhang et al. 2016). PI and VI converge to the optimal value and policy under mild conditions ensuring a contraction mapping (Busoniu et al. 2010). However, such mild conditions are actually so only in systems with a finite number of states and actions. DP techniques applied to continuous states are commonly known *approximate* dynamic programming (ADP). All these techniques, DP, RL, adaptive DP and ADP have been successfully used in a wide range of areas including control, tracking, robotics, power systems (Lewis and Liu 2013; Han, Feng, and Cui 2017; Kiumarsi et al. 2018).

This chapter will focus only on the off-line ADP (using a batch of data assumed to be available *a priori*), so the reader is referred to the previous cited works and references therein for on-line/adaptive/reinforcement learning options. In a generic ADP case, function approximation needs to be used (Busoniu et al. 2010; Powell 2011); however, with arbitrary parametrisations $V(x, \theta)$, PI and VI might not converge. This is also the case for Actor-Critic using policy iteration where the "Critic" evaluates a the value function for a given policy and the "Actor" updates the policy with some policy-improvement method (Lewis, Vrabie, and Vamvoudakis 2012). A way to avoid PI/VI divergence is trying to minimise the so-called mean-square Bellman error (Sutton et al. 2009) via gradient methods (Munos, Baird, and Moore 1999; Baird and Moore 1999) or other monotonic optimisation methods (Lagarias et al. 1998). However, such methods open up the possibility of getting caught on local minima if they are not properly initialised.

As an alternative, (De Farias and Van Roy 2003) proposed a linear-programming (LP) approach, changing equalities to inequalities in the Bellman equation, to generate a lower bound of the value function using a function approximator. The cited work referred to finite, discrete, states and action spaces; exact LP solutions to some discrete dynamic programming problems were known much earlier (Manne 1960). However, it laid the ground to propose randomised approaches to "large" (finite) state and action spaces with some probabilistic guarantees (De Farias and Van Roy 2004; Falsone and Prandini 2015), obtaining an approximate lower bound of an optimal control problem whereas the exact solution may be potentially intractable with LP. Bellman inequalities are also used in other settings involving semidefinite-programming constraints (Ariño, Pérez, and Sala 2010; Stellato, Geyer, and Goulart 2017), not considered in the scope of this work.

The objective of this chapter is proposing a methodology, derived from the seminal ideas in (De Farias and Van Roy 2003) to approximately solve deterministic optimal control problems[1] with approximate dynamic programming and LP (coined as ADP-LP) in a grid of samples of state and action data points (as a continuous state and input space is, in general, intractable). However, in a continuous-state problem, the successor state will not belong to the grid, and the data and regres-

[1]In (De Farias and Van Roy 2003), a stochastic scenario in finite state and action spaces was discussed; however, our interest will be restricted to deterministic optimal control problems so stochastic options will be left out of the scope of this work.

sor choice for the approximator $V(x, \theta)$ may end up yielding unbounded solutions or ill-conditioned ones.

Thus, the contributions of this chapter will be:

1. detecting improper regressor arrangements causing unbounded or ill-conditioned solutions, and suggesting solutions to this issue via addition, deletion or shifting of regressors;
2. introducing outer terminal ingredients to penalise leaving the region with data availability, and inner terminal ones to avoid anomalous behaviour (such as offset) close to the origin due to the approximation error and control action gridding;
3. computing an upper bound of the value function to determine a gap with respect to the lower bound in order to qualitatively check whether the regressors are suitable;
4. analysing the relationship with the fuzzy Q-iteration using Look-Up-Tables (LUT) proposed in (Busoniu et al. 2010), showing that ADP-LP can avoid value iteration yielding identical results.

In order to check the validity of the approach in engineering applications, an experimental setup with swing-up and stabilisation of an inverted pendulum is presented.

The structure of the chapter is as follows: the next section will discuss preliminary definitions and basic concepts, as well as the problem statement. Section 6.3 will discuss and define all relevant ingredients of the problem setup. The proposed methodology for regressor choice and regularisation will be discussed in Section 6.4, the computation of a value function upper bound will review in Section 6.5 and a discussion on approximation quality, computational and implementation issues will be laid on Section 6.6. Section 6.7 will present examples and experiments validating the proposal.

6.2 Preliminares

Let us consider the following (deterministic) discrete non-linear dynamic system:

$$x_+ = f(x, u) \tag{6.1}$$

where $x \in \mathbb{X}$ is the state space, $u \in \mathbb{U}$ is the input and $x_+ \in \mathbb{X}$ denotes the successor state so that $f : \mathbb{X} \times \mathbb{U} \mapsto \mathbb{X}$, and $\mathbb{U}$ will denote the set of valid control actions.

A policy $u = \pi(x)$ is a function $\mathbb{X} \to \mathbb{U}$ that represents the closed-loop controller of the system achieving dynamics $x_+ = f(x, \pi(x))$.

The cost $V_\pi : \mathbb{X} \mapsto \mathbb{R}$, associated to a policy $\pi(x)$ starting from an initial state x_0 will be defined as:

$$V_\pi(x_0) := \sum_{k=0}^{\infty} \gamma^k L(x_k, \pi(x_k)) \tag{6.2}$$

being x_k the state at time instant k, $L(x, u)$ is a scalar function $\mathbb{X} \times \mathbb{U} \to \mathbb{R}$, also known as "immediate cost" and $0 < \gamma < 1$ is a discount factor. In the sequel, we will assume $L(x, u) \geq 0$ for all $(x, u) \in \mathbb{X} \times \mathbb{U}$.

Note that different variations of later-stated optimal control problems and their respective solutions will arise, depending on whether $\mathbb{X}$ and $\mathbb{U}$ contain a finite number of elements or not (continuous state/action spaces in the latter case, for instance).

The aim of optimal control is to find an optimal policy $\pi^*(x)$ minimising $V_\pi(x)$. In this sense, it is well-known that the cost of an arbitrary policy must satisfy Bellman equation (Bertsekas 2017; Lewis and Vrabie 2009):

$$V_\pi(x) = L(x, \pi(x)) + \gamma V_\pi(f(x, \pi(x))) \tag{6.3}$$

and the value function of the optimal policy, denoted as $V^*(x)$, verifies:

$$V^*(x) = TV^*(x) \tag{6.4}$$

where the Bellman operator T is defined as:

$$TV(x) := \min_{u \in \mathbb{U}} \left(L(x, u) + \gamma V(f(x, u)) \right) \tag{6.5}$$

Now, from $V^*(x)$, the optimal policy can be obtained with:

$$\pi^*(x) = \arg\min_{u\in\mathbb{U}} \left(L(x,u) + \gamma V^*(f(x,u))\right). \tag{6.6}$$

Some developments require, often just as intermediate steps, to solve the so-called "policy evaluation" problem in which the value function V_π of a given non-optimal policy must be obtained.

6.2.1 Approximate Dynamic Programming

Consider now a function approximator $V(x,\theta)$ where θ is a vector of adjustable parameters. Based on Bellman's equation (6.3), given a policy $\pi(x)$, if we define the Bellman residual (Lagoudakis and Parr 2003) as:

$$\epsilon_\pi(x,\theta) := V_\pi(x,\theta) - L(x,\pi(x)) - \gamma V_\pi(f(x,\pi(x)),\theta) \tag{6.7}$$

then, the approximated solution to the "policy evaluation" problem, given policy $\pi(x)$, is given by:

$$\theta^*_\pi := \arg\min_\theta \|\epsilon_\pi(x,\theta)\|^2 \tag{6.8}$$

where $\|\epsilon_\pi(x,\theta)\|^2 := \int_{\mathbb{X}} \epsilon_\pi(x,\theta)^2$, incorporating maybe some state-dependent weighting function (omitted for notational simplicity). The integral should be understood as a sum in discrete state spaces; also, in continuous state-spaces, to avoid numerical integration steps, the integral is often understood as the sum of $\epsilon(x_k,\theta)^2$ over a set of fitting points $\{x_1,\dots,x_N\}$.

Let us consider a parametrisation of V_π linear in θ, i.e.,

$$V_\pi(x,\theta) = \phi^T(x)\theta \tag{6.9}$$

where elements of vector $\phi(\cdot)$ will be denoted as *regressors*.

With the parametrisation (6.9), the above minimisation (6.8) is a Least Squares problem because:

$$\epsilon_\pi(x,\theta) = \left(\phi(x) - \gamma\phi(f(x,\pi(x))\right)^T \theta - L(x,\pi(x)) \tag{6.10}$$

Other nonlinear parametrisations might need computing gradients (Baird and Moore 1999) to carry out the minimisation in (6.8), with the risk of being trapped in local minima.

Analogously, given the value function $V_\pi(x, \theta^*_\pi)$, we can obtain an approximate *policy improvement* by solving:

$$\pi_+(x, \theta^*_\pi) := \arg\min_{u \in \mathbb{U}} \left(L(x,u) + \gamma V_\pi(f(x,u), \theta^*_\pi) \right), \tag{6.11}$$

Note that the parametrisation of π_+ in (6.11) is, in a general case, not "explicit" if there is not a closed-form solution to the minimisation problem at the right-hand side of the equality. However, from a formal point of view, π_+ does indeed depend on θ^*_π. Note, too, that the policy improvement is "approximate" because V_π itself is an approximation of the true value function, unless the parametrisation is flexible enough so that $\|\epsilon_\pi(\cdot,\cdot)\| \equiv 0$.

Policy Iteration: The iterative execution of the two previous steps: *policy evaluation* and *policy improvement* is an algorithm known as (fitted) Policy Iteration (PI) (Lewis and Vrabie 2009; Lagoudakis and Parr 2003).

Value Iteration: As an alternative to PI, from a initial estimate of the value function $V(x, \theta_0)$, an iterative procedure can be implemented as follows:

$$\theta_{j+1} := \arg\min_{\hat{\theta}} \left\| V(x, \hat{\theta}) - \min_{u \in \mathbb{U}} \left(L(x,u) + \gamma V(f(x,u), \theta_j) \right) \right\|^2 \tag{6.12}$$

where the $\|\cdot\|$ notation denotes, as in (6.8), the average over the state space. This is also known as (fitted) *value iteration* (VI), see (Lewis and Vrabie 2009; Munos and Szepesvári 2008). Contrarily to PI, there is no need in VI of involving any policy π in the computations arising from (6.12).

PI/VI Convergence

It is well-known that PI and VI algorithms, in its discrete implementation (non-approximated), where the parameters are indeed implemented with tables (Sutton and Barto 2018) converge to the optimal value function and policy. Also, it is well-known that in a general case, the approximated version of it, with continuous states, convergence is not guaranteed any more. To the authors' knowledge, only in very particular scenarios, where contractivity of the composition of the Bellman operator T plus projection to the regressor's column space is satisfied, we can guarantee convergence to a given " fixed point" that could be close enough to the optimal one to produce a reasonably good policy for a specific application and parametrisation (Busoniu et al. 2010). The main issue is that to prove contractivity for a given functional approximator $V(x, \theta)$ is difficult or even impossible, because theorems related with them are not usually constructive (see Busoniu et al. 2010 for details). The required contractivity can be proved only in very particular cases, such as using a Look-Up-Table (LUT) with as many regressors as available data (one per data point) (Busoniu et al. 2010). In practice, this requires a regular discretisation of data along the state space, which easily falls into the a.k.a *curse of dimensionality* for high dimensional systems. It is obvious that also the large amount of regressors can represent a drawback in many real scenarios. As an alternative approach to avoid convergence issues, monotonic gradient-descent minimisation of the Bellman residual (6.7) may be pursued in some cases; however, a good initialisation to avoid local minima, as well as an efficient computation of the gradient of (6.11) is needed (Díaz, Armesto, and Sala 2020).

Linear Programming approaches

Linear programming can, indeed, be used to get solutions for *exact* dynamic programming in discrete state and input spaces (Manne 1960; Sutton and Barto 1998). In approximate programming, to avoid PI/VI convergence problems, in (De Farias and Van Roy 2003), authors provide a lower bound estimation of the value function that verifies:

$$V(x, \theta) \leq L(x, u) + \gamma V(f(x, u), \theta) \quad \forall (x, u) \in \mathbb{X} \times \mathbb{U} \tag{6.13}$$

Obviously, (6.13) is equivalent to:

$$V(x, \theta) \leq \min_{u \in \mathbb{U}} \left(L(x, u) + \gamma V(f(x, u), \theta) \right) \tag{6.14}$$

i.e., $V(x, \theta) \leq TV(x, \theta)$ for all $x \in \mathbb{X}$, which is consistent with (6.4), but replacing equalities by inequalities. As $\mathbb{X} \times \mathbb{U}$ is finite, we can write (6.13) as a finite set of inequalities.

The authors of (De Farias and Van Roy 2003) prove that, indeed, any $V(x, \theta)$ fulfilling (6.13) is a lower bound of the optimal cost function, i.e., $V(x, \theta) \leq V^*(x)$ in the case $\gamma < 1$; indeed, T is a monotonic and contractive operator so

$V \leq TV \leq T^2V \leq \cdots \leq T^\infty V = V^*$, see the cited work and references therein for details.

If the function approximator is linear in parameters, then maximization of a weighted average of V over the state space, subject to (6.13), can be solved as a LP problem, because the inequality constraints can be written as:

$$\Big(\phi^T(x) - \gamma\phi^T(f(x,u))\Big) \cdot \theta \leq L(x,u) \quad \forall(x,u) \in \mathbb{X} \times \mathbb{U} \tag{6.15}$$

The reader is referred to (De Farias and Van Roy 2003) for further details.

6.2.2 Problem statement

The above discussion has presented well known ideas on ADP. As previously discussed, PI/VI have convergence issues, which might be difficult to know in advance for a given parametrisation; gradient-descent minimisation of the Bellman residual can be trapped in local minima, so initialisation may be a crucial decision. On the other hand, the LP approach to ADP provides theoretically guaranteed lower bounds for finite state spaces.

This chapter will exploit the LP approach to ADP. However, the said LP approach needs some refinements to provide meaningful and numerically reliable solutions in practice which, in many cases, involve continuous state and action spaces.

Terminal ingredients when the trajectories leave the region of interest will be also incorporated to the methodology (otherwise, the infinite-time problem would be ill-defined unless the region of interest is invariant). Previous ideas in this respect appear in (Díaz, Armesto, and Sala 2019).

In addition to this, an upper bound estimate might be convenient to provide a guess how far our cost estimates are and consequently decide whether or not the regressors are appropriate for our application. This upper bound can be obtained from the policy associated policy of the lower bound estimate and it can be iteratively improved with PI until a suitable stopping criteria is met.

This chapter will examine the issues which might end up in unbounded or ill-conditioned solutions as a consequence of poor dataset/regressor choices; as a result, some methodological proposals for regressor and data arrangement will be provided. When the number of regressors approaches the number of data points, our proposal will encompass earlier ones based on interpolative lookup tables; the relationship with them and the PI/VI options used for its adjustment will be, too, subject to scrutiny.

6.3 ADP via linear programming: problem setup

In the sequel, we will assume as available for the learning task a dataset $\mathcal{D} \subseteq \mathbb{D} \times \mathbb{U} \times \mathbb{X}$, consisting on N triplets of data

$$\mathcal{D} := \{(x_1, u_1, x_{1+}), (x_2, u_2, x_{2+}), \dots, (x_N, u_N, x_{N+})\} \tag{6.16}$$

where $x_{k+} = f(x_k, u_k)$ and the set $\mathbb{D} \subseteq \mathbb{X}$ will be understood as a "region of interest" in the state space with data availability, i.e., assuming that the dataset covers $\mathbb{D}$ "densely enough". We will denote the state space region with no data availability as $\mathbb{T} := \mathbb{X} \sim \mathbb{D}$.

The elements of a triplet $\xi := (x, u, x_+) \in \mathcal{D}$ will be denoted, from left to right, as 'source state', 'action' and 'successor state'. When speaking about x_+, we will refer to x as its 'predecessor'. A symbolic representation of the relevant sets[2] $\mathbb{X}$, $\mathbb{D}$ and $\mathbb{T}$, jointly with a few data points in $\mathcal{D}$ appear in Figure 6.1. We will assume that sets $\mathbb{X}$, $\mathbb{T}$ and $\mathbb{D}$ are such that the successor of any source state in $\mathbb{D}$ lies in $\mathbb{X}$.

Note that the order of data is, in principle, arbitrary, so that they will not be assumed to be sorted as a temporal sequence. The dataset can be obtained either from simulation, if a theoretical model of (6.1) is available, or via experimental data acquisition. In the present developments, we will assume the data set is fixed, without any exploration-exploitation decision to be made in the ADP optimisation. Nevertheless, as later discussed, some choices of dataset and regressors will lead to ill-posed ADP formulations so, in the eventuality of such a case, taking new data samples may be advisable as a possible remedy.

[2]Note that in dynamic programming with discrete state and action spaces, we may have $\mathbb{D} \equiv \mathbb{X}$, $\mathbb{T} = \emptyset$. This situation poses no particular problem in the LP-based setting addressed in this manuscript.

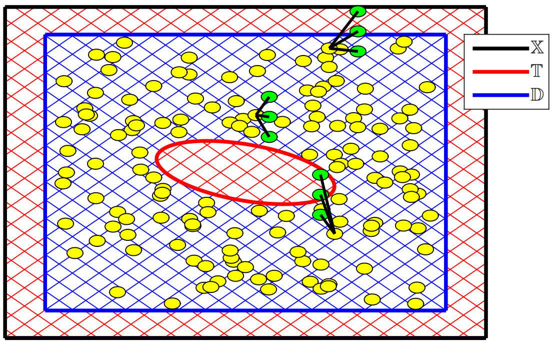


Figure 6.1: Illustration of the different sets involved in an optimal control problem ($\mathbb{X} = \mathbb{T} \cup \mathbb{D}$). The yellow circles in the data-availability region $\mathbb{D}$ indicate source states, the green ones indicate successor ones (of just three cases, to avoid clutter).

6.3.1 Terminal ingredients

In the developments in this chapter, the objective of the ADP is learning an estimate of the value function of the optimal control problem in the region of interest $\mathbb{D}$. However, this needs to be suitably defined, because, by definition, there are no data in $\mathcal{D}$ with source states in $\mathbb{T}$, so (6.13) cannot be stated at any point in $\mathbb{T}$.

In order to deal with this situation, we will understand $\mathbb{T}$ as a set of terminal states (terminal set), so the value function $V(x)$ for $x \in \mathbb{T}$ will be known, by assumption, equal to $V_{\mathbb{T}}(x)$, to be denoted as terminal cost, and *not* approximated by $V(x, \theta)$.

The use of terminal sets is common practice in model predictive control (MPC) (Camacho and Bordons 2013), where $\mathbb{T}$ is an invariant set under a non-saturating linear quadratic regulator; terminal states are also commonplace in discrete dynamic programming problems where a final reward/punishment is received when reaching some absorbing states (Sutton and Barto 2018). These problems end up posing an optimal control problem of reaching $\mathbb{T}$ while minimising

$$V_\pi(x_0) = V_{\mathbb{T}}(x_\tau) + \sum_{k=0}^{\tau-1} \gamma^k \bar{L}(x_k, \pi(x_k))$$

where final time τ is the reaching time, i.e., $x_k \in \mathbb{D}$ for $k < \tau$ and $x_\tau \in \mathbb{T}$, or infinity if $\mathbb{T}$ is not reached under policy π.

With no loss of generality, the terminal cost $V_\mathbb{T}(x)$ can be added to the immediate cost when reaching $\mathbb{T}$, i.e., defining

$$L(x,u) = \begin{cases} \bar{L}(x,u) & x \in \mathbb{D}, f(x,u) \in \mathbb{D} \\ \bar{L}(x,u) + \gamma V_\mathbb{T}(f(x,u)) & x \in \mathbb{D}, f(x,u) \in \mathbb{T} \\ 0 & x \in \mathbb{T} \end{cases} \tag{6.17}$$

and assuming absorbing terminal states, i.e., $f(x,u) := x$ for $x \in \mathbb{T}$ we have that $V_\pi(x) \equiv 0$ for all $x \in \mathbb{T}$ for any policy. Thus, with this notation we can express the above reaching control problem as an equivalent infinite-time one with cost L as stated in Section 6.2.

A symbolic representation of the relevant sets $\mathbb{X}$, $\mathbb{D}$ and $\mathbb{T}$, jointly with a few data points in $\mathcal{D}$ appear in Figure 6.1. In a control problem, the “inner” ellipsoidal terminal set would represent a region where a good performance controller is known (for instance, a linearised LQR one, which is the standard option in stable linear MPC (Goodwin, Seron, and De Doná 2006)). The “outer” terminal set would have a “large” terminal cost associated to, say, constraint violations.

6.3.2 LP constraints and cost index

Formally, with the above-defined dataset and terminal ingredients considered, the constraints (6.13) will be stated, over the data set, for $k = 1, \dots N$, as:

$$V(x_k, \theta) \le L(x_k, u_k) + \gamma V(x_{k+}, \theta) \tag{6.18}$$

if $x_{k+} \in \mathbb{D}$, and

$$V(x_k, \theta) \le L(x_k, u_k) \tag{6.19}$$

if $x_{k+} \in \mathbb{T}$. Note that, actually, if regressors are set so that $\psi(x) = 0$ for $x \in \mathbb{T}$ then (6.19) and (6.18) are actually identical. Thus, the separated notation is just a way to emphasise that (6.18) involves a function of the adjustable parameters in both sides of the Bellman inequality, whereas, in data points whose successor state is terminal, then only a plain upper bound to the value function is needed, as expressed by (6.19).

Under the above constraints, the cost index to maximise will be redefined to be:

$$M(\theta) = \int_{\mathbb{X}} \vartheta(x) V(x,\theta)\, dx \tag{6.20}$$

where $\vartheta(x)$ is an arbitrary positive weighting to emphasise the adjustment in a certain region of the state space.

ADP-LP problem Once regressors $\phi(x)$ for the function approximator are chosen, the cost (6.20) can be expressed as a linear-in-parameter expression:

$$M(\theta) = \left(\int_{\mathbb{X}} \vartheta(x) \phi^T(x)\, dx\right) \theta := c^T \theta \tag{6.21}$$

where the term between the parentheses is just a vector, once the integral is evaluated. The integral may be consider a "symbolic" expression to be approximated by a finite sum or, alternatively, it can be directly evaluated with numerical or symbolic software for a given choice of $\mathbb{X}$, $\vartheta(\cdot)$ and $\phi(\cdot)$.

Again, if the linear-in-parameter expression of (6.18)–(6.19) is written as:

$$\left(\phi^T(x_k) - \gamma \phi^T(x_{k+})\right) \theta \leq L(x_k, u_k), \quad \text{if } x_{k+} \in \mathbb{D} \tag{6.22}$$

$$\phi^T(x_k)\theta \leq L(x_k, u_k), \quad \text{if } x_{k+} \in \mathbb{T} \tag{6.23}$$

then, from the above discussion, the value function approximation estimation will be carried out by solving the approximate dynamic programming problem of maximising (6.21) subject to (6.22)–(6.23), which, trivially, can be expressed in compact notation as $A\theta \leq b$ for some matrix A and column vector b.

6.3.3 Regularity and well-posedness of ADP-LP

Although inequalities (6.18) may be posed for any desired linear-in-parameter function approximator (regressor choice in (6.9)), there are situations in which the resulting LP optimisation problem yields an unbounded or badly-conditioned result. We will provide means to detect such cases, as well as some methodological guidelines to suitably set up the ADP-LP problem (6.21)–(6.23), so that these badly-posed problems can be avoided.

Let us consider the above ADP-LP problem with constraints $A\theta \leq b$. Matrix A has size $N \times M$ and its k-th row, if $x_{k+} \in \mathbb{D}$, is given by:

$$\phi^T(x_k) - \gamma\phi^T(x_{k+}) \tag{6.24}$$

Obviously, if $x_{k+} \in \mathbb{T}$, the corresponding row of A will be just $\phi^T(x_k)$. Vector b, of size $N \times 1$, will have $L(x_k, u_k)$ as the element in position k.

In the sequel, we will assume regressors are non-negative, i.e., $\phi(x) \geq \mathbf{0}$ for all $x \in \mathbb{D}$. Without loss of generality, we will assume, too, that $\max_{x\in\mathbb{D}} \phi(x) = 1$, i.e., they are normalised to maximum value of one over the region $\mathbb{D}$.

Note that $\theta = \mathbf{0}$ will always be a feasible solution, because of the assumption that $L(x, u) \geq 0$ for all x and u.

Let us now describe the different issues that may arise when solving the ADP-LP problem:

Unbounded solutions

If there exists a column of A, denoted as A_j for column j, where all elements are non-positive (obviously, that can only happen in rows with the structure (6.24), i.e., involving non-terminal states only), then the LP solution will be unbounded because the associated θ_j can increase without limits as $A_j\theta_j < 0$ for all elements as θ_j increases. Thus, non-positive (or very small column-wise maximum values) should be avoided.

Ill-conditioned solutions

If a column of A is positive, but just by a very small amount, i.e. $\max A_j$ is small, then the optimal value of the associated parameter θ_j will be large. In this case, it will not be unbounded, as in the above paragraph, but still large compared to the rest. That will, in many cases, render solutions where “small” values for the value function estimate $V(x, \theta)$ will be computed via addition or subtraction of large regressors. For instance, if $V(1) = 3 = 5002\phi_1(1) - 5032\phi_2(1)$, then errors in the estimate of any of these parameters would entail large variations in the estimate of $V(x)$. This situation will produce heavy distortions on the value function estimates for intermediate points not belonging to $\mathcal{D}$ that should be avoided.

Rank deficiency

If matrix A were not full column rank, there would be a null space of parameters enabling arbitrary large values of parameters to fulfil the inequality constraints. If c is not orthogonal to the said null space, that would lead to unbounded values for $M(\theta^*)$, conversely, if c is orthogonal to the null space, bounded solutions of $M(\theta^*)$ would be produced but with unbounded feasible values of θ^*. Occurrence of these issues is, of course, to be avoided.

6.4 Regressor choice/regularisation methodology

The above issues on the LP problem are caused by a defective choice of regressors or data points. Let us analyse, then, the root underlying issues in such choice and suggest solutions to them.

6.4.1 Regressor activation

A possible cause of ill-conditioned LP problems may be the non-activation of regressors, i.e., even if they are by assumption "active" somewhere in $\mathbb{D}$ reaching value 1, maybe the dataset does not sufficiently "excite" them. Then, the associated parameters may attain a large value in the LP optimisation, distorting the resulting value function map $V(x, \theta)$, as above discussed.

Hence, if, for a given regressor ϕ_j,

$$\max_{1 \le k \le N} \phi_j(x_k) \le \beta_0 \tag{6.25}$$

being β_0 a given design parameter (approximately one), we will deem such regressor unfit for the given LP problem, as there is not even a single data point close to the maximum-activation region of the regressor.

Basically, these regressor activation issues may be solved by removing non-active regressors or, if the maximum activation region of the regressors is an interesting zone of the state space $\mathbb{D}$ that the data in $\mathcal{D}$ happen to not explore, then further data gathering in that region of the state space is, obviously, recommended.

6.4.2 Unused data points

If $\max_j \phi_j(x_k)$ is low, then no regressor is significantly active at the data point x_k, so the information provided by the referred data point will be, basically, unused. Actually, this is not a problem yielding ill-conditioned LP solutions but adding regressors in the vicinity of these unused data points may be advisable to use the information provided by these data.

6.4.3 Successor activation

If

$$\max_{1\leq k\leq N} (\phi_j(x_k) - \gamma\phi_j(x_{k+})) \leq 0. \tag{6.26}$$

then column j of the matrix A in the LP problem will have all its elements negative or zero. Thus, the ensuing LP problem would end up with an unbounded solution.

The interpretation of (6.26) is that, for all available data points, the regressor ϕ_j is active with larger intensity on the successor state than on the source state, see Figure 6.2(a) for a representative case.

In order to resolve this issue, a data point must be added in the point $\hat{x} \in \mathbb{D}$ where the problematic regressor attains a maximum value, i.e., $\phi_j(\hat{x}) = 1$, which exists by assumption. If the data collection phase is finished and that extra data point cannot be gathered, then the peak activation of the regressor should be shifted to, say, the nearest existing data point in $\mathcal{D}$. Note that, actually, this regularisation proposal has the beneficial side-effect of improving (6.25).

6.4.4 Flat regressors

Even if condition (6.26) does not occur for a given set of regressors, we might be "close" to it: a possible cause of badly-conditioned setups is having too "flat" regressors. If

$$\max_{1\leq k\leq N} (\phi_j(x_k) - \gamma\phi_j(x_{k+})) \leq \beta_1 \tag{6.27}$$

being β_1 a second design parameter (small, positive), then the difference in activation between a state and its successor for regressor ϕ_j is very small, i.e., the regressor takes almost the same value for the source states in the dataset and their successors, see Figure 6.2(b) for a representative case. As above discussed, the result will be a large estimation of parameter θ_j.

Lowering the discount factor γ is a way to avoid (6.27), but that "disguises" this regressor problem by changing the problem statement, which may have as a consequence the learning of a policy which is significantly different to the one originally sought. Thus, alternatives to modifying γ should be considered.

In that sense, differences in regressor values between close states with high activation may be increased by using a "power" parameter vector p, i.e., replacing the original $\phi_j(x)$ by $\phi_j(x)^{p_j}$, modifying the parameter $p_j > 1$ (regressor power coefficient) until

$$\max_{1\leq k\leq N} \left(\phi_j^{p_j}(x_k) - \gamma\phi_j^{p_j}(x_{k+})\right) = \beta_1 \tag{6.28}$$

Note that the power factor is equivalent, in the commonly used Gaussian radial-basis (RBF) regressor (Park and Sandberg 1991), to changing the variance parameter Σ_j, dividing it by p_j.

$$\phi_j(x) = e^{-(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j)} \tag{6.29}$$

6.4.5 Low input excitation

Another cause of the above problem (6.27) can be an unsuitable input excitation: maybe all tested u are too "small" to move x_k to a successor state sufficiently separated from it to yield significant differences in regressor values, see Figure 6.2(c) for a representative example.

If additional data gathering is still possible, if (6.27) occurs and deficient input excitation is suspected, then adding data obtained using larger input excitation is indeed recommended.

6.4.6 Linearly dependent regressors

If the condition number of A is below a given threshold, we are near the rank-deficiency issues discussed in the previous page.

An ill-conditioned A can be fixed by removing linearly dependent regressors, if these can be spotted by "inspection" (say, detecting two RBF neurons centred at the same point, for instance).

Such inspection step can be systematically avoided by explicitly requiring the projection of θ to be zero on the null-space of A. If we carry out the singular value

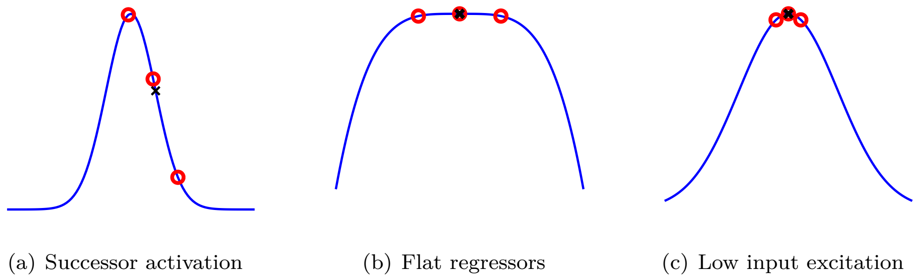

(a) Successor activation (b) Flat regressors (c) Low input excitation

Figure 6.2: Representative examples of regressor regularisation problem for cases such as: successor activation, flat regressors and deficient input excitation. The regressor is represented with a continuous blue line; source states causing regularisation problems are represented with a black cross and their successor states, for a given input excitation, are represented with red circles.

decomposition $A = \mathcal{U}\mathcal{S}\mathcal{V}^T$, selecting the columns of $\mathcal{V}$ associated to the singular values below a conditioning threshold, denoting that sub-matrix as $\mathcal{V}_{null}$, we can add the equality constraint $\mathcal{V}_{null}^T\theta = 0$ to the ADP-LP problem.

6.4.7 *The case of "interpolative" regressors*

Let us now denote as $\mathcal{D}_x \subseteq \mathbb{D}$ the finite set of source states in the dataset $\mathcal{D}$, with cardinality, say, N_x. Thus, by definition, for any state in $\mathcal{D}_x$ there is a triplet in the dataset $\mathcal{D}$ containing it as a first element, so $\mathcal{D}_x := \{\tilde{x} : \exists\, \xi = (\tilde{x}, u, x_+) \in \mathcal{D}\}$.

Let us define the matrix $P := [\phi(\tilde{x}_1) \ \ldots \ \phi(\tilde{x}_{N_x})]$, being now x_i the enumeration of the N_x elements of $\mathcal{D}_x$ in arbitrary order, with a slight abuse of notation. If P is square (i.e., $M = N_x$, as many regressors as source states) and invertible, then if we define

$$\widetilde{V} := \left(V(\tilde{x}_1), \ldots, V(\tilde{x}_{N_x})\right)^T, \tag{6.30}$$

we can set up the linear, invertible, change of variable given by $\widetilde{V} = P\theta$. Thus, with these regressors we can, actually, think of the *value function at source states*

$\widetilde{V}$ being the vector of adjustable parameters[3]. This kind of regressors will be denoted as *interpolative* regressors.

The above motivates the use of LUTs as function approximators for ADP-LP, which is, indeed, a sensible option in optimal control applications. This idea has a close relationship with LUT-based options in discrete dynamic programming problems and the fuzzy Q-iteration using LUT in (Busoniu et al. 2010) for approximated learning in continuous state spaces. The latter work proposes a standard value-iteration algorithm to adjust the LUT parameters, proving its convergence based on contractive mapping argumentations. Notwithstanding, 6.9.1 proves that, if such contractiveness assumptions holds, the optimal converged value-iteration solution coincides with the one found by solving our ADP-LP problem. Thus, our ADP-LP approach seems, in principle, better than the VI-LUT one, because, apart from providing an identical solution in this LUT case, it can find feasible solutions for alternate regressors even if the contractiveness assumptions fail (but at the expense of a larger memory footprint in computer implementation).

6.5 Computation of a value function upper bound

As it stands now, we have computed a sensible lower approximation $V(x, \theta_{LB})$ to the optimal value function. However, it would be desirable to known how "precise" our estimation is with a particular regressor structure. Computation of an "upper" bound to the value function would be, then, helpful: if the gap between upper and lower bounds is large, a denser set of regressors may be advisable.

The simplest upper bound would be accumulating the Bellman residuals (6.10) over trajectories, yielding a conservative worst-case estimate

$$V^*(x) \leq V(x, \theta_{LB}) + \frac{1}{1-\gamma} \max_x |\epsilon_\pi(x, \theta_{LB})| \tag{6.31}$$

where π would be, of course, the one in (6.11) setting the parameter vector to θ_{LB}.

[3]In fact, the number of regressors can be larger than N_x in some cases, while still yielding bounded solutions: the remaining degrees of freedom act by modifying the interpolation between the value of $V(x)$ at source states (details omitted for brevity). However, the possibility of "odd" shapes in these interpolated points does not seem a sensible option and, hence, using $M = N_x$ (i.e., fixing how intermediate values are interpolated) seems a sensible decision.

Another option is using the ideas in (Lincoln and Rantzer 2006; Rantzer 2006), where relaxed upper and lower bounds are computed[4] via iterations in the form:

$$\min_{\xi} \left(\underline{\nu}L(x,\xi) + \gamma V(f(x,\xi),\theta_{k-1})\right) \leq V(x,\theta_k) \\ \leq \overline{\nu}L(x,u) + \gamma V(f(x,u),\theta_k) \quad \forall (x,u) \in \mathbb{X}\times\mathbb{U} \quad (6.32)$$

with $\underline{\nu} \leq 1 \leq \overline{\nu}$ being relaxation parameters, trading off complexity of $V(x,\theta)$ against the gap between $\underline{\nu}$ and $\overline{\nu}$. In this way, (6.32) embeds (6.13) in value-iteration settings.

The cited works prove that

$$\min_{\{u_0,\ldots,u_k\}} \sum_{i=0}^{k} \gamma^i \underline{\nu} l(x_i,u_i) \leq V(x,\theta_k).$$

Thus, if a converged solution is found, denoting its parameters with θ_∞ then:

$$\underline{\nu}V^*(x) \leq V(x,\theta_\infty) \leq \overline{\nu}V^*(x)$$

However, these iterations may end up yielding infeasible constraints. Then instead of fixing $\overline{\nu}$ LP optimisation may be carried out seeking to minimise it in each iteration (feasibility is guaranteed); nevertheless, iterations may diverge. Convergence or recursive feasibility can only be guaranteed if the precision of the approximator is very high, see (Rantzer 2006). The reader is referred to (De Farias and Van Roy 2003; Rantzer 2006) for further details.

Another option is to compute an upper bound using an LMI initial controller as in (Díaz, Armesto, and Sala 2020). However, this provides a conservative solutions depending on the Takagi-Sugeno model (TS) used (Takagi and Sugeno 1985) and the value function (Q-function) is parametrised in a quadratic form. Unfortunately, this approach does not provide a proper upper-bound, because it is based on fitted value function approximation and LMIs might not provide valid solutions for bounded control inputs, as in the swing-up control of an inverted pendulum used in Section 6.7.

As an alternative the previous ideas, we propose to evaluate upper bounds via a linear-programming version of policy iteration, with the standard policy evaluation/improvement steps in an inequality-based version, to be detailed next.

[4]In our context, terminal-ingredient modifications would be needed; however, they are omitted in equation (6.32) for compactness of the notation, details left to the reader.

LP Policy evaluation (upper bound) Considering an arbitrary policy $\pi(x)$, an LP-based policy evaluation can be stated as:

$$L(x, \pi(x)) + \gamma V(f(x, \pi(x)), \theta_\pi) \leq V(x, \theta_\pi) \tag{6.33}$$

Indeed, if a feasible value of θ_π can be found, then $V(x, \theta_\pi) \geq V^*(x)$, by monotonicity and contractiveness of the Bellman operator ((De Farias and Van Roy 2003, Lemma 1) reversing signs). In this case, minimisation of (6.21), instead of the maximisation when computing lower bounds, will be carried out in the sequel, to obtain the lowest feasible upper bound (on average).

Under the assumptions in Section 6.3, with $\gamma < 1$, we can ensure that there exists a "constant" V_{max} that fulfils (6.33). Indeed, the successor state $f(x, u)$ can either lie in $\mathbb{D}$ or in $\mathbb{T}$. In each case, V_{max} should fulfil:

$$L(x, u) + \gamma V_{max} \leq V_{max} \quad if \ f(x, u) \in \mathbb{D} \tag{6.34}$$
$$L(x, u) + \gamma V_{\mathbb{T}}(f(x, u)) \leq V_{max} \quad if \ f(x, u) \in \mathbb{T} \tag{6.35}$$

thus, let us define:

$$\begin{aligned} V_{\mathbb{D},max} &:= \frac{1}{1-\gamma} \max_{(x,u)\in\mathbb{D}\times\mathbb{U}} L(x, u) \\ V_{\mathbb{T},max} &:= \max_{(x,u,f(x,u))\in\mathbb{D}\times\mathbb{U}\times\mathbb{T}} (L(x, u) + \gamma V_{\mathbb{T}}(f(x, u))) \end{aligned}$$

so we can assert that any constant V_{max} such that

$$V_{max} \geq \max(V_{\mathbb{D},max}, V_{\mathbb{T},max}) \tag{6.36}$$

fulfils (6.33).

As a conclusion, if the chosen regressor arrangement can fit a constant function, policy evaluation (6.33) will always be feasible.

Policy improvement Once a feasible upper bound for the policy evaluation is available, obviously, the new policy arising from (6.11) will for sure achieve a better upper bound; formally:

$$\begin{aligned} &L(x,\pi_+(x)) + \gamma V(f(x,\pi_+(x)),\theta_\pi) \\ &\qquad \le L(x,\pi(x)) + \gamma V(f(x,\pi(x)),\theta_\pi) \le V(x,\theta_\pi) \end{aligned} \tag{6.37}$$

Iteration scheme Based on the above discussion, we propose, given a starting policy $\pi_0(x)$, and its associated upper bound (6.33) $V(x,\theta_0)$, to carry out the iterations, starting with $k=0$ of a policy improvement step:

$$\pi_{k+1} := \arg\min_{u\in\mathbb{U}} \left(L(x,u) + \gamma V(f(x,u),\theta_k)\right) \tag{6.38}$$

and a policy evaluation one

$$L(x,\pi_{k+1}(x)) + \gamma V(f(x,\pi_{k+1}(x)),\theta_{k+1}) \le V(x,\theta_{k+1}) \tag{6.39}$$

$$V(x,\theta_{k+1}) \le V(x,\theta_k) \tag{6.40}$$

where inequality (6.40) has been added to force actual "point-wise" improvement of the bound (otherwise, the optimisation index (6.21) will make the bound decrease "on average" but point-wise increase cannot be excluded unless $V(x,\theta_{k+1}) \le V(x,\theta_k)$ is, too, enforced).

Summary: methodology proposal Our proposal is, hence,the following steps:

1. solve the lower-bound ADP-LP setup in sections 6.3 and 6.4, obtaining a parameter vector θ_{LB},
2. compute the policy π_0 with (6.11) form the lower-bound value function estimate.
3. evaluate an upper bound (6.33) for such policy, yielding the initial parameter vector θ_0.
4. Iterate (6.39)–(6.40) and (6.38) until a suitable stopping criteria is met. Denote the last parameter vector as θ_{UB}.

As a result of the above steps, we can *approximately* guarantee (see Section 6.6.1 for discussion on the meaning of such approximation)

$$V(x, \theta_{LB}) \leq V^*(x) \leq V(x, \theta_{UB}) \tag{6.41}$$

so if the gap between the upper and lower bounds is deemed excessive for a particular application, a regressor rearrangement (and increasing the number of them, possibly) is recommended. Note that the bounds (6.41) would be *exact* for a complete dataset (finite state and action spaces). Examples will show that the obtained upper bound in (6.41) is, indeed, better than (6.31).

Interpolative regressors As discussed in the 6.9.1 (see (6.53) and the discussion on it), if we have interpolative and contractive regressors, upper and lower bounds in (6.41) coincide; actually, they are the fixed-point solution of standard value iteration. Suitably arranged lookup-tables are such a case (Busoniu et al. 2010); our proposal can obtain separate upper and lower bounds in case contractiveness fails and VI does not converge.

6.6 Discussion

This section will discuss approximation, computational and implementation issues.

6.6.1 Approximation properties

As there exist a finite number of data points and regressors, the results of the Linear Programming optimisation are only an approximation of the optimal value function. Let us discuss the issues arising:

- Granularity in control actions u makes the computed value function to be sub-optimal with respect to a, say, continuous space of control actions as in linear LQR problems, for instance.
- Finite granularity in x forces fulfilment of the Bellman inequality only at the points in which data are available.

Thus, if we added more data points or more tested control actions at each state to a given dataset, more restrictions on the value function would appear so the obtained result of the LP optimisation will be lower: if we denote as $M^*(\mathcal{D}, \phi)$ the optimal cost solution of the ADP-LP problem with a dataset $\mathcal{D}$ and regressors ϕ, then, given two datasets $\mathcal{D}_2 \supseteq \mathcal{D}_1$, then $M^*(\mathcal{D}_2, \phi) \leq M^*(\mathcal{D}_1, \phi)$.

Let us define $\overline{\mathcal{D}}$ as the complete dataset containing all elements of $\mathbb{X} \times \mathbb{U}$ and their successors, i.e., for all $(x, u) \in \mathbb{X} \times \mathbb{U}$ the triplet $(x, u, f(x, u)) \in \overline{\mathcal{D}}$. Obviously, such complete dataset can be only actually built if both $\mathbb{X}$ and $\mathbb{U}$ are finite. For this complete dataset, we can assert $M^*(\overline{\mathcal{D}}, \phi) \leq M^*(\mathcal{D}, \phi)$ for any $\mathcal{D}$.

Now, recalling that the number of regressors is also limited, we can assert that adding more regressors would increase our ADP-LP estimate, i.e., if $\phi^{[2]} \supseteq \phi^{[1]}$, then $M^*(\mathcal{D}, \phi^{[1]}) \leq M^*(\mathcal{D}, \phi^{[2]})$.

In a finite case, if we had the complete dataset $\overline{\mathcal{D}}$ and a LUT regressor arrangement fitting any arbitrary function over $\mathbb{X}$, we would then be in the non-approximated LP case to the dynamic programming problem. This regressor will be denoted as $\overline{\Phi}_{\overline{\mathcal{D}}}$, and its associated optimal value of the cost index $M^*(\overline{\mathcal{D}}, \overline{\Phi}_{\overline{\mathcal{D}}})$.

Thus, the following diagram can be made:

$$\begin{array}{lccc} & \text{Scarce Dataset} & & \text{Complete Dataset} \\ \text{Poorer regressor} & M^*(\mathcal{D}, \phi^{[1]}) & \geq & M^*(\overline{\mathcal{D}}, \phi^{[1]}) \\ & \wedge\!| & & \wedge\!| \\ \text{Richer regressor} & M^*(\mathcal{D}, \phi^{[2]}) & & M^*(\overline{\mathcal{D}}, \overline{\Phi}_{\overline{\mathcal{D}}}) \end{array}$$

In summary, in a finite setting, the bottom-right option would be the exact DP solution and the top-right one would be the proven lower bound proposed in (De Farias and Van Roy 2003); on the other hand, in an infinite state/action space, the two options at the right would be elusive unattainable possibilities, to be approximated with a "sufficiently high" number of data triplets and a "sufficiently expressive" set of regressors, as we pursue in this manuscript.

Regarding the upper bound policy evaluation (6.33), parallel argumentations can be made; so, a similar diagram to the above can be set up, changing the signs of the inequalities. As discussion details are completely analogous, they have been omitted.

6.6.2 Computational/implementation issues

Data-based ADP has curse-of-dimensionality problems when exploring high dimensional state and input spaces, either by gridding or by random sampling or by gathering experimental data. Thus, proposals in this direction are, in principle, applicable to low-dimensional problems; this applies to the ADP-LP ideas in this chapter as well as the related ones in other literature we compare with.

Compared to ordinary least-squares based policy or value iteration in, say, (Busoniu et al. 2010; Lewis and Vrabie 2009) our proposal has two distinctive advantages: first, convergence issues are avoided without the need of contractiveness guarantees; second, we provide explicit state-dependent upper and lower bounds of the value function to assess the accuracy of our computations, which are better than, say, state-independent bounds based on the Bellman residual (6.31).

Modern efficient convex programming techniques can be used in LP solvers to produce the required parameter vectors in our proposal. Furthermore, sparsity improves execution speed and memory footprint of linear programming. Hence, it is advised to use sparse regressors, which amount to a “local” activation field. For instance, LUT’s are clearly the simplest sparse solution, but neural networks (trimmed to zero when activation is below a threshold) or piecewise-polynomial regressors can also be sensible options.

The result of the ADP-LP methodology is a value function estimate $V(x, \theta^*)$. As earlier discussed, in order to build the control action, the optimal one is the result of the computation (6.11). That computation can be carried out on-line if computational resources allow for it, or off-line and stored, say, on a control-map LUT to interpolate in later operation.

6.7 Examples and experimental results

In this section, simulations on linear systems will be presented in order to evaluate the performance obtained with the proposed methodology only using the lower bound. In these systems, the saturation of the control action is not considered. Additionally, we show some results applying the complete methodology(lower and upper bound) regarding the stabilisation of an inverted pendulum. In the inverted pendulum example, both the simulation and the experimental validation of the resulting controllers are presented.

6.7.1 Examples 1: First-order linear system

Consider the first-order linear system:

$$x_+ = Ax + Bu$$

with $A = 1$, $B = -0.5$, the available dataset (x,u) is in the operation region defined in: $x \in [-5, 5]$, $u \in [-0.5, 0.5]$. The state data x is in a regular grid of 55 points, and the control input u contains 21 equidistant points.

The immediate cost is defined as $L(x, u) = Qx^2 + Ru^2$, where $Q = 1$, $R = 10$. The discount factor is set to $\gamma = 0.9999$. In addition, a final cost is defined as $J_{out} = 100$ for those trajectories whose state leaves the interval $[-5, 5]$.

The purpose of this example is to compare the results of our proposal with the results given by the discounted linear quadratic regulator (LQR) problem. The discounted LQR comes from solving the modified Riccati equation with the discount factor γ:

$$S_+ = Q + \gamma A^T SA - \gamma^2 A^T SB(\gamma B^T SB + R)^{-1} B^T SA \tag{6.42}$$

We compare the results of our algorithm using two neural networks with 5 and 15 RBF neurons, respectively. The RBF neurons are uniformly distributed with $\Sigma = 1.11$ and 0.37, respectively. In addition, we compare our results with an interpolation table with the same number of samples that the data table (55 samples), following the ideas in (Busoniu et al. 2010).

Figure 6.3 represents the value function (or approximation lower bounds of the value function as a result of linear programming). It was obtained with the different proposes, as described next: discounted LQR in black, interpolation table (Look-Up Table, LUT) with 55 samples in blue, neural network(NN) with 5 neurons in green, and NN with 15 neurons in red. One can observe that when the regressor becomes more powerful(more adjustable parameters) the performance of the lower bound obtained by LP improves. Being the best of all one obtained by the interpolation table.

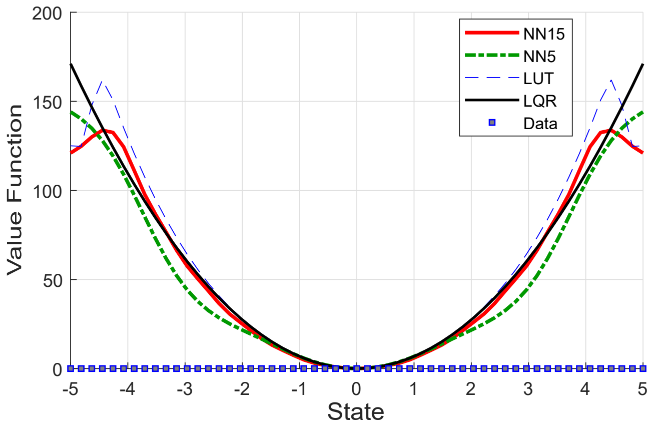


Figure 6.3: Lower bound of the value function.

Note that the look-up table can be considered as the closest to the "exact" optimum because it considers 55 adjustable parameters. In some places, the lower bound using LUT is higher than the LQR cost because the exploration was done by a finite number of saturated control actions (input range is $\{-0.5, 0.5\}$). The optimal linear regulator can produce intermediate actions and the presence of saturation also increases the optimal cost with respect to the LQR. There are also deformations at the left and right ends of the region because of the cost of leaving the operational region. This cost was intentionally set at a constant value lower than the LQR cost. With this fixed cost, the optimal policy near to the right and left ends is to give up and not to reach the origin. In other words, the optimal policy is to leave the modelling region. Obviously, if such an effect is not desired, it would be solved by placing a higher terminal penalty.

Figure 6.4 shows the control laws by each proposal, with the same colour code and labels using in the value function figure.

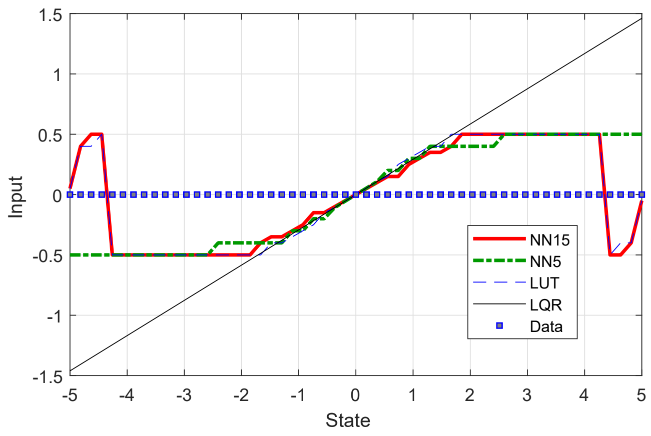


Figure 6.4: Control Input

It is observed that near the origin the controllers approximate the discrete linear controller, intuitively, we expected this result. The saturation and cost effect of leaving the modelling region $\mathbb{X}$ are evident in regions away from the origin, as discussed above.

6.7.2 Example 2: Second order system (Mountain-Car)

Consider the problem of reaching a target position by accelerating a mass with limited energy, described in Figure 6.5. Variations of this problem have also appeared in other literature references like (Sutton and Barto 1998; Kretchmar and Anderson 1997; Busoniu et al. 2010).

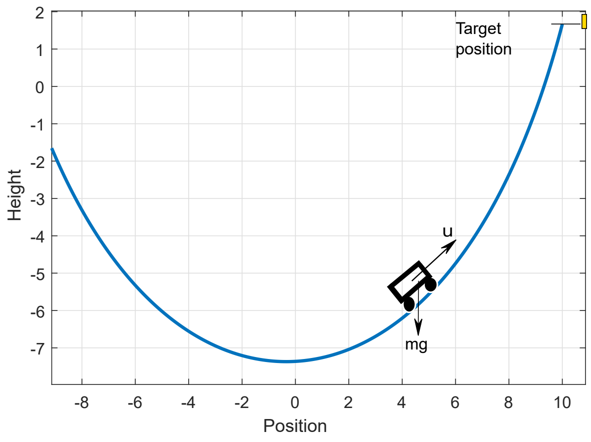


Figure 6.5: *Mountain-Car* system

This system can be represented by a second order model where the position is x, velocity is v, and u is the control action(tangential force):

$$\dot{x} = v, \;\; \dot{v} = u - mg \cdot \sin(0.12x + 0.04)$$

with a mass of $m = 1$ kg and gravity $g = 9.81$ m/s^2. The sinusoidal term denotes the effect on the tangential acceleration of the slope of the blue curve in Figure 6.5 when the cart is in position x.

The control objective is to reach the target position $(x \geq 10)$ with a control action in the range $[-2, 2]$. The immediate cost is defined as:

$$L(x, u) = 1 + 0.2u^2$$

The discount factor is $\gamma = 0.999$.

Note that, intentionally, the predefined limits of the control action are not sufficient to overcome gravity. Therefore, the optimum controller should increase its kinetic energy by swinging around the "valley" until it gets enough energy to go up. Since the target is "away" from the bottom, the result is an unstable dynamics around that bottom point. The aim of this example is to show how a neuronal controller can learn this strategy.

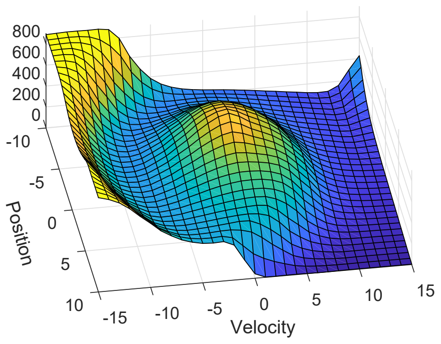
The modelling region $\mathbb{X}$ where the learning dataset was generated is the rectangle determined by the position interval $x \in [-10, 10]$ m, and the velocity interval $[-15, 15]$ m/s. The cost to leave the modelling area on the right side is 0, on the contrary if the car leaves on the left side the cost is 900. This value is set to be high enough with the aim that the trajectories avoid the left end. Additionally, this cost value should be chosen to avoid distorting the approximation of the value function of the figures 6.6(a) and 6.7(a) at interior points to $[-10, 10]$). The data covers an area with velocities less than 15 m/s, in the case that the absolute value of the velocity in a successor state exceeds 15 m/s the cost to leave $\mathbb{X}$ is also 900.

The methodology was evaluated with RBF regressors and with regressors based on bidimensional interpolation table that detailed below. The data grid was designed with 31 points on the position axis and 31 points on the velocity axis. The control input was discretised in 11 points. All grids are evenly-spaced.

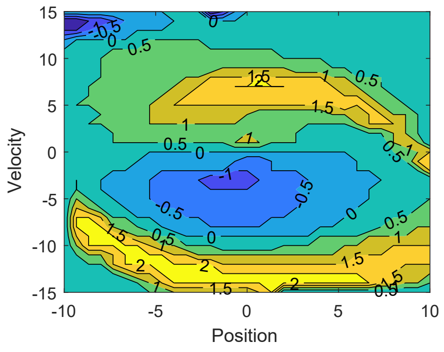
The results with a model of 15x15 evenly-spaced RBF neurons (with a parameter $\Sigma = \text{diag}(\{0.74, 1.11\})$, with a total of 225 adjustable parameters) are represented in Figure 6.6. It shows the lower bound of the value function, the controller obtained (contour map), the temporal response and the phase plane. The phase plane indicates the position and velocity trajectory until the right side is reached. The simulation is finished when the target position is reached. The controller was obtained by optimising at one step the equation (6.11) over the discrete set $\mathbb{X}$. Subsequently, the control action in intermediate points outside the initial dataset was obtained using an online u interpolation table. Although it would also be reasonable to optimise (6.11) online, but perhaps with a larger computational cost.

The results using the regressors associated with a bidimensional interpolation table (31x31 points, with 961 adjustable parameters) are shown in Figure 6.7. It is observed that the neural network has learned a similar control law with almost four times fewer adjustable parameters. An important aspect to consider in the control map is that it capable of correctly estimating the temporal difference related to the "slope" of the V map. This means the relationship between $V(x)$ and $V(x_+)$. The "constant" level differences in the value function do not produce significant changes in the resulting control law.

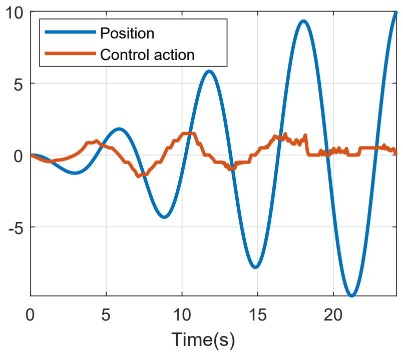
In simulations, we can observe that the results are not significantly different using a neural network with almost four times fewer adjustable parameters than the interpolation table. With the proposed methodology, the controller achieves to lift the car to the target position.



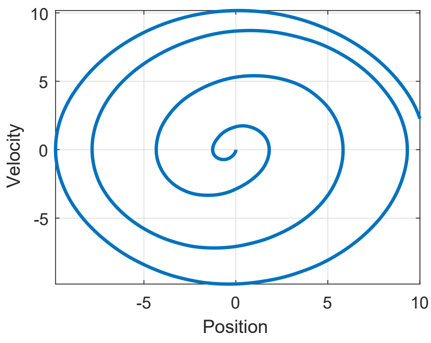
(a) Lower bound of the value function



(b) Control law

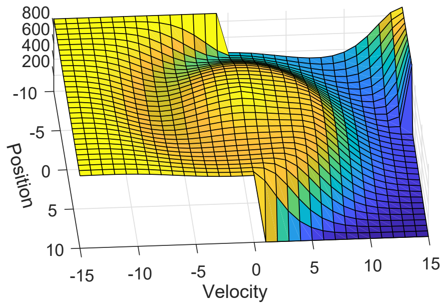


(c) Temporal response

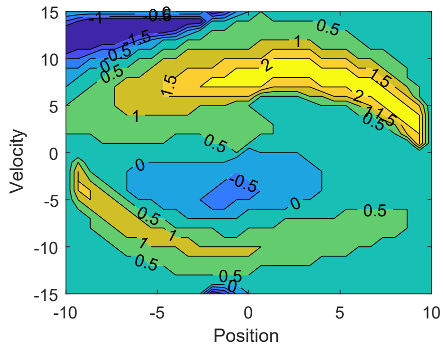


(d) Phase plane

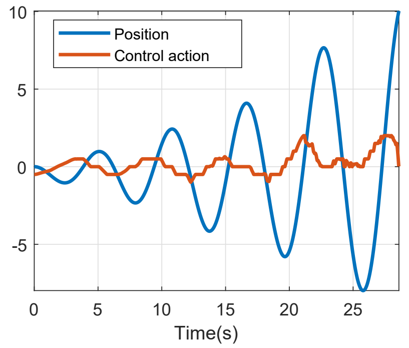
Figure 6.6: Control law and simulation with 15x15 RBF neurons.



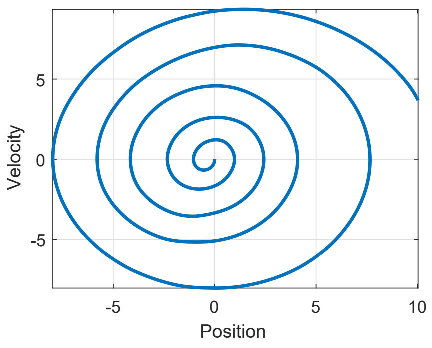
(a) Lower bound of the value function



(b) Control law



(c) Temporal response



(d) Phase plane

Figure 6.7: Control law and temporal response with interpolation table of 31x31.

The lower bound of the value function obtained with 961 adjustable parameters is better (closer to the optimal value function) than that obtained by the 15x15 neural network. Although the simulations and associated control actions are not significantly different and our proposal has approximately four times fewer adjustable parameters. The neural network with 15x15 neurons were chosen because lower granularities learned disturbed value functions that did not take the car out of the valley, and higher granularities performed similarly to the one presented[5].

In order to verify the comparative advantages of our proposal, the classic value iteration algorithm was implemented on the same problem. Since the interpolation table of 961 adjustable parameters verifies the necessary contractivity conditions (Busoniu et al. 2010). The VI Algorithm converges after approximately 250 iterations, basically to the same solution that our proposal finds (Figure 6.7). However, with a neural network with 15x15 RBF neurons these iterations do not converge. Therefore, it is not possible to apply algorithm VI with regressors with a reduced number of parameters. On the other hand, using linear programming it is possible to implement it without any problem.

6.7.3 Example 3: Path planning.

In this example, we consider the robot ABB IRB 360 with Delta configuration shown in Figure 6.8. The objective of the optimal control is to achieve a final configuration from an initial configuration. In order to get this objective, we use a kinematic control of the position (x, y, z) in the final effector. The immediate cost index is:

$$L(x, e_p) = ||\dot{x}||^2 + ||\dot{y}||^2 + ||\dot{z}||^2 + ||e_p||^2$$

where e_p is the position error with respect to the desired position $(x_d, y_d, z_d) = [-0.3 \quad 0.1 \quad 0.1]^T$. The discount factor is $\gamma = 0.999$. The workspace was defined in the range $\{-0.4 \leq x \leq 0.4, -0.4 \leq y \leq 0.4, 0 \leq z \leq 0.24\}$. The granularity of the workspace was established in 11x11x11. The control actions used are $\{-0.25, -0.125, 0, +0.125, +0.25\}$ m/s on each of the three Cartesian axes.

[5]The selection of the number of neurons is a compromise between quality of adjustment, simplicity of description and computational cost.

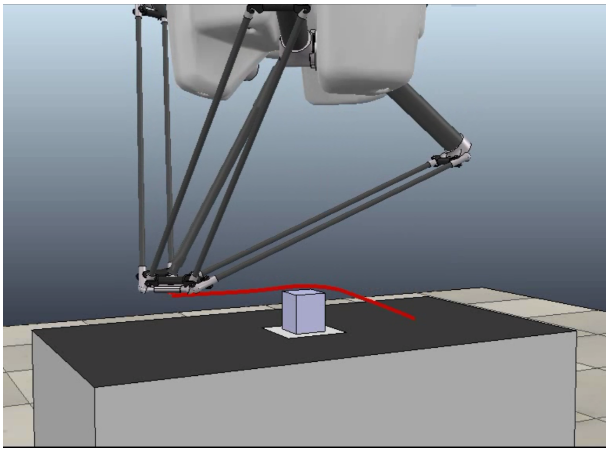


Figure 6.8: Robot ABB IRB 360

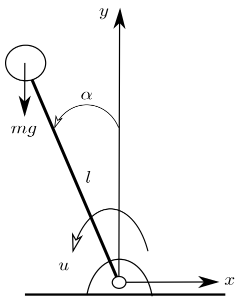
This workspace contains a prismatic obstacle and the robot must avoid collisions with this obstacle. Applying the methodology proposed in this chapter, the obstacle is considered outside the valid operating region. The size of the obstacle has increased in one cell of the previously proposed grid that is common in many path planning proposals (Latombe 2012). The robot structure does not collide with the internal obstacle when the final effector is at the edge of the enlarged prism, it was verified with the V-REP software (Rohmer, Singh, and Freese 2013).

The cost assigned to configurations that leave the workspace or collide with the enlarged prism is $J_{outside} = 1e5$. The regressor used is an interpolation table, where the nodes associated with the obstacle were removed to avoid unbounded solutions.

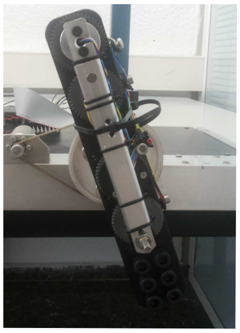
The figure 6.8 shows in red the path described by the robot's final effector from an initial position whose Cartesian coordinates are $[0.2 \ -0.2 \ 0.04]^T$.

6.7.4 Example 4: Inverted Pendulum

The goal is stabilising the upper (unstable) equilibrium point of an inverted pendulum, a schematic representation is shown in the Figure 6.9(a). The theoretical model of the pendulum is:



(a) Variables definition



(b) Real pendulum

Figure 6.9: Inverted pendulum: variables definition and real pendulum setup.

$$\dot{\alpha} = \omega \tag{6.43}$$

$$\dot{\omega} = \frac{1}{J}\left(mgl\sin(\alpha) - b\omega + \frac{K}{R}u\right) \tag{6.44}$$

where α is the pendulum angle and u the applied voltage to the DC motor. J is the mass inertia, m is the bar mass, l is the bar length, g is the gravity, b is the Coulomb friction and R is the motor electrical resistance and K the current-torque gain.

The objective is to apply the learnt controller to a physical pendulum prototype as shown in Figure 6.9(b). Thus, the parameters of the model (6.43)-(6.44) will be identified first from the experimental data. Such data are shown in Figure 6.10, which consists of a given input profile (bottom plot), generated from two sequences of a swing-up control law (increasing kinetic energy with $u = sign(\omega)$ plus a random low-pass filtered signal), a PD controller activated when position is close to the unstable equilibrium, and a disconnection letting the pendulum fall to the bottom stable equilibrium.

The only available sensor is a Hall-effect encoder measuring the angle α. The control sampling period is $\Delta t = 0.01$ s. In order to carry out state-feedback control, "measurements" of ω are needed. Thus, in this application, a suitably tuned Kalman filter for a double-integrator model has been coded running with a 10x oversampling, i.e. $\Delta t_{KF} = 0.001$ s, and the speed estimate every 10 samples has been used for control. Nevertheless, as the only actual measurement is the

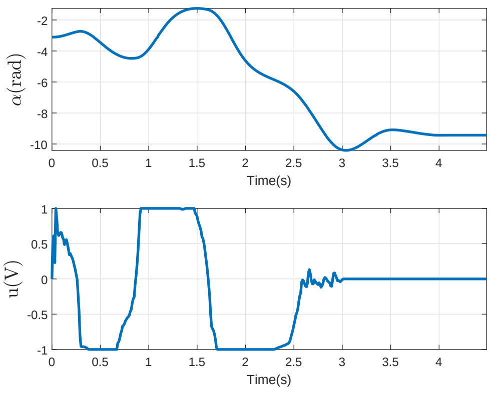


Figure 6.10: Test Data for identification.

position one, the identification has been carried out exclusively with position encoder data.

The resulting discrete-time model, with state vector $x = (\alpha, \omega)^T$ is:

$$\alpha_+ = \alpha + \Delta t \cdot \omega \tag{6.45}$$

$$\omega_+ = \omega + \Delta t(42.27 \sin(\alpha) - 2.27\omega + 24.21u) \tag{6.46}$$

achieved a 96.5% fit of the measured position over the validation data segment, see Figure 6.11 (the command `idnlgrey` of Matlab's R2017b System Identification Toolbox® has been used to identify the Euler discretisation of (6.43)-(6.44)). Dataset acquisition is performed in real time using a NI myRIO-1900 embedded device by National Instruments and LabVIEW 2015. A dead-zone compensation was added to approximately compensate for dry Coulomb friction phenomena in motor and gears.

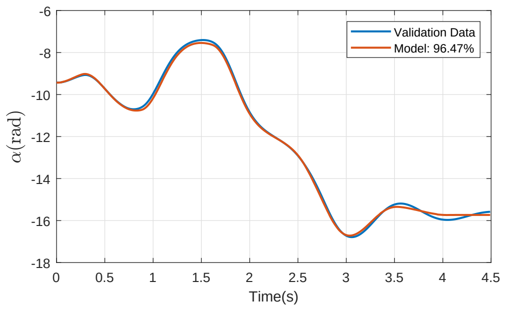


Figure 6.11: Validation data (output), and simulated identified model, with fit figures.

6.7.5 ADP-LP problem setup

The actual optimal control problem was set in order to have the standard LQR immediate cost, $\bar{L}(x,u) = x^T Q x + u^T R u$ with $Q = \begin{pmatrix} 10 & 0 \\ 0 & 0.1 \end{pmatrix}$, $R = 1$. The discount factor was set to $\gamma := 0.99$.

An ADP-LP problem will be posed and solved based on a dataset $\mathcal{D}$ originated from a uniform gridding 43×43 of the state space $[-\pi, \pi] \times [-15\pi, 15\pi]$, and a 11-point grid of the control action space $\mathbb{U} := [-1, 1]$. The successor states in $\mathcal{D}$ are generated by simulation of the identified model (6.45)-(6.46).

Terminal Ingredients The linearisation of (6.45)-(6.46) will be used to obtain a discounted-LQR terminal control law, with an associated quadratic value function $V_{LQR}(x) = x^T S x$, where $S = \begin{pmatrix} 265.61 & 19.96 \\ 19.96 & 2.47 \end{pmatrix}$ is the Riccati solution to:

$$K = \left(\gamma B^T S B + R\right)^{-1} \gamma B^T S A \tag{6.47}$$

$$S = \gamma A^T S \left(A - BK\right) + Q \tag{6.48}$$

and state feedback gain is $u = -\begin{bmatrix} 4.7117 & 0.5737 \end{bmatrix} x$.

In addition to this, we have chosen the ellipsoid $\mathbb{T}_1 := \{x : V_{LQR}(x) < 12.10\}$ as an inner region in which we will assume as optimal the discounted LQR solution; this ellipsoid region is depicted in red in latter value function and control map figures just in the middle of the figures. Indeed, this region provides a non-saturated LQR control-law. On the other hand, if trajectories exit the gridded state-space region, we will assume they enter an "outer" terminal region $\mathbb{T}_2 := \{x \in \mathbb{R}^2 : x \notin [-\pi, \pi] \times [-15\pi, 15\pi]\}$ and that an instant penalisation of $V_{\mathbb{T}_2} := 10000$ is considered, basically forcing the system to avoid entering $\mathbb{T}_2$. Thus, the terminal set will be defined to be $\mathbb{T} := \mathbb{T}_1 \cup \mathbb{T}_2$, and the terminal cost $V_{\mathbb{T}}(x)$ will be defined to be equal to $V_{LQR}(x)$ if $x \in \mathbb{T}_1$, and $V_{\mathbb{T}_2}$ if $x \in \mathbb{T}_2$. With these ingredients, the ADP-LP immediate cost (6.17) can be built.

As a last remark, the "inner" terminal set $\mathbb{T}_1$ may be empty, if it were desired for the controller to learn the optimal control law close to the origin. However, given that the dataset contains a finite set of points with a finite granularity of the control law, the learnt controllers will exhibit some kind of offset/chattering behaviour close to the origin due to approximation errors. This has been, indeed, the case with the system in this example; for brevity, no further details are discussed on this non-satisfactory behaviour, solved with the ellipsoidal $\mathbb{T}_1$.

Regressor choice Two regressor arrangements have been tested:

1. A two-dimensional 43×43 LUT (triangular membership functions).
2. A 15×15 uniformly-distributed grid of RBF neurons $\phi_j(x) = e^{-(x-\mu_j)^T \Sigma^{-1}(x-\mu_j)}$, normalised to unit sum.

The first regressor, with 1849 adjustable parameters will be assumed to be close to a "ground-truth" value function, albeit with the error sources discussed in Section 6.6 due to interpolation on a finite grid. However, our objective is testing how the RBF setup with 225 parameters can provide a reasonable value function estimate, in case the 1849-parameter LUT were unattainable due to its memory footprint, as it would happen in higher-dimensional problems or with denser grids.

We have generated the data gridding and uniformly distributed the RBF centroids so that a neuron centroid coincides with one of each three consecutive source states in the dataset. Also, we have removed the regressors (in both neural network and LUT) whose centroid (interpolation point) belongs to $\mathbb{T}$; otherwise unbounded solutions of the ADP-LP problem would have been obtained. A last decision in the regressor arrangement involved tuning the variance parameter Σ of the RBF neurons so that flat-regressor issues are absent; the finally chosen value for this parameter was $\Sigma = \text{diag}(0.0542, 12.1847)$. All criteria discussed in Section 6.3.3 were satisfactorily checked with this regressor proposal. Indeed, we intentionally generated other defective arrangements with different neuron centroid placement/overlap and neurons inside the inner terminal ellipsoid, and they produced the issues discussed in Section 6.4.

6.7.6 Simulated and experimental results

Value function and control map The proposed ADP-LP lower bound computations, as well as the subsequent upper bound ones yielded two estimates $V(x, \theta_{LB})$ and $V(x, \theta_{UB})$. As discussed in the 6.9.1, for the LUT regressors, both bounds coincide.

We, too, tried standard policy/value iteration setups; the 43x43 LUT was indeed contractive and worked perfectly, as proven in the 6.9.1. However, our 15x15 RBF and other RBF and LUT arrangements with similar number of adjustable parameters failed to converge under the referred widely-used algorithms, illustrating the advantages of the LP approach proposed here.

Figure 6.12 compares the value function of the 43x43 LUT (likely, the closest we can reasonably get to the ellusive "ground-truth" optimal value function with the given dataset) with the estimates $V(x, \theta_{LB})$ and $V(x, \theta_{UB})$ with our 15x15 RBF[6]. Note that the value function peaks at the lower equilibrium $(-\pi, 0)$, which is intuitively expected. Even if the green LUT result were not available, if the gap between the orange and blue surfaces were deemed excessive, that would advise increasing the flexibility of the chosen regressors with, say, more neurons.

The control map derived from $V(x, \theta_{UB})$ depicted in Figure 6.13 can be shown to resemble other energy-based swing-up controllers (Yoshida 1999) which increase the mechanical energy until it is enough to reach the upper position. Figure 6.14 depicts the 43x43 LUT control map, showing that the 15xs15 RBF achieved a control map that is close to it.

Time simulation and experiments Figure 6.15 depicts a representative example of the system trajectories with both simulated (red) and actual experiment (blue) with the controller arising from the value function $V(x, \theta_{UB})$ estimated by the 15x15 RBF. The simulated pendulum, produces a swing-up control law as previously discussed, where we can clearly appreciate that the system commutes to the terminal LQR control law at time instant $t \approx 4$ s. The real pendulum, produces a similar response.

The system identification was carried out with data at low speeds (the maximum speed on identification experiment data was 8 rad/s). However, the value function has been approximated in a region including speeds up to 47 rad/s. Thus, we are also interested in analysing how the learnt control map performs in high-speed

[6]The bound (6.31) proves that adding $3.31 \cdot 10^4$ to the orange lower bound yields a guaranteed upper bound; however, it is not shown in the figure because it is, basically, four times higher than the upper bound in the blue surface.

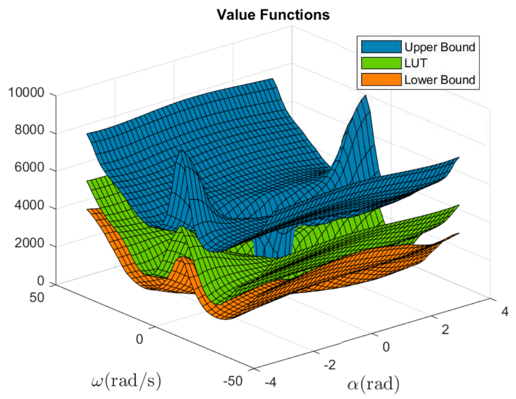


Figure 6.12: Value function estimates: [green] 43x43 LUT, [orange] 15x15 lower bound $V(x, \theta_{LB})$, [blue] 15x15 upper bound $V(x, \theta_{UB})$.

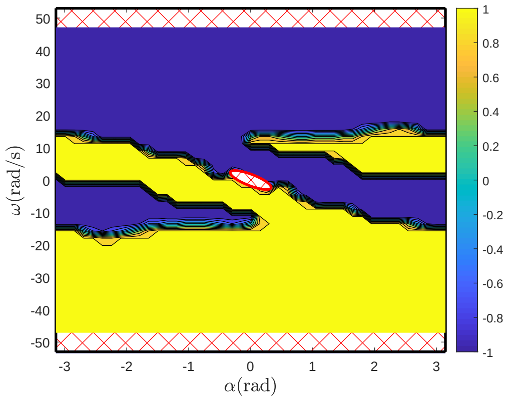


Figure 6.13: Control Policy with 15×15 RBF neurons using θ_{UB}, plus red-crossed region $\mathbb{T}$.

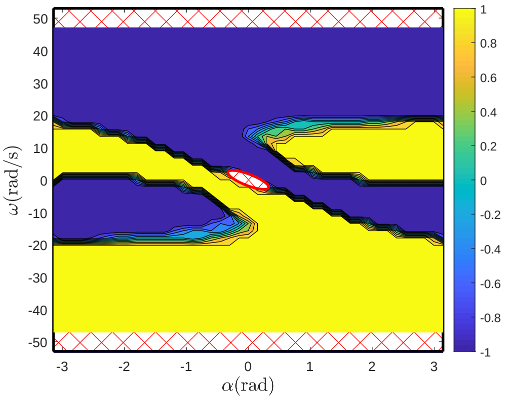


Figure 6.14: Control policy with LUT 43×43 (red-crossed region is $\mathbb{T}$).

situations. This is shown in Figure 6.16: a phase plane of the trajectory starting at point $x = (-\pi, 0)$ (whose time evolution is shown on Figure 6.15) is shown together with another trajectory starting at $x = (-\pi, 40)$).

It can be seen that the trajectory at high speeds has a satisfactory behaviour: the mechanical energy is being reduced by braking until the system has the correct amount of energy to enter the LQR terminal region.

6.8 Conclusions

Based on earlier linear-programming approaches to approximate dynamic programming, this chapter has presented a methodology which includes terminal ingredients and regressor regularisation to produce well-conditioned and reasonably accurate solutions to optimal control problems. The results are an approximate lower and upper bounds of the value function, so the gap between them can guide regressor choice. The validity of the approach has been experimentally tested.

Obviously, even if equally-spaced RBF neurons and LUTs have been used in the examples in this work, other neuron distribution of physics-based regressors might have as well been tested to try to obtain good controllers with a reduced number of parameters. Nevertheless, these problem-dependent considerations are implementation details outside of the scope of this work.

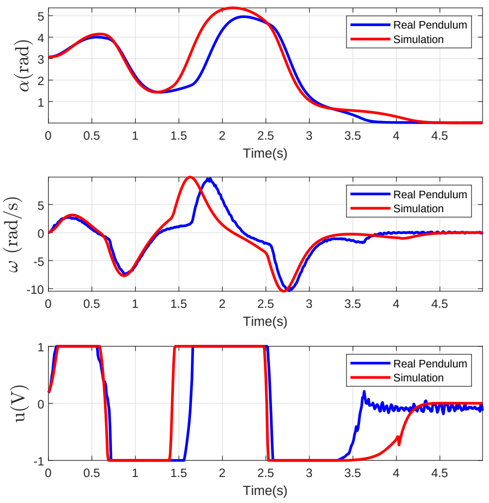


Figure 6.15: Simulation and experimental pendulum responses with a 15×15 RBF: position, speed and control action (top to bottom).

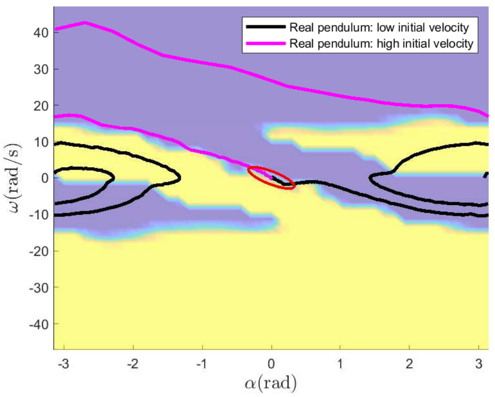


Figure 6.16: Phase planes of two trajectories with 15×15 RBF controller. The background image corresponds to Figure 6.13; inner LQR terminal set outlined with its boundary ellipsoid.

6.9 Appendix

6.9.1 Value iteration under interpolative regressors

As discussed in the main text, with a slight abuse of notation, in the rest of this section we will assume $\widetilde{V}$ defined in (6.30) to be the vector of adjustable parameters and $V(x, \widetilde{V})$ the function that interpolates between them.

If we have interpolative regressors, using the equivalent parametrisation $\widetilde{V}$, we can set up the asynchronous value iteration in (Busoniu et al. 2010), being l the iteration number:

$$\widetilde{V}_k^{[l+1]} := T_k V(\tilde{x}_k, \widetilde{V}^{[l]}) \quad k = 1, \dots, N_x \tag{6.49}$$

being the operator T_k defined as:

$$T_k V(\tilde{x}_k, \widetilde{V}) := \min_{u \in \mathbb{U}(\tilde{x}_k)} \Big(L(\tilde{x}_k, u) + \gamma V(f(\tilde{x}_k, u), \widetilde{V}) \Big) \tag{6.50}$$

where $\mathbb{U}(\tilde{x})$ denotes the set of actually tested control actions for the source state $\tilde{x} \in \mathcal{D}_x$ in the given dataset, i.e., $\mathbb{U}(\tilde{x}) := \{u : \exists \xi \ s.t.\ (\tilde{x}, u, \xi) \in \mathcal{D}\}$.

Now, computations arising from equation (6.49) will be short-handed to the notation:

$$\widetilde{V}^{[l+1]} := T_{\mathcal{D}} \widetilde{V}^{[l]} \tag{6.51}$$

in order to emphasise the application of the dataset-dependent Bellman operator, so its action transforms a value estimate over the source states, $\widetilde{V}^{[l]}$, to another one $\widetilde{V}^{[l+1]}$. Obviously, if the dataset $\mathcal{D}$ were "complete" (i.e., containing all possible transitions in it, in a finite case), then the dataset-specific Bellman operator would be just the standard Bellman operator, $T_{\overline{\mathcal{D}}} \equiv T$, widely used in literature, say, (Busoniu et al. 2010).

Obviously, if $T_{\mathcal{D}}$ is contractive, the iterations (6.51) will converge to a fixed point $\widetilde{V}^* = T_{\mathcal{D}} \widetilde{V}^*$.

Now, recall that the ADP-LP constraints, under the $\mathcal{D}$-complete regressors, can be equivalently stated as:

$$\widetilde{V} \leq T_{\mathcal{D}} \widetilde{V} \tag{6.52}$$

Hence, we can assert that if $T_{\mathfrak{D}}$ is contractive, the fixed point $\widetilde{V}^* = T_{\mathfrak{D}}\widetilde{V}^*$ of the value iteration (6.51) is a feasible solution of the ADP-LP problem posed in Section 6.3.2. If all coefficients multiplying $\widetilde{V}$ in the cost index $M(\widetilde{V})$ are positive, then the optimal ADP-LP solution coincides with $\widetilde{V}^*$.

Now, regarding the upper bound computations in Section 6.5, policy evaluation could be written as:

$$\widetilde{V}^{\pi} \geq \Big(L(\tilde{x}_k, \pi(\tilde{x}_k)) + \gamma V(f(\tilde{x}_k, \pi(\tilde{x}_k), \widetilde{V}) \Big) \tag{6.53}$$

if π were the policy associated to the converged $\widetilde{V}^*$, it is easy to realise that $\widetilde{V}^*$ is also a feasible solution of (6.53). In summary, in this case upper and lower bounds obtained with linear-programming methodologies in this paper coincide with the asynchronous value-iteration result.

Remark: Note that the contractiveness of $T_{\mathfrak{D}}$, plus positive weights, is a sufficient condition for the ADP-LP solution to obtain in one shot (no iterations) the converged VI solution. But, actually, it is straightforward to realise that, if weight coefficients are positive, under the constraints (6.52), LP will obtain a bounded fixed point $\tilde{V}^* = T_{\mathfrak{D}}\tilde{V}^*$ in all cases in which the LP problem renders feasible and bounded, without the explicit need of contractiveness: our LP proposal succeeds even in situations where value iteration fails.

Note, too, that, if regressors are positive, then monotonicity holds; if they add one, then $T_{\mathfrak{D}}(\tilde{V} + a) \leq (T_{\mathfrak{D}}\tilde{V}) + \gamma a$, so the Blackwell conditions (Blackwell 1965) for contrativeness hold. This is the case, for instance, in the interpolative LUT regressor proposal in (Busoniu et al. 2010, Chapter 4) where value iteration (6.51) was proposed as the parameter learning mechanism. Thus, the LP proposal obtains exactly the same results as these LUT regressors, but it also may render feasible even in non-contractive interpolative regressor choices (and, of course, as discussed in the main text, even if the chosen parametrisation is not interpolative). In summary, as a conclusion of this appendix, if VI works with interpolative regressors, so it will ADP-LP; it will work in other cases, too.

Chapter 7

An Incremental Approximate Dynamic Programming Methodology

The dynamic programming computes the optimal solution by solving the Bellman equation. The linear programming approach can be used to solve the Bellman equation; the optimal solution obtained by linear programming is exact if the state and input spaces are finite. However, when the number of states and control actions is large or continuous, the use of approximation functions is necessary. The linear programming with function approximation would result in a lower bound of the optimal value function. This Chapter proposes a methodology to use the approximate dynamic programming via linear programming to learn a suboptimal policy using data available only in the exploration region. Initially, the exploration is carried out with a predefined controller in order to collect data and applying approximate dynamic programming via linear programming. The exploration step progressively increments the learning region until a suboptimal policy is obtained. The methodology is applied to a discrete nonlinear system in order to compare the policy obtained with a methodology that uses the complete dataset of state and action spaces.

7.1 Introduction

The objective of this Chapter is to propose a methodology that will extend the ideas in Chapter 6 to approximately solve deterministic optimal control problems with ADP via LP using a set of sampled trajectories.

The structure of the Chapter is as follows: the next Section 7.2 states the problem statement, the problem setup will define in Section 7.3, and Section 7.4 will present a simulation example in order to validate the proposal.

7.2 Problem statement

This Chapter will use the LP approach to ADP. It argues that the dataset corresponds to exploration samples that are obtained with a predefined initial controller, the learned controller, or a combination of the above, unlike the previous Chapter 6 where the complete dataset is available. Terminal ingredients when the trajectories leave the region of interest will also be incorporated into the methodology in order to avoid the infinite-time problem would be ill-defined unless the region of interest is invariant. In this case, the regions of interest are formed by the zones of influence of the active regressors that depend on previous exploration data.

7.3 Problem setup

Consider a deterministic discrete nonlinear dynamic system and its cost V_π, that are defined in the equations (7.1) and (7.2), respectively.

$$x_+ = f(x, u) \tag{7.1}$$

$$V_\pi(x_0) := \sum_{k=0}^{\infty} \gamma^k L(x_k, \pi(x_k)) \tag{7.2}$$

Being $x \in \mathbb{X}$ the state space, $u \in \mathbb{U}$ is the input and $x_+ \in \mathbb{X}$ denotes the successor state so that $f : \mathbb{X} \times \mathbb{U} \mapsto \mathbb{X}$, and $\mathbb{U}$ will denote the set of valid control actions. A policy $\pi(x)$ computes the control action u; $\pi(x)$ is a function $\mathbb{X} \to \mathbb{U}$ that maps the state in closed-loop control actions of the system achieving dynamics $x_+ = f(x, \pi(x))$. In the equation (7.2), the $V_\pi : \mathbb{X} \mapsto \mathbb{R}$ is associated to a

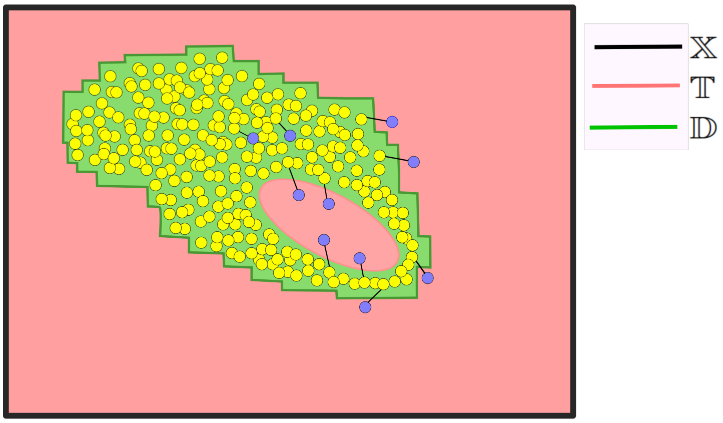


Figure 7.1: Illustration of the different sets involved in an optimal control problem ($\mathbb{X} = \mathbb{T} \cup \mathbb{D}$). The yellow circles in the data-availability region $\mathbb{D}$ indicate source states, the blue ones indicate successor ones.

policy $\pi(x)$ starting from an initial state x_0, where x_k is the state at time instant k, $L(x, u)$ is an immediate cost function $\mathbb{X} \times \mathbb{U} \to \mathbb{R}$, and it is assumed that $L(x, u) \geq 0$ for all $(x, u) \in \mathbb{X} \times \mathbb{U}$. The range of discount factor is $0 < \gamma < 1$.

It is assumed that there is a predefined controller for the exploration step. In the initial exploration and in the region where the learned-controlled is not defined the predefined controller is used in order to get a dataset $\mathcal{D}$, consisting of N triplets of data:

$$\mathcal{D} := \{(x_1, u_1, x_{1+}), (x_2, u_2, x_{2+}), \ldots, (x_N, u_N, x_{N+})\} \tag{7.3}$$

where $x_{k+} = f(x_k, u_k)$ and the set $\mathbb{D} \subseteq \mathbb{X}$ will be understood as a "region of interest" in the state space with data availability. The state-space region with no data availability is denoted as $\mathbb{T} := \mathbb{X} \sim \mathbb{D}$. The elements of a triplet $\xi := (x, u, x_+) \in \mathcal{D}$ will be denoted, 'source state', 'action ' and 'successor state', respectively. When speaking about x_+, it refers to x as its 'predecessor'. A symbolic representation of the relevant sets $\mathbb{X}$, $\mathbb{D}$ and $\mathbb{T}$, jointly with a few data points in $\mathcal{D}$ appear in Figure 7.1.

In the case that a theoretical model of (7.1) is available, the data set can be obtained from the simulation; otherwise, the data can be obtained by experimental data acquisition. In this Chapter, it assumes the dataset is not fixed, and the exploration-exploitation is applied to solve the ADP optimisation problem.

The posing of the problem should consider issues of regularisation and well posed to avoid unbounded or ill-conditioned solutions and rank deficiency. These problems are presented by a wrong selection of regressors or data points.

Other issues should also be taken into consideration such as unused data points, problematic regressors due to poor activation of successor states, flat regressors, low input excitation and linearly dependent regressors. These problems are solved with the appropriate establishment of regressor activation thresholds, as well as an appropriate selection of the type of regressor to be used, for more details, see Section 6.4.

Our proposal is:

1. explore the state space using learned/predefined controller in order to get dataset for applying ADP-LP approach.
2. solve the lower-bound ADP-LP setup in Section 7.3, obtaining a parameter vector θ_{LB},
3. compute the policy π form the lower-bound value function estimate.
4. Iterate 1–3 until a suitable stopping criteria is met.

7.4 Simulation example

Consider a nonlinear second-order discrete-time model:

$$x_{t+1} = \begin{bmatrix} \frac{15}{x_1}\sin\frac{x_1}{15} & 0.0049 \\ 0 & 0.9540 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0.0021 \\ 0.8505 \end{bmatrix} u \tag{7.4}$$

The state space is defined for the position α in the range $[-\pi, \pi]$rad, the angular velocity w in the range $[-16\pi, 16\pi]$rad/s, the control input range $[-5, 5]$V, and the sampling time is 0.005s.

The control problem will be stated using the quadratic cost with: $L(x, u) = x^T Q x + u^T R u$ with $Q = \begin{pmatrix} 5 & 0 \\ 0 & 0.01 \end{pmatrix}$ and $R = 0.01$. The discount factor will be set to $\gamma = 0.99$. Additionally, we define two terminal regions $\mathbb{T} = \mathbb{T}_1 \cup \mathbb{T}_2$. The first region $\mathbb{T}_1$ has a cost of 1000 when the system abandons its defined operation region and the second region $\mathbb{T}_2$ defined by an inner ellipse has an LQR cost equal to 6.914. The data-availability (operating region) $\mathbb{D}$ is defined by the region of the state space in which the data has activated the regressors used to approximate the value function. The representation of the regions is shown in Figure 7.1.

7.4.1 Controller

Initially, the exploration is carried out with a PD controller. It was implemented to gather the dataset at the beginning of the simulation, but also it is a backup controller in regions where the learning-controller is not defined. The control action of the PD controller is described in the equation (7.5).

$$u_{PD} = -K_p(\alpha - \alpha_{ref}) - K_d w \tag{7.5}$$

We introduced a probing noise signal in the PD control action in each time step in order to get a PD exploration input $u_{expl_{PD}}$; it is shown in the equation (7.6).

$$u_{expl_{PD}} = u_{PD}(1+\rho) + \rho \cdot \text{Input}_{max}, \quad \rho = rd \cdot K_{noise}, \tag{7.6}$$

for $K_p = 50$, $K_d = 0.5$, $K_{noise} = 0.70$ and rd is uniformly distributed random number ($rd \sim U(-\frac{1}{2}, \frac{1}{2})$). Note that the final control action has multiplicative and additive noise. The Figure 7.2 represents the iteration 1 with a initial PD controller. In the exploration step, a noise signal is used in both the control signal and initial point in each trajectory. The learned-controller is obtained using the equation (7.9).

7.4.2 Exploration

In each iteration, an exploration step is done. The dataset consists of 10 trajectories with 50 samples each one. The initial point x_0 is $[-1.9780; 30.9571]$ and the final reference point is $[0; 0]$. In each trajectory, the initial point is computed with a normal distribution $\mathcal{N}(x_0, \sigma^2)$ where $\sigma = [0.25; 1.5]$. The control action to explore is defined like:

$$u_{expl}(x) = \begin{cases} u_{expl_{IADP}} & x \in \mathbb{D} \\ u_{expl_{PD}} & x \in \mathbb{T} \end{cases} \tag{7.7}$$

$$u_{expl_{IADP}} = u_{IADP}(1+\rho) + \rho \cdot \text{Input}_{max} \tag{7.8}$$

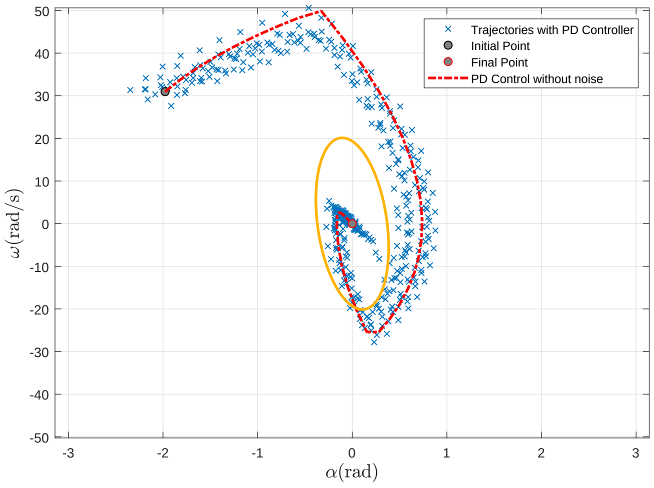


Figure 7.2: Initial dataset obtained using a PD exploration controller. The initial exploration is performed by adding probing noise to the PD controller in order to generate a dataset composed of several trajectories from the same initial point.

$$u_{IADP} = \pi_k(x) = \arg\min_{u \in \mathbb{U}} \left(L(x,u) + \gamma V(f(x,u), \theta_k) \right) \tag{7.9}$$

7.4.3 Discretised state and action spaces

The grid of the state space has a granularity of 55×55, and the input action is discretised in 11 points. In each iteration, the samples active in an incremental way the region where the regressors are excited. The region of influence of each regressor is activated if the samples exceed a threshold, and there is a minimum number of samples in its influence area.

7.4.4 Regressors configuration

The regressor to be tested is a two-dimensional 55×55 LUT (triangular membership functions). The minimum threshold value is 0.01, and at least two samples are required to activate the regressor. The samples in the terminal regions are not used because the regressors are not defined in these regions. The terminal cost is 1000 to samples that do not excite at least four regressors from the active neural region. This adjustment attempts a cautious exploration at the edges of

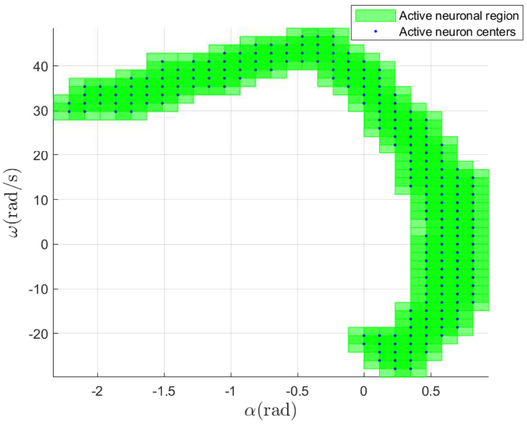
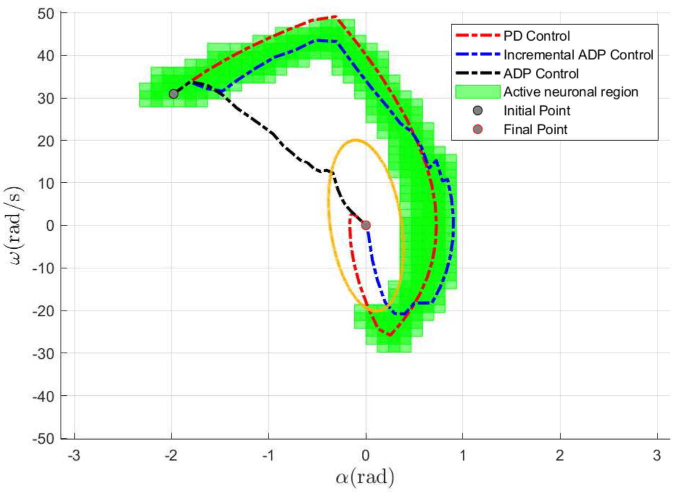


Figure 7.3: Active neural region with the initial dataset. The region is defined by the area of influence of each active neuron.

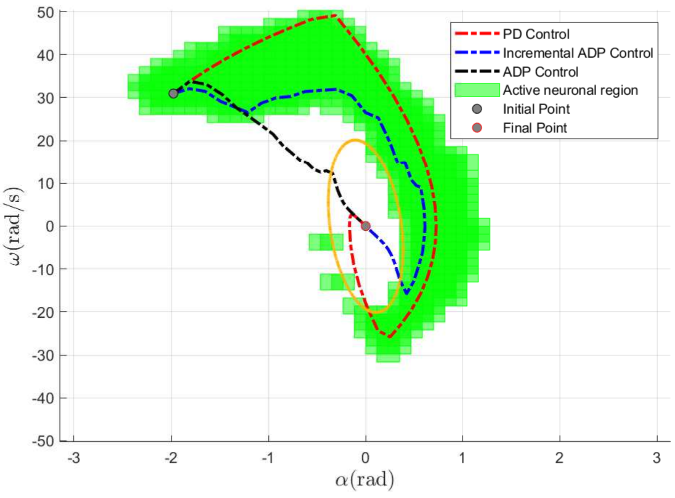
the active neuronal region. With these specifications, the only regions that are enough excited are shown in green in Figure 7.3.

The inequalities of the LP problem are formed using the dataset, the neuronal regions that have been activated and terminal regions. Once the LP is resolved, a new controller for the exploration in the next iteration is obtained. Note that, the new controller is only defined in the regions where the regressors are active. Therefore, exploration is performed with a combination of the learned controller and the initial PD controller.

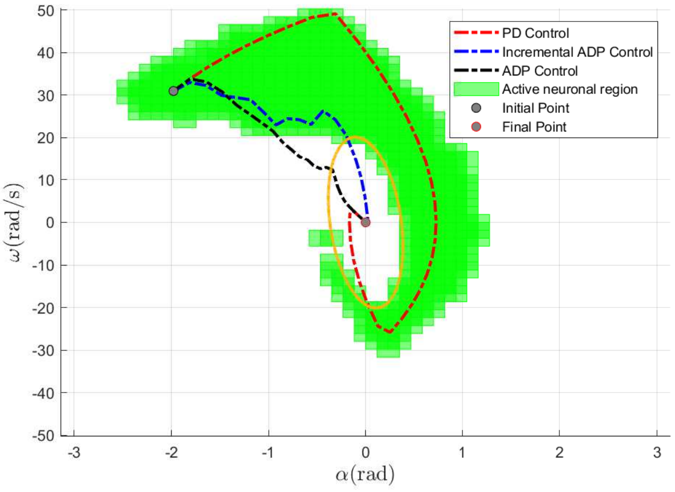
The Figure 7.4 shows some iterations in the learning process, and the comparison between the initial PD controller, incremental controller, and the ADP controller obtained if the LP problem had been solved with the whole dataset. Note that in each iteration, the dataset increases progressively with the data from each exploration.



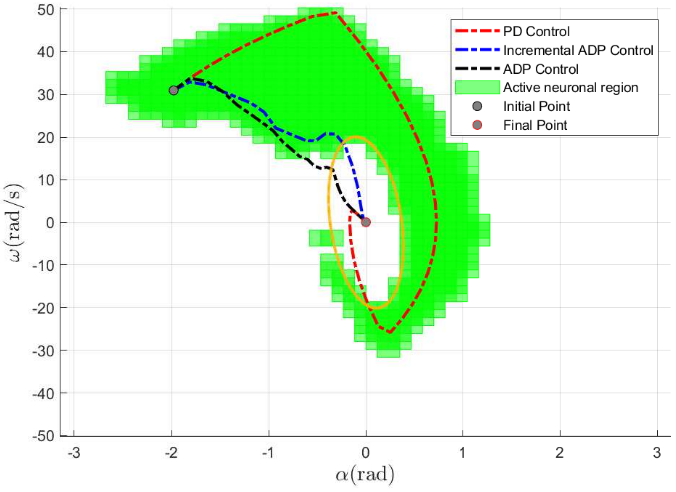
(a) Iteration 1



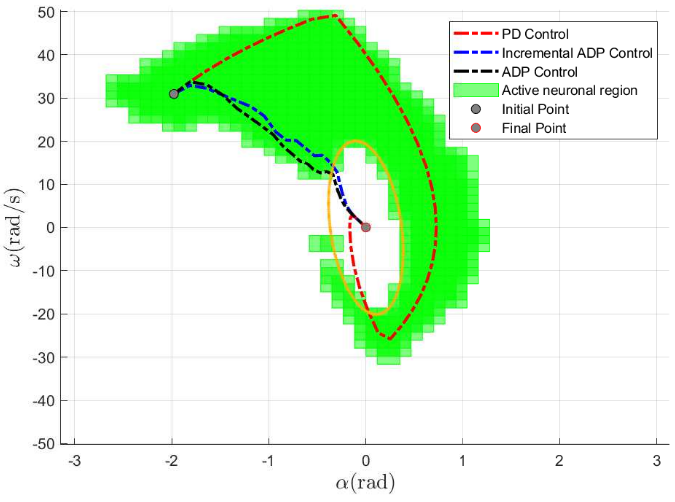
(b) Iteration 5



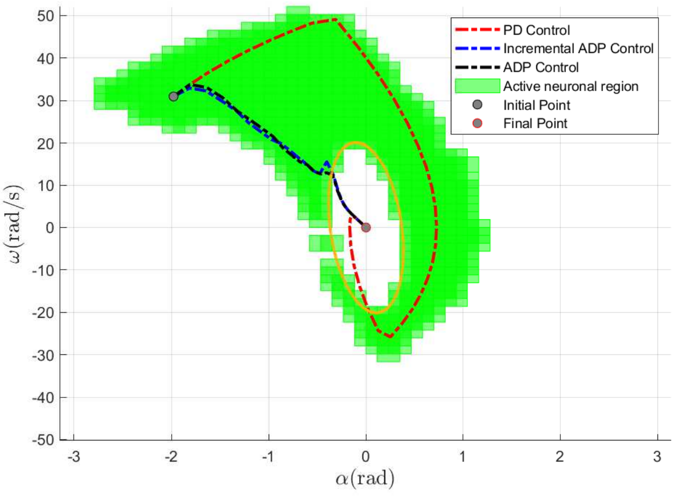
(c) Iteration 10



(d) Iteration 15



(e) Iteration 20



(f) Iteration 35

Figure 7.4: Iterations performed in the learning process, the neuronal active region increases in each iteration due to the combined exploration of the learned controller and the predefined (backup) controller.

7.5 Conclusions

This chapter has presented a methodology that includes learning suboptimal controllers using linear programming to approximate dynamic programming. The necessary conditions and requirements are the use of terminal regions and regressor regularisation to produce well-conditioned solutions. The result is an approximation of the lower bound of the value function in the region of specific interest in which there are exploration samples. Through exploration using an initial controller and then a learned controller, the learning region will increase progressively. The methodology has been validated by means of a simulation in a discrete system of which we know its optimal solution in order to compare them. With this approach what has been achieved is to obtain results very similar to those that would be achieved if the whole set of data of the system were available.

Chapter 8

Conclusions and future work

8.1 Conclusions

Dynamic programming and reinforcement learning are methods that solve sequential decision problems and provide an optimal solution to them. This type of problems appears in a wide range of applications in science and engineering, so its study and development have increased in recent years. Despite their success in several applications, these methods are limited to problems of small dimension and whose states and actions are discrete and can be stored in tables, in addition to another problem that usually requires a large amount of data to learn optimal policies. In this context, the motivation of this thesis is to develop methodologies that help solve these limitations. The following are the main conclusions drawn from the research carried out.

Chapter 2 is focused on the description of concepts that cover the systems with low dimensionality with finite number of states. Initially, the elements of the DP/RL problem are described by the Markov decision process. The most popular DP algorithms are policy iteration and value iteration. They solve the exact DP problems in an iterative way, and the convergence to the optimal solution is guaranteed. The main difference between DP and RL is the need of the model in order to compute the optimal solution; therefore, DP algorithms are model-based, and RL algorithms are model-free. The RL methods do not need the model, and these methods use the data obtained in the interaction/exploration of the controller on the system. In this context, the RL methods can be considered an extension of the applicability of the DP principles to problems where the model is not available.

Although RL does not use a model, the limitation to discrete finite problems of low dimensionality persists. In order to face problems it is necessary to use methods of approximation of functions, which is dealt with in Chapter 3. The approximation methods can be parametric and nonparametric. In parametric methods the number of adjustable parameters is fixed in the whole learning process. On the other hand, in the nonparametric methods, the number of adjustable parameters is variable and depend on the samples. The convergence properties to the optimal solution are lost when approximation functions are used. One way to efficiently manage data is through the use of batch algorithms; they use techniques of experience replay and supervised learning that improve the approach and add stability to the algorithms. Finally, the problems of dynamic programming and approximate dynamic programming can be raised as a problem of linear programming. Applying this approach, what is obtained is a lower bound of the optimal value function by relaxing the Bellman equation by inequality constraints.

Chapter 4 describes policy search (PS) methods, they find the optimal policy optimising directly over the parameter space. The optimisation over the parameter spaces allows solving RL problems in continuous spaces. The policy search methods can be model-free or model-based. Model-free methods are widely used as they calculate solutions using data only. In general, model-free PS methods have three steps that are repeated iteratively and these are exploration, the policy evaluation and policy update. The policy update stage is essential in this approach; the techniques for performing policy update are generally gradient or gradient-free. There are a variety of algorithms that have been successfully applied in many robotic and electromechanical systems. On the other hand, when the model is available, or the model can be obtained, PS techniques based on the model can be used. These techniques generally learn the system model, with this model, they simulate experiments, learn a new policy, and finally, this policy is used in the real system.

Chapter 5 presented a fuzzy fitted Q-function methodology to obtain approximately optimal controllers based on Takagi-Sugeno systems. The method first proposes to identify a fuzzy model based on collected data, incorporating partial model information via the memberships arising from sector-nonlinearity TS theoretical models. Second, a guaranteed cost controller based on LMIs can be designed for the identified system. Last, based on this controller, we can use it to initialise a fuzzy function approximator of the Q-function and apply Q-function fitting algorithms from the same dataset used in the identification phase. The Q-function fitting step can be carried out with standard policy/value iteration or, alternatively, if there are convergence problems, via generic (monotonic) optimisation algorithms. Examples illustrate that the methodology proposal (LMI initialisation plus monotonic optimisation) is advantageous with respect to standard PI/VI (which may have convergence issues) or BRM (which may converge to a local minima if not properly initialised). This methodology combines model-based optimal control (via identification) with data-based learning. We have shown that the proposed initialisation (guaranteed-cost controller for the identified TS system) can be considered a good starting condition for most fitted Q-function algorithms. The chapter presents results both in simulation and with real experimental data from an inverted pendulum.

Chapter 6 describes a methodology for learning optimal controllers based on data. It described a methodology based on earlier linear-programming approaches to approximate dynamic programming, this methodology includes terminal ingredients and regressor regularisation to produce well-conditioned and reasonably accurate solutions to optimal control problems. This solution does not have the convergence problems of other classic iterative algorithms. In addition, we have analysed the conditions in which this problem will provide adequate and well-conditioned solutions and, therefore, can be considered as a good approximation of the problem to be solved with a few parameters, in comparison with implementations based on interpolation of tables (with guaranteed convergence at the cost of a large number of adjustable parameters). The results are an approximate lower and upper bounds of the value function, so the gap between them can guide regressor choice. Although, in the simulation examples of this chapter are equispaced RBF neural networks or polynomial regressors have been used, in each specific problem another distribution of neurons or other types of regressors could be contemplated in order to try to obtain reasonable performances with a reduced number of parameters. Additionally, the validity of the approach has been experimentally tested in a inverted pendulum in order to learn the swing-up policy.

Chapter 7 presents an improved version of the methodology that use linear programming to approximate dynamic programming is presented. The necessary conditions and requirements are the use of terminal regions and regressor regularisation to produce well-conditioned solutions. The result is a lower bound of the value function in a specific region where there are exploration data. Through exploration using an initial controller and then a learned controller, the learning region is progressively increased. The methodology has been validated by means of a simulation in a discrete linear system of which we know its optimal solution in order to compare them. With this approach what has been achieved is to obtain results very similar to those that would be achieved if the whole set of data of the system were available.

8.2 Future work

The results obtained motivate us to expand our focus in several directions. The methodology proposed in Chapter 5 provides approximately an optimal controller based on TS modelling; this approach can be extended to applications that consider additional constraints. For example, the saturation of control actions can be included to obtain more realistic situations. Additionally, the methodology could be validated in an n-degree of freedom (DOF) system in which the identification and modelling would have to be computationally more efficient and mainly solve modelling uncertainties. Going one step further, the objective for future work would be a real-time implementation.

In Chapter 6 the idea of relaxing of Bellman's equation is used, changing equality for inequality, in this way a problem of linear programming is posed to calculate a lower and upper level of the optimal value function V, the approach for the improvement of this approach would be the use of Q value functions in combination with neural regressors or even deep neural networks. The same considerations can be considered for Chapter 7 to generalise their application and validate the methodologies proposed through their implementation in real platforms. In addition, combine this methodology with actor-critic architectures, especially with policy search algorithms that require the evaluation of the approximate value function.

Bibliography

Abbeel, Pieter, Morgan Quigley, and Andrew Y Ng (2006). “Using inaccurate models in reinforcement learning”. In: *Proceedings of the 23rd international conference on Machine learning.* ACM, pp. 1–8 (cit. on pp. 36, 42).

Albertos, Pedro and Antonio Sala (2006). *Multivariable control systems: an engineering approach.* London, U.K.: Springer (cit. on p. 11).

Albus, James S (1975). “A new approach to manipulator control: The cerebellar model articulation controller (CMAC)”. In: *Journal of Dynamic Systems, Measurement, and Control* 97.3, pp. 220–227 (cit. on p. 26).

Allgower, Frank and Alex Zheng (2012). *Nonlinear model predictive control.* ISBN: 978-3-0348-9554-5. DOI: 10.1007/978-3-0348-8407-5 (cit. on p. 88).

Alpaydin, Ethem (2014). *Introduction to machine learning.* MIT press (cit. on p. 26).

Amari, Shun-Ichi (1998). “Natural gradient works efficiently in learning”. In: *Neural computation* 10.2, pp. 251–276 (cit. on pp. 39, 40).

Antos, András, Csaba Szepesvári, and Rémi Munos (2008). “Learning near optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path”. In: *Machine Learning* 71.1, pp. 89–129 (cit. on pp. 56, 60, 71).

Ariño, Carlos, Emilio Pérez, and Antonio Sala (2010). “Guaranteed cost control analysis and iterative design for constrained Takagi–Sugeno systems”. In: *Engineering Applications of Artificial Intelligence* 23.8, pp. 1420–1427 (cit. on pp. 57, 89).

Ariño, Carlos, Andrés Querol, and Antonio Sala (2017). “Shape-independent model predictive control for Takagi–Sugeno fuzzy systems”. In: *Engineering Applications of Artificial Intelligence* 65, pp. 493–505 (cit. on p. 88).

Armesto, Leopoldo et al. (2015). “Duality-based nonlinear quadratic control: Application to mobile robot trajectory-following”. In: *IEEE Transactions on Control Systems Technology* 23.4, pp. 1494–1504 (cit. on pp. 57, 88).

Atkeson, Christopher G and Juan Carlos Santamaria (1997). “A comparison of direct and model-based reinforcement learning”. In: *Proceedings of International Conference on Robotics and Automation.* Vol. 4. IEEE, pp. 3557–3564 (cit. on p. 42).

Baird, Leemon C. and Andrew W. Moore (1999). “Gradient descent for general reinforcement learning”. In: *Advances in neural information processing systems*, pp. 968–974 (cit. on pp. 56, 89, 93).

Barto, A. G., R. S. Sutton, and C. W. Anderson (1983). “Neuronlike adaptive elements that can solve difficult learning control problems”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5, pp. 834–846. ISSN: 0018-9472 (cit. on p. 21).

Barto, Andrew Gehret, Steven J Bradtke, and Satinder P Singh (1991). *Real time learning and control using asynchronous dynamic programming.* University of Massachusetts at Amherst, Department of Computer (cit. on p. 18).

Baxter, Jonathan and Peter L Bartlett (2000). “Direct gradient-based reinforcement learning”. In: *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No. 00CH36353).* Vol. 3. IEEE, pp. 271–274 (cit. on p. 37).

— (2001). “Infinite-horizon policy-gradient estimation”. In: *Journal of Artificial Intelligence Research* 15, pp. 319–350 (cit. on p. 40).

Bellman, Richard (1957). *Dynamic Programming.* Princeton, USA: Princeton University Press (cit. on pp. 9, 10, 13, 15, 17, 59).

Bertsekas, D. P. and J. N. Tsitsiklis (1996). *Neuro-Dynamic Programming.* Belmont, MA, USA: Athena Scientific (cit. on pp. 12, 14, 15, 18, 19, 23–26, 59, 70).

Bertsekas, Dimitri P (2012). *Dynamic Programming and Optimal Control.* 4th edition. Vol. 2. Belmont, MA, USA: Athena Scientific (cit. on p. 10).

— (2017). *Dynamic Programming and Optimal Control.* 4th edition. Vol. 1. Belmont, MA, USA: Athena Scientific (cit. on pp. 10–12, 14, 56, 88, 91).

— (2019). *Reinforcement Learning and Optimal Control.* Athena scientific Belmont, MA (cit. on pp. 10, 18, 24–26).

Bethke, Brett, Jonathan P How, and Asuman Ozdaglar (2008). “Approximate dynamic programming using support vector regression”. In: *2008 47th IEEE Conference on Decision and Control.* IEEE, pp. 3811–3816 (cit. on p. 26).

Beuchat, P. N., A. Georghiou, and J. Lygeros (2020). “Performance Guarantees for Model-Based Approximate Dynamic Programming in Continuous Spaces”. In: *IEEE Transactions on Automatic Control* 65.1, pp. 143–158. ISSN: 2334-3303. DOI: `10.1109/TAC.2019.2906423` (cit. on pp. 32, 33).

Beuchat, Paul, Angelos Georghiou, and John Lygeros (2016). “Approximate dynamic programming: a Q-function approach”. In: *ArXiv e-prints* 1602 (cit. on pp. 32, 33).

Blackwell, David (1965). “Discounted dynamic programming”. In: *The Annals of Mathematical Statistics* 36.1, pp. 226–235 (cit. on pp. 88, 130).

Boyan, Justin A (2002). “Technical update: Least-squares temporal difference learning”. In: *Machine learning* 49.2-3, pp. 233–246 (cit. on p. 31).

Boyd, Stephen et al. (1994). *Linear matrix inequalities in system and control theory.* SIAM (cit. on p. 56).

Bradtke, Steven J and Andrew G Barto (1996). “Linear least-squares algorithms for temporal difference learning”. In: *Machine learning* 22.1-3, pp. 33–57 (cit. on p. 31).

Breiman, Leo (2017). *Classification and regression trees.* Routledge (cit. on p. 26).

Busoniu, Lucian et al. (2010). *Reinforcement learning and dynamic programming using function approximators.* Boca Raton, FL, USA: CRC press (cit. on pp. 9, 11, 12, 14, 15, 17–19, 24–28, 36, 56, 60, 61, 88–90, 94, 105, 109, 110, 112, 114, 119, 129, 130).

Busoniu, Lucian et al. (2018). “Reinforcement learning for control: Performance, stability, and deep approximators”. In: *Annual Reviews in Control* 46, pp. 8 –28. ISSN: 1367-5788. DOI: `https://doi.org/10.1016/j.arcontrol.2018.09.005` (cit. on pp. 13, 18, 19, 25, 28, 32, 36).

Camacho, Eduardo F. and Carlos Bordons (2013). *Model predictive control.* London, U.K.: Springer (cit. on pp. 88, 97).

Chong, Edwin KP and Stanislaw H Zak (2013). *An introduction to optimization.* Vol. 76. John Wiley & Sons (cit. on p. 71).

Cleveland, William S and Susan J Devlin (1988). “Locally weighted regression: an approach to regression analysis by local fitting”. In: *Journal of the American statistical association* 83.403, pp. 596–610 (cit. on p. 43).

Daniel, Christian, Gerhard Neumann, and Jan R Peters (2012). “Hierarchical relative entropy policy search”. In: *International Conference on Artificial Intelligence and Statistics*, pp. 273–281 (cit. on p. 41).

Davies, Scott (1997). “Multidimensional triangulation and interpolation for reinforcement learning”. In: *Advances in neural information processing systems*, pp. 1005–1011 (cit. on p. 26).

Dayan, Peter and Geoffrey E. Hinton (1997). “Using Expectation-Maximization for Reinforcement Learning”. In: *Neural Computation* 9.2, pp. 271–278. DOI: `10.1162/neco.1997.9.2.271`. eprint: `https://doi.org/10.1162/neco.1997.9.2.271` (cit. on p. 41).

De Asis, Kristopher et al. (2018). “Multi-step reinforcement learning: A unifying algorithm”. In: *Thirty-Second AAAI Conference on Artificial Intelligence* (cit. on p. 20).

De Farias, Daniela Pucci and Benjamin Van Roy (2003). “The linear programming approach to approximate dynamic programming”. In: *Operations research* 51.6, pp. 850–865 (cit. on pp. 24, 32, 33, 89, 94, 95, 106, 107, 110).

— (2004). “On constraint sampling in the linear programming approach to approximate dynamic programming”. In: *Mathematics of operations research* 29.3, pp. 462–478 (cit. on p. 89).

De Paula, Mariano et al. (2015). “Multimodal Control in Uncertain Environments using Reinforcement Learning and Gaussian Processes”. In: *Rev. Ib. Automatica Informatica Industrial* 12.4, pp. 385–396 (cit. on p. 70).

Deisenroth, Marc and Carl E Rasmussen (2011). “PILCO: A model-based and data-efficient approach to policy search”. In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472 (cit. on pp. 36, 43).

Deisenroth, Marc Peter, Gerhard Neumann, Jan Peters, et al. (2013). “A survey on policy search for robotics”. In: *Foundations and Trends in Robotics* 2.1–2, pp. 1–142 (cit. on pp. 15, 24, 36–43, 56).

Deisenroth, Marc Peter, Carl Edward Rasmussen, and Dieter Fox (2011). “Learning to control a low-cost manipulator using data-efficient reinforcement learning”. In: (cit. on p. 43).

Díaz, H., L. Armesto, and A. Sala (2020). “Fitted Q-Function Control Methodology Based on Takagi-Sugeno Systems”. In: *IEEE Transactions on Control Systems Technology* 28.2, pp. 477–488. ISSN: 2374-0159. DOI: `10.1109/TCST.2018.2885689` (cit. on pp. 94, 106).

Díaz, H., A. Sala, and L. Armesto (2017). “Improving LMI controllers for discrete nonlinear systems using policy iteration”. In: *2017 21st International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 833–838. DOI: `10.1109/ICSTCC.2017.8107140` (cit. on pp. 57, 69).

Díaz, Henry, Leopoldo Armesto, and Antonio Sala (2016). “Improvement of LMI controllers of Takagi-Sugeno models via Q-learning”. In: *IFAC PapersOnLine* 49.5. 4th IFAC Conference on Intelligent Control and Automation Sciences ICONS, pp. 67 –72. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2016.07.091 (cit. on pp. 57, 78).

— (2019). “Metodología de programación dinámica aproximada para control óptimo basada en datos”. In: *Revista Iberoamericana de Automática e Informática industrial* 16.3, pp. 273–283. ISSN: 1697-7920. DOI: 10.4995/riai.2019.10379 (cit. on p. 95).

Doya, Kenji, Hidenori Kimura, and Mitsuo Kawato (2001). “Neural mechanisms of learning and control”. In: *Control Systems, IEEE* 21.4, pp. 42–54 (cit. on p. 10).

Dreyfus, Stuart E and Richard Bellman (1962). *Applied dynamic programming.* Princeton University Press (cit. on p. 10).

Drucker, Harris et al. (1997). “Support Vector Regression Machines”. In: *Advances in Neural Information Processing Systems 9.* Ed. by M. I. Jordan and T. Petsche. MIT Press, pp. 155–161 (cit. on p. 46).

Endo, Gen et al. (2008). “Learning CPG-based biped locomotion with a policy gradient method: Application to a humanoid robot”. In: *The International Journal of Robotics Research* 27.2, pp. 213–228 (cit. on p. 36).

Engel, Yaakov, Shie Mannor, and Ron Meir (2005). “Reinforcement learning with Gaussian processes”. In: *Proceedings of the 22nd international conference on Machine learning.* ACM, pp. 201–208 (cit. on p. 26).

Ernst, Damien, Pierre Geurts, and Louis Wehenkel (2005). “Tree-based batch mode reinforcement learning”. In: *Journal of Machine Learning Research* 6.Apr, pp. 503–556 (cit. on pp. 26, 31).

Fairbank, M. and E. Alonso (2012). “The divergence of reinforcement learning algorithms with value-iteration and function approximation”. In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2012.6252792 (cit. on pp. 56, 60, 75).

Falsone, Alessandro and Maria Prandini (2015). “An iterative scheme for the approximate linear programming solution to the optimal control of a Markov Decision Process”. In: *Control Conference (ECC), 2015 European*. IEEE, pp. 1200–1205 (cit. on p. 89).

Farahmand, A. m. et al. (2009). “Regularized Fitted Q-Iteration for planning in continuous-space Markovian decision problems”. In: *2009 American Control Conference*, pp. 725–730. DOI: `10.1109/ACC.2009.5160611` (cit. on p. 26).

Fernández, Fernando and Daniel Borrajo (1999). “VQQL. Applying vector quantization to reinforcement learning”. In: *Robot Soccer World Cup*. Springer, pp. 292–303 (cit. on p. 29).

Fujimoto, Scott, Herke van Hoof, and David Meger (2018). “Addressing function approximation error in actor-critic methods”. In: *arXiv:1802.09477* (cit. on p. 42).

Gabillon, Victor, Mohammad Ghavamzadeh, and Bruno Scherrer (2013). “Approximate dynamic programming finally performs well in the game of Tetris”. In: *Advances in neural information processing systems*, pp. 1754–1762 (cit. on p. 41).

Geist, Matthieu and Olivier Pietquin (2013). “Algorithmic survey of parametric value function approximation”. In: *IEEE Transactions on Neural Networks and Learning Systems* 24.6, pp. 845–867 (cit. on pp. 28, 56).

Glorennec, Pierre Yves (2000). “Reinforcement learning: An overview”. In: *Proceedings European Symposium on Intelligent Techniques (ESIT-00), Aachen, Germany*. Citeseer, pp. 14–15 (cit. on p. 29).

Goodwin, Graham, María M Seron, and José A De Doná (2006). *Constrained control and estimation: an optimisation approach*. Springer (cit. on p. 98).

Gordon, Geoffrey J (1995). “Stable function approximation in dynamic programming”. In: *Machine Learning Proceedings 1995*. Elsevier, pp. 261–268 (cit. on p. 31).

Greensmith, Evan, Peter L Bartlett, and Jonathan Baxter (2004). “Variance reduction techniques for gradient estimates in reinforcement learning”. In: *Journal of Machine Learning Research* 5.Nov, pp. 1471–1530 (cit. on p. 39).

Guerra, Thierry M, Antonio Sala, and Kazuo Tanaka (2015). “Fuzzy control turns 50: 10 years later”. In: *Fuzzy sets and systems* 281, pp. 168–182 (cit. on p. 57).

Haarnoja, Tuomas et al. (2018). “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *arXiv:1801.01290* (cit. on p. 42).

Han, Ke-Zhen, Jian Feng, and Xiaohong Cui (2017). “Fault-tolerant optimised tracking control for unknown discrete-time linear systems using a combined reinforcement learning and residual compensation methodology”. In: *International Journal of Systems Science* 48.13, pp. 2811–2825 (cit. on p. 89).

Hansen, N. and A. Ostermeier (1996). “Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation”. In: *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 312–317. DOI: `10.1109/ICEC.1996.542381` (cit. on p. 41).

Hansen, Nikolaus, Sibylle D. Muller, and Petros Koumoutsakos (2003). “Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)”. In: *Evolutionary Computation* 11.1, pp. 1–18. DOI: `10.1162/106365603321828970`. eprint: `https://doi.org/10.1162/106365603321828970` (cit. on p. 41).

Hardin, James W. and Joseph Hilbe (2001). *Generalized Linear Models and Extensions.* College Station, Texas: Stata Press, p. 245 (cit. on p. 46).

Haykin, Simon (1998). *Neural Networks: A Comprehensive Foundation.* 2nd. Upper Saddle River, NJ, USA: Prentice Hall PTR. ISBN: 0132733501 (cit. on p. 46).

Heidrich Meisner, Verena and Christian Igel (2009). “Neuroevolution strategies for episodic reinforcement learning”. In: *Journal of Algorithms* 64.4, pp. 152–168 (cit. on pp. 36, 41).

Hellendoorn, Hans and Dimiter Driankov (1997). *Fuzzy model identification: selected approaches.* Springer (cit. on p. 64).

Hessel, Matteo et al. (2018). “Rainbow: Combining improvements in deep reinforcement learning”. In: *Thirty-Second AAAI Conference on Artificial Intelligence* (cit. on p. 32).

Horiuchi, T et al. (1996). “Fuzzy interpolation-based Q-learning with continuous states and actions”. In: *Proceedings of IEEE 5th International Fuzzy Systems.* Vol. 1. IEEE, pp. 594–600 (cit. on p. 29).

Howard, Ronald A (1960). *Dynamic programming and Markov processes.* Cambriage, MA: MIT Press (cit. on pp. 15, 16).

Jaakkola, Tommi, Michael I Jordan, and Satinder P Singh (1994). “Convergence of stochastic iterative dynamic programming algorithms”. In: *Advances in neural information processing systems*, pp. 703–710 (cit. on p. 19).

Jie, Tang and Pieter Abbeel (2010). “On a connection between importance sampling and the likelihood ratio policy gradient”. In: *Advances in Neural Information Processing Systems*, pp. 1000–1008 (cit. on pp. 40, 44, 50).

Jodogne, Sébastien, Cyril Briquet, and Justus H Piater (2006). “Approximate policy iteration for closed-loop learning of visual tasks”. In: *European Conference on Machine Learning.* Springer, pp. 210–221 (cit. on p. 26).

Jouffe, L. (1998). “Fuzzy inference system learning by reinforcement methods”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 28.3, pp. 338–355. ISSN: 1094-6977. DOI: 10.1109/5326.704563 (cit. on p. 29).

Kaelbling, Leslie Pack, Michael L Littman, and Andrew W Moore (1996). “Reinforcement learning: A survey”. In: *Journal of artificial intelligence research*, pp. 237–285 (cit. on p. 14).

Kappen, Hilbert J, Wim Wiegerinck, and B van den Broek (2007). “A path integral approach to agent planning”. In: *Autonomous Agents and Multi-Agent Systems* (cit. on p. 41).

Kim, H Jin et al. (2004). “Autonomous helicopter flight via reinforcement learning”. In: *Advances in neural information processing systems*, pp. 799–806 (cit. on p. 41).

Kiumarsi, Bahare et al. (2014). “Reinforcement Q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics”. In: *Automatica* 50.4, pp. 1167–1175 (cit. on p. 18).

Kiumarsi, Bahare et al. (2018). “Optimal and autonomous control using reinforcement learning: A survey”. In: *IEEE transactions on neural networks and learning systems* 29.6, pp. 2042–2062 (cit. on p. 89).

Kober, Jens, J Andrew Bagnell, and Jan Peters (2013). “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11, pp. 1238–1274 (cit. on p. 36).

Kober, Jens and Jan Peters (2016). *LEARNING MOTOR SKILLS.* Springer (cit. on p. 36).

Kober, Jens and Jan R Peters (2009). “Policy search for motor primitives in robotics”. In: *Advances in neural information processing systems*, pp. 849–856 (cit. on pp. 37, 41).

Kober, Jens et al. (2010). “Movement templates for learning of hitting and batting”. In: *2010 IEEE International Conference on Robotics and Automation.* IEEE, pp. 853–858 (cit. on p. 36).

Konda, Vijay R and John N Tsitsiklis (2000). “Actor critic algorithms”. In: *Advances in neural information processing systems*, pp. 1008–1014 (cit. on p. 21).

Kong, Augustine (1992). *A Note on Importance Sampling using Standardized Weights.* Tech. rep. Department of Statistics, University of Chicago (cit. on p. 45).

Kretchmar, R Matthew and Charles W Anderson (1997). “Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning”. In: *Neural Networks, 1997., International Conference on.* Vol. 2. IEEE, pp. 834–837 (cit. on p. 114).

Lagarias, Jeffrey C et al. (1998). “Convergence properties of the Nelder–Mead simplex method in low dimensions”. In: *SIAM Journal on optimization* 9.1, pp. 112–147 (cit. on pp. 56, 81, 89).

Lagoudakis, Michail G and Ronald Parr (2002). “Model-free least-squares policy iteration”. In: *Advances in neural information processing systems*, pp. 1547–1554 (cit. on p. 31).

— (2003). “Least-squares policy iteration”. In: *Journal of machine learning research* 4.Dec, pp. 1107–1149 (cit. on pp. 31, 92, 93).

Lange, Sascha, Thomas Gabel, and Martin Riedmiller (2012). “Batch Reinforcement Learning”. In: *Reinforcement Learning: State-of-the-Art*. Ed. by Marco Wiering and Martijn van Otterlo. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 45–73. ISBN: 978-3-642-27645-3. DOI: `10.1007/978-3-642-27645-3_2` (cit. on p. 31).

Lange, Sascha and Martin Riedmiller (2010). “Deep auto-encoder neural networks in reinforcement learning”. In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8 (cit. on p. 31).

Latombe, Jean-Claude (2012). *Robot motion planning*. Vol. 124. Springer (cit. on p. 120).

Lewis, Frank, Draguna Vrabie, and Vassilis Syrmos (2012). *Optimal control*. Third edition. Hoboken, NJ: John Wiley & Sons (cit. on p. 27).

Lewis, Frank L and Derong Liu (2013). *Reinforcement learning and approximate dynamic programming for feedback control*. Hoboken, NJ, USA: Wiley (cit. on pp. 9, 36, 56, 88, 89).

Lewis, Frank L and Draguna Vrabie (2009). “Reinforcement learning and adaptive dynamic programming for feedback control”. In: *Circuits and Systems Magazine, IEEE* 9.3, pp. 32–50 (cit. on pp. 10, 15, 18, 36, 56, 91, 93, 110).

Lewis, Frank L, Draguna Vrabie, and Kyriakos G Vamvoudakis (2012). “Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers”. In: *Control Systems, IEEE* 32.6, pp. 76–105 (cit. on pp. 56, 59, 88, 89).

Lillicrap, Timothy P et al. (2015). “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (cit. on p. 42).

Lin, Long-Ji (1992). “Self-improving reactive agents based on reinforcement learning, planning and teaching”. In: *Machine learning* 8.3-4, pp. 293–321 (cit. on p. 31).

Lincoln, B. and A. Rantzer (2006). “Relaxing dynamic programming”. In: *IEEE Transactions on Automatic Control* 51.8, pp. 1249–1260. ISSN: 2334-3303. DOI: `10.1109/TAC.2006.878720` (cit. on p. 106).

Liu, Derong et al. (2017). *Adaptive dynamic programming with applications in optimal control*. Berlin, Germany: Springer (cit. on pp. 27, 59, 60, 88).

Liu, Dong and Guang-Hong Yang (2018). “Model-free adaptive control design for nonlinear discrete-time processes with reinforcement learning techniques”. In: *International Journal of Systems Science* 49.11, pp. 2298–2308 (cit. on p. 88).

Ljung, Lennart (2001). “System identification”. In: *Wiley Encyclopedia of Electrical and Electronics Engineering* (cit. on p. 14).

Lovera, Marco et al. (2011). “Guest editorial special issue on applied LPV modeling and identification”. In: *IEEE Transactions on Control Systems Technology* 19.1, pp. 1–4 (cit. on p. 64).

Lughofer, Edwin (2011). *Evolving fuzzy systems-methodologies, advanced concepts and applications*. Vol. 53. Springer (cit. on pp. 64, 71).

Maei, Hamid Reza (2011). “Gradient temporal-difference learning algorithms”. In: (cit. on p. 30).

Maei, Hamid Reza et al. (2010). “Toward off-policy learning control with function approximation.” In: *ICML*, pp. 719–726 (cit. on p. 30).

Manne, Alan S (1960). “Linear programming and sequential decisions”. In: *Management Science* 6.3, pp. 259–267 (cit. on pp. 32, 89, 94).

Márquez, Raymundo et al. (2013). “Improvements on non-quadratic stabilization of Takagi-Sugeno models via line-integral Lyapunov functions”. In: *IFAC Proceedings Volumes* 46.20, pp. 473–478 (cit. on p. 65).

Melo, Francisco S, Sean P Meyn, and M Isabel Ribeiro (2008). “An analysis of reinforcement learning with function approximation”. In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 664–671 (cit. on p. 30).

Mnih, Volodymyr et al. (2015). “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540, p. 529 (cit. on pp. 26, 32).

Mnih, Volodymyr et al. (2016). “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*, pp. 1928–1937 (cit. on p. 42).

Munos, R., L. C. Baird, and A. W. Moore (1999). “Gradient descent approaches to neural-net-based solutions of the Hamilton-Jacobi-Bellman equation”. In: *IJCNN’99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*. Vol. 3, 2152–2157 vol.3. DOI: `10.1109/IJCNN.1999.832721` (cit. on pp. 56, 89).

Munos, Rémi and Andrew Moore (2002). “Variable resolution discretization in optimal control”. In: *Machine learning* 49.2-3, pp. 291–323 (cit. on pp. 26, 27).

Munos, Rémi and Csaba Szepesvári (2008). “Finite-time bounds for fitted value iteration”. In: *Journal of Machine Learning Research* 9.May, pp. 815–857 (cit. on pp. 60, 93).

Murphy, Susan A (2005). “A generalization error for Q-learning”. In: *Journal of Machine Learning Research* 6.Jul, pp. 1073–1097 (cit. on p. 29).

Nelder, John A and Roger Mead (1965). “A simplex method for function minimization”. In: *The computer journal* 7.4, pp. 308–313 (cit. on p. 43).

Ng, Andrew Y and Michael Jordan (2000). “PEGASUS: A policy search method for large MDPs and POMDPs”. In: *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., pp. 406–415 (cit. on p. 43).

Niv, Yael (2009). “Reinforcement learning in the brain”. In: *Journal of Mathematical Psychology* 53.3, pp. 139–154 (cit. on p. 10).

Ormoneit, Dirk and Śaunak Sen (2002). “Kernel-based reinforcement learning”. In: *Machine learning* 49.2-3, pp. 161–178 (cit. on pp. 26, 31).

Park, Jooyoung and Irwin W Sandberg (1991). "Universal approximation using radial-basis-function networks". In: *Neural computation* 3.2, pp. 246–257 (cit. on p. 103).

Pastor, J. et al. (2018). "Learning Upper-Level Policy using Importance Sampling-based Policy Search Method". In: *2018 7th International Conference on Systems and Control (ICSC)*, pp. 188–193. DOI: 10.1109/ICoSC.2018.8587772 (cit. on pp. 36, 40, 43).

Pavlov, IP (1927). "Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex." In: (cit. on p. 10).

Pelikan, Martin, David E Goldberg, and Fernando G Lobo (2002). "A survey of optimization by building and using probabilistic models". In: *Computational optimization and applications* 21.1, pp. 5–20 (cit. on p. 71).

Peters, Jan, Katharina Mulling, and Yasemin Altun (2010). "Relative entropy policy search". In: *Twenty-Fourth AAAI Conference on Artificial Intelligence* (cit. on p. 41).

Peters, Jan and Stefan Schaal (2006). "Policy gradient methods for robotics". In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, pp. 2219–2225 (cit. on pp. 37, 38).

— (2007). "Reinforcement learning by reward weighted regression for operational space control". In: *Proceedings of the 24th international conference on Machine learning*. ACM, pp. 745–750 (cit. on p. 41).

— (2008a). "Natural actor-critic". In: *Neurocomputing* 71.7, pp. 1180–1190 (cit. on p. 40).

— (2008b). "Reinforcement learning of motor skills with policy gradients". In: *Neural networks* 21.4, pp. 682–697 (cit. on p. 40).

Peters, Jan, Sethu Vijayakumar, and Stefan Schaal (2003). "Reinforcement learning for humanoid robotics". In: *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pp. 1–20 (cit. on p. 37).

Pomprapa, Anake, Steffen Leonhardt, and Berno JE Misgeld (2017). "Optimal learning control of oxygen saturation using a policy iteration algorithm and a proof-of-concept in an interconnecting three-tank system". In: *Control Engineering Practice* 59, pp. 194–203 (cit. on p. 56).

Postoyan, R. et al. (2017). "Stability Analysis of Discrete-Time Infinite-Horizon Optimal Control With Discounted Cost". In: *IEEE Transactions on Automatic Control* 62.6, pp. 2736–2749. ISSN: 0018-9286. DOI: `10.1109/TAC.2016.2616644` (cit. on p. 13).

Powell, Warren B (2011). *Approximate Dynamic Programming: Solving the curses of dimensionality.* Second edition. Hoboken, NJ, USA: Wiley (cit. on pp. 10, 11, 14, 15, 25, 28, 56, 59, 89).

— (2019). "A unified framework for stochastic optimization". In: *European Journal of Operational Research* 275.3, pp. 795–821 (cit. on p. 9).

Puterman, Martin L (2014). *Markov decision processes: discrete stochastic dynamic programming.* New York, NY, USA: John Wiley & Sons (cit. on p. 11).

Radac, Mircea-Bogdan, Radu-Emil Precup, and Raul-Cristian Roman (2017). "Model-Free control performance improvement using virtual reference feedback tuning and reinforcement Q-learning". In: *International Journal of Systems Science* 48.5, pp. 1071–1083. DOI: `10.1080/00207721.2016.1236423` (cit. on pp. 57, 88).

Rantzer, Anders (2006). "Relaxed dynamic programming in switching systems". In: *IEE Proceedings-Control Theory and Applications* 153.5, pp. 567–574 (cit. on p. 106).

Rasmussen, Carl Edward (2003). "Gaussian processes in machine learning". In: *Summer School on Machine Learning.* Springer, pp. 63–71 (cit. on pp. 26, 43).

Rasmussen, Carl Edward and Christopher K. I. Williams (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press. ISBN: 026218253X (cit. on p. 46).

Riedmiller, Martin (2005). "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *European Conference on Machine Learning.* Springer, pp. 317–328 (cit. on pp. 26, 31, 60).

Riedmiller, Martin, Mike Montemerlo, and Hendrik Dahlkamp (2007). “Learning to drive a real car in 20 minutes”. In: *2007 Frontiers in the Convergence of Bioscience and Information Technologies*. IEEE, pp. 645–650 (cit. on p. 31).

Rohmer, Eric, Surya PN Singh, and Marc Freese (2013). “V-REP: A versatile and scalable robot simulation framework”. In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, pp. 1321–1326 (cit. on p. 120).

Rubinstein, Reuven Y and Dirk P Kroese (2013). *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media (cit. on p. 41).

Rückstieß, Thomas, Martin Felder, and Jürgen Schmidhuber (2008). “State dependent exploration for policy gradient methods”. In: *Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 234–249 (cit. on p. 37).

Rummery, Gavin A and Mahesan Niranjan (1994). *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, England (cit. on p. 20).

Sala, Antonio (2009). “On the conservativeness of fuzzy and fuzzy-polynomial control of nonlinear systems”. In: *Annual Reviews in Control* 33.1, pp. 48–58 (cit. on pp. 63, 68).

Schaal, Stefan et al. (2005). “Learning movement primitives”. In: *Robotics Research. The Eleventh International Symposium*. Springer, pp. 561–572 (cit. on p. 36).

Schaul, Tom et al. (2015). “Prioritized experience replay”. In: *arXiv preprint arXiv:1511.05952* (cit. on p. 32).

Schneider, Jeff G (1997). “Exploiting model uncertainty estimates for safe dynamic control learning”. In: *Advances in neural information processing systems*, pp. 1047–1053 (cit. on p. 43).

Schölkopf, Bernhard, Christopher JC Burges, Alexander J Smola, et al. (1999). *Advances in kernel methods: support vector learning*. MIT press (cit. on p. 26).

Schulman, John et al. (2015a). “High-dimensional continuous control using generalized advantage estimation”. In: *arXiv preprint arXiv:1506.02438* (cit. on p. 42).

Schulman, John et al. (2015b). “Trust region policy optimization”. In: *International conference on machine learning*, pp. 1889–1897 (cit. on p. 40).

Schulman, John et al. (2017). “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (cit. on p. 40).

Sehnke, Frank et al. (2010). “2010 Special Issue: Parameter-exploring Policy Gradients”. In: *Neural Netw.* 23.4, pp. 551–559. ISSN: 0893-6080. DOI: `10.1016/j.neunet.2009.12.004` (cit. on p. 39).

Seijen, Harm van (2016). *Effective Multi-step Temporal-Difference Learning for Non-Linear Function Approximation.* arXiv: `1608.05151 [cs.AI]` (cit. on p. 20).

Shawe-Taylor, John, Nello Cristianini, et al. (2004). *Kernel methods for pattern analysis.* Cambridge university press (cit. on p. 26).

Sherstov, Alexander A and Peter Stone (2005). “Function approximation via tile coding: Automating parameter choice”. In: *International Symposium on Abstraction, Reformulation, and Approximation.* Springer, pp. 194–205 (cit. on p. 29).

Sigaud, Olivier and Olivier Buffet (2013). *Markov decision processes in artificial intelligence.* John Wiley & Sons (cit. on p. 11).

Sigaud, Olivier and Freek Stulp (2019). “Policy search in continuous action domains: an overview”. In: *Neural Networks* (cit. on pp. 15, 24, 36).

Singh, Satinder et al. (2000). “Convergence results for single-step on-policy reinforcement learning algorithms”. In: *Machine learning* 38.3, pp. 287–308 (cit. on pp. 19, 20).

Singh, Satinder P, Tommi Jaakkola, and Michael I Jordan (1995). “Reinforcement learning with soft state aggregation”. In: *Advances in neural information processing systems*, pp. 361–368 (cit. on p. 26).

Skinner, Burrhus Frederic (1938). *The behavior of organisms: An experimental analysis.* New York, NY: Appleton-Century (cit. on p. 10).

Škrjanc, Igor (2015). "Evolving fuzzy-model-based design of experiments with supervised hierarchical clustering". In: *IEEE Transactions on Fuzzy Systems* 23.4, pp. 861–871 (cit. on p. 64).

Smola, Alex J. and Bernhard Scholkopf (2004). "A Tutorial on Support Vector Regression". In: *Statistics and Computing* 14.3, pp. 199–222. ISSN: 0960-3174. DOI: `10.1023/B:STCO.0000035301.49549.88` (cit. on p. 46).

Stellato, Bartolomeo, Tobias Geyer, and Paul J Goulart (2017). "High-speed finite control set model predictive control for power electronics". In: *IEEE Transactions on power electronics* 32.5, pp. 4007–4020 (cit. on p. 89).

Stulp, Freek and Olivier Sigaud (2012). "Path integral policy improvement with covariance matrix adaptation". In: *arXiv preprint arXiv:1206.4621* (cit. on pp. 38, 41).

Sun, Yi et al. (2009). "Efficient natural evolution strategies". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation.* ACM, pp. 539–546 (cit. on p. 40).

Sutton, R. S., A. G. Barto, and R. J. Williams (1992). "Reinforcement learning is direct adaptive optimal control". In: *IEEE Control Systems Magazine* 12.2, pp. 19–22. ISSN: 1066-033X. DOI: `10.1109/37.126844` (cit. on p. 19).

Sutton, Richard S (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1, pp. 9–44 (cit. on pp. 18–20).

— (1996). "Generalization in reinforcement learning: Successful examples using sparse coarse coding". In: *Advances in neural information processing systems*, pp. 1038–1044 (cit. on p. 20).

Sutton, Richard S and Andrew G Barto (1998). *Reinforcement learning: An introduction.* Vol. 1. MIT press Cambridge (cit. on pp. 94, 114).

— (2018). *Reinforcement learning: An introduction.* Second Edition. MIT press (cit. on pp. 9–15, 17, 18, 20, 25, 27, 30–32, 36, 42, 56, 59, 60, 63, 93, 97).

Sutton, Richard S, Hamid R Maei, and Csaba Szepesvári (2009). “A convergent $o(n)$ temporal-difference algorithm for off-policy learning with linear function approximation”. In: *Advances in neural information processing systems*, pp. 1609–1616 (cit. on p. 30).

Sutton, Richard S et al. (2000). “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems*, pp. 1057–1063 (cit. on p. 40).

Sutton, Richard S et al. (2009). “Fast gradient-descent methods for temporal-difference learning with linear function approximation”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, pp. 993–1000 (cit. on pp. 30, 56, 60, 89).

Sutton, Richard Stuart (1984). “Temporal credit assignment in reinforcement learning”. In: (cit. on p. 18).

Szepesvári, Csaba (2010). “Algorithms for reinforcement learning”. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 4.1, pp. 1–103 (cit. on pp. 14–19, 28, 29, 31).

Szepesvári, Csaba and William D Smart (2004). “Interpolation-based Q-learning”. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM, p. 100 (cit. on p. 27).

Szita, István and András Lörincz (2006). “Learning Tetris using the noisy cross-entropy method”. In: *Neural computation* 18.12, pp. 2936–2941 (cit. on p. 41).

Takagi, Tomohiro and Michio Sugeno (1985). “Fuzzy identification of systems and its applications to modeling and control”. In: *IEEE transactions on systems, man, and cybernetics* 1, pp. 116–132 (cit. on pp. 56, 106).

Tanaka, Kazuo, Hiroshi Ohtake, and Hua O Wang (2009). “Guaranteed cost control of polynomial fuzzy systems via a sum of squares approach”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.2, pp. 561–567 (cit. on pp. 65, 88).

Tanaka, Kazuo and Hua O Wang (2001). *Fuzzy control systems design and analysis: a linear matrix inequality approach*. John Wiley & Sons (cit. on pp. 56, 57, 63, 64).

Theodorou, E., J. Buchli, and S. Schaal (2010). "Reinforcement learning of motor skills in high dimensions: A path integral approach". In: *2010 IEEE International Conference on Robotics and Automation*, pp. 2397–2403. DOI: `10.1109/ROBOT.2010.5509336` (cit. on p. 41).

Theodorou, Evangelos, Jonas Buchli, and Stefan Schaal (2010). "A generalized path integral control approach to reinforcement learning". In: *journal of machine learning research* 11.Nov, pp. 3137–3181 (cit. on p. 37).

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox (2005). *Probabilistic robotics.* MIT press (cit. on p. 44).

Tsitsiklis, John N and Benjamin Van Roy (1997). "Analysis of temporal-diffference learning with function approximation". In: *Advances in neural information processing systems*, pp. 1075–1081 (cit. on p. 29).

Tuan, Hoang Duong et al. (2001). "Parameterized linear matrix inequality techniques in fuzzy control system design". In: *IEEE Transactions on fuzzy systems* 9.2, pp. 324–332 (cit. on p. 65).

Van Hasselt, Hado, Arthur Guez, and David Silver (2016). "Deep reinforcement learning with double q-learning". In: *Thirtieth AAAI conference on artificial intelligence* (cit. on p. 32).

Van Seijen, Harm et al. (2009). "A theoretical and empirical analysis of Expected Sarsa". In: *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning.* IEEE, pp. 177–184 (cit. on p. 20).

Vijayakumar, Sethu and Stefan Schaal (2000). "Locally Weighted Projection Regression: An O(n) Algorithm for Incremental Real Time Learning in High Dimensional Space". In: *in Proceedings of the Seventeenth International Conference on Machine Learning (ICML*, pp. 1079–1086 (cit. on p. 46).

Waldock, Antony and Brian Carse (2008). "Fuzzy Q-learning with an adaptive representation". In: *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence).* IEEE, pp. 720–725 (cit. on p. 27).

Wang, Yang, Brendan O'Donoghue, and Stephen Boyd (2015). "Approximate dynamic programming via iterated Bellman inequalities". In: *International*

Journal of Robust and Nonlinear Control 25.10, pp. 1472–1496 (cit. on pp. 32, 33).

Wang, Ziyu et al. (2015). “Dueling network architectures for deep reinforcement learning”. In: *arXiv preprint arXiv:1511.06581* (cit. on p. 32).

Watkins, Christopher JCH and Peter Dayan (1992). “Q-learning”. In: *Machine learning* 8.3-4, pp. 279–292 (cit. on p. 19).

Watkins, Christopher John Cornish Hellaby (1989). “Learning from delayed rewards”. PhD thesis. University of Cambridge England (cit. on p. 58).

Wei, C. et al. (2015). “Reinforcement-Learning-Based Intelligent Maximum Power Point Tracking Control for Wind Energy Conversion Systems”. In: *IEEE Transactions on Industrial Electronics* 62.10, pp. 6360–6370. ISSN: 0278-0046. DOI: 10.1109/TIE.2015.2420792 (cit. on p. 56).

Werbos, Paul J (1992). “Approximate dynamic programming for real-time control and neural modeling”. In: *Handbook of intelligent control: Neural, fuzzy, and adaptive approaches* 15, pp. 493–525 (cit. on p. 19).

Wiering, Marco and Martijn van Otterlo (2012). *Reinforcement Learning: State-of-the-art*. Vol. 12. Springer Science & Business Media (cit. on pp. 10, 12, 13, 15–17, 24, 30, 31).

Wierstra, Daan et al. (2008). “Natural evolution strategies”. In: *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on.* IEEE, pp. 3381–3387 (cit. on p. 40).

Williams, Ronald J. (1992a). “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine Learning* 8.3, pp. 229–256. ISSN: 1573-0565. DOI: 10.1007/BF00992696 (cit. on pp. 38, 39).

Williams, Ronald J (1992b). “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4, pp. 229–256 (cit. on p. 40).

Witten, Ian H (1977). “An adaptive optimal controller for discrete-time Markov environments”. In: *Information and control* 34.4, pp. 286–295 (cit. on p. 21).

Wu, Huai-Ning and Kai-Yuan Cai (2004). “H 2 guaranteed cost fuzzy control for uncertain nonlinear systems via linear matrix inequalities”. In: *Fuzzy sets and systems* 148.3, pp. 411–429 (cit. on p. 57).

Xu, Xin et al. (2014). “Kernel-based approximate dynamic programming for real-time online learning control: An experimental study”. In: *IEEE Transactions on Control Systems Technology* 22.1, pp. 146–156 (cit. on p. 70).

Yoshida, K. (1999). “Swing-up control of an inverted pendulum by energy-based methods”. In: *Proc. American Control Conf.* Vol. 6, 4045–4047 vol.6. DOI: 10.1109/ACC.1999.786297 (cit. on p. 125).

Zhang, Huaguang et al. (2012). *Adaptive dynamic programming for control: algorithms and stability.* London, U.K.: Springer (cit. on pp. 56, 70).

Zhang, Jilie et al. (2016). “Nearly data-based optimal control for linear discrete model-free systems with delays via reinforcement learning”. In: *International Journal of Systems Science* 47.7, pp. 1563–1573 (cit. on p. 88).