ALGEBRAIC MULTIGRID FOR REINFORCEMENT LEARNING

Omer Ziv

ALGEBRAIC MULTIGRID FOR REINFORCEMENT LEARNING

Research Thesis

Submitted in Partial Fulfillment of the
Requirements for the
Master of Science in Electrical Engeeniring

Omer Ziv

Submitted to the Senate of
the Technion - Israel Institute of Technology

CHESHVAN 5765 HAIFA NOVEMBER 2004

Acknowledgements

The Research Thesis was done under the supervision of Professor Nahum Shimkin in the Faculty of Electrical Engineering. I gratefully acknowledge the constant support and encouragement of my thesis advisor Professor Nahum Shimkin.
I am also indebt to Professor Irad Yavneh for his invaluable advice and support and for exposing me to field of Algebraic Multigrid.

The generous financial help of the Technion, the Fund for Promotion Research at the Technion and the Forchhiemer Foundation Fellowship are gratefully acknowledged.

Contents

List of Figures

List of Algorithms

Abstract

Many control optimization applications in real life are well modeled within the Markov Decision Processes framework (MDP). Reinforcement Learning is an evolving research domain that offers tools to improve control on-line via interaction using a trial and error approach, in problems where an explicit MDP model is unavailable. The well known TD(λ) and the recently introduced LSTD(λ) and λ-LSPE algorithms became standard tools in reinforcement learning. However, in applications with more than a few hundreds of unknowns, TD(λ) is often too slow to converge and LSTD(λ) and λ-LSPE become too computationally demanding. In this dissertation we show how TD(λ) can be speeded up while keeping computational complexity comparable.

We start with the application of a numerical scheme called Algebraic Multigrid (AMG) to speed up policy evaluation when the MDP model is fully known. We then use this approach to fix a deficiency in TD(λ) that causes its convergence rate to deteriorate over time. This is achieved in our proposed level-based Multigrid TD(λ) algorithm (LB-MGTD) by applying TD(λ) at various resolution scales. This algorithm further enables to combine either LSTD(λ) or λ-LSPE applied on a coarse resolution state space with TD(λ) applied on finer resolution levels. The second algorithm we propose is the simultaneous Multigrid TD(λ) algorithm (S-MGTD), which applies updates at different resolution levels simultaneously. We show this algorithm to be equivalent to a TD(λ) variant that uses modified eligibility traces. Experiments show substantial speedup both in the known model case and in the learning case. The main contributions of this dissertation are

- Two novel algorithms that speed up TD(λ) based on the Multigrid approach.
- Convergence proof of S-MGTD for the non-symmetric case. This is surprising since the convergence proof for standard AMG is limited to the symmetric case.
- A proof of TD(λ) convergence when used with modified eligibility traces. The linkage of this variant with S-MGTD suggests that this is expected to speed up convergence.

Notations and Abbreviations

t	Discrete time step
γ	Discount factor
α_t	Learning step
$\mathbf{z}_t$	Eligibility trace vector
g	Scalar reward
$\mathbf{g}$	Vector of one step rewards
$\mathbb{R}$	Real numbers
$\mathbb{E}$	Expectation operator
s	State
a	Action
$\mathcal{S}$	State space
$\phi(s)$	Vector of basis functions
$\mathbf{\Phi}$	Matrix of basis functions. $\mathbf{\Phi}_{i,k} = \phi_k(s = i)$
π	Policy
q	Stationary distribution of a Markov Chain
$\mathbf{D}$	Stationary distribution diagonal matrix. $\mathbf{D} = diag(\pi)$
$\mathbf{I}$	Identity matrix
$\mathbf{P}$	Transition probability matrix. $\mathbf{P}_{i,j} = p(s_{t+1} = j \mid s_t = i)$
p	Probability
K	Number of basis functions
k	Basis function index
$\mathbf{A}$, $\mathbf{B}$	Matrices defined in least-squares based TD methods
$\mathbf{b}$	Vector defined in least-squares based TD methods
θ	Linear coefficient vector
$\mathbf{e}$	Error vector of θ

r_ℓ	Scalar residual at level ℓ
res	Residual vector
ℓ	Resolution level index. The finest resolution is indexed $\ell = 0$. The coarsest is indexed $\ell_{\max}$.
$V(s)$	Cost-to-go value of state s
V	Vector of cost-to-go values
λ	TD learning parameter
d	Temporal difference
n, m	Time indices
i, j	State indices
$\hat{}$	Notation for approximation
^{T}	Transpose

MDP	Markov Decision Process
DP	Dynamic Programming
RL	Reinforcement Learning
TD	Temporal Difference
LS	Least Squares
RLS	Recursive Least Squares
LSPE	Least Squares Policy Estimation
MG	Multigrid
AMG	Algebraic Multigrid
FMG	Full Multigrid
SPD	Symmetric positive definite

Chapter 1

Introduction

Reinforcement learning (RL) [BT95] is an active area of machine learning research with the purpose of deriving methods for seeking optimal control in stochastic dynamic environments. Most RL algorithms relate to standard stochastic dynamic programming (DP) methods, adapting them to work on-line. TD(λ) is a well known family of algorithms within RL research [S88] [TVR97], inspired by the value iteration method in DP [BT95]. It is of gradient descent nature and serves to evaluate a stationary control from measured samples. Recently, two new algorithm families of a least-squares nature known as LSTD(λ) [B02] [XHH02] and λ-LSPE [NB03] [BBN03] were introduced and comprehensively analyzed. They offer considerably faster convergence at the expense of an increase from $O(K)$ to $O(K^2)$ in computational complexity per sample and memory resources, where K is the number of unknown variables to be learned. The algorithms above were successful for solving problems that pose significant challenge for standard methods [T92]. Still, for complicated applications having more than hundreds of unknowns the λ-LSPE and LSTD(λ) algorithms are not feasible due to high complexity and TD(λ) due to slow convergence. It is therefore of interest to seek intermediate algorithms with $O(K)$ complexity that converge faster than TD(λ).

Extensive research in the DP and RL fields has been directed to speeding up convergence by using aggregations and state space decompositions which enable processing at various resolution scales (see literature survey). The underlying idea is to generate a low dimensional approximation of the problem that is cheaper to solve, interpolate its solution, and use it as an approximated solution of the original problem. Major questions are how to approximate the problem and how to interpolate the solution.

In [HA98] a one way coarse-to-fine scheme was combined with value iteration. In order to do so, the continuous state space was disecretized at various resolutions, interpolating the

solution linearly between geometrically adjacent states. It is important to notice the underlying assumption of solution smoothness between adjacent states. To achieve small interpolation error requires a-prior guarantees on the solution smoothness and structure, which poses a heavy limitation for practical problems.

In this dissertation we take a different approach called Multigrid to speed up the convergence of standard algorithms in DP and RL. Multigrid is a family of multilevel numerical methods for accelerating the iterative solution of large sparse linear equation systems [S01]. It is well known in Multigrid literature that the convergence of standard iterative methods that operate locally is slowed down by errors of global nature. The Multigrid idea is to eliminate error components at various resolution scales: local components are removed by standard iterative methods that operate locally, and global error components are eliminated after they are approximated on a coarser grid [TOS01]. It turns out that the above deficiency of standard iterative methods well explored in Multigrid literature, has much in common with deficiencies in standard iterative methods of DP and inherited by TD(λ). It is therefore natural to ask if the solution offered by Multigrid can be applied to speed up DP methods or TD(λ).

The Multigrid approach transfers residual errors and solution corrections between grid levels and uses coarse grids to approximate the error. This differs from the approach in [HA98] described above, which transfers approximated solutions between grid levels, and uses the coarse grid to approximate the solution. rations,

This is a significant conceptual difference. Standard iteration methods eliminate local inconsistencies in the error of the current solution, guaranteeing smoothness of the error in some sense [TOS01]. It therefore makes more sense to interpolate the error resulting after iterations rather than the solution. The potential of applying this Multigrid principle to DP was recognized in [AC88]. However, the applicability of such methods remains limited by the need to manually construct transfer operators between resolution levels. Such constructions are natural when discretizing a continuous problem [MM02] [HA98]. This approach is difficult if not impossible for complex unstructured meshes, especially for applications in which no physical grid or geometrical information exists (see for example [T92]). In particu-

lar, such information usually does not exist in the learning case for which advance knowledge of the model is unavailable.

For a solution to this problem, we turn to an extension of (geometric) Multigrid, known as Algebraic Multigrid (AMG) [B86] [MC87] [S01]. It automates the construction of transfer operators between levels of different scale. This is done by exploiting algebraic relations between unknown variables as reflected by the equation system. The fact that AMG makes no use of any geometrical information and requires no physical grid makes it suitable as a "black box" solver. We note that all the algorithms proposed in this dissertation can be extended to work with geometric Multigrid, when ordered meshes are natural to the problem at hand.

This dissertation has two main parts: in the first we apply AMG to the field of dynamic programming, and in the second to TD learning. In the field of DP, we use AMG to speed up standard methods for policy evaluation and for seeking optimal control. It turns out that AMG is natural to speed up the well-known value iteration and its variations. In addition, standard AMG literature provides systematic theoretical basis and tools to automate the construction of aggregates. This offers some insight and tools to address a key open question of how to form aggregation or task hierarchies automatically. We propose an AMG scheme for policy evaluation and discuss convergence issues. We show conditions for which aggregation forms a low dimensional Markov chain. Potentially this may be used to seek optimal control for a reduced state MDP. This approach is kept for future work.

Inspired by the analysis and the positive results of AMG application to DP, we apply similar tools to TD learning. We analyze the dynamics of TD(λ) to reveal a deficiency which limits its asymptotic convergence rate. Based on this observation, we propose a novel Multigrid-TD algorithm which speeds up TD(λ) and has a comparable computational complexity. Based on this algorithm, we show that TD(λ) with an over-complete basis, may have an improved convergence rate over standard TD(λ). Our research suggests how this basis should be selected.

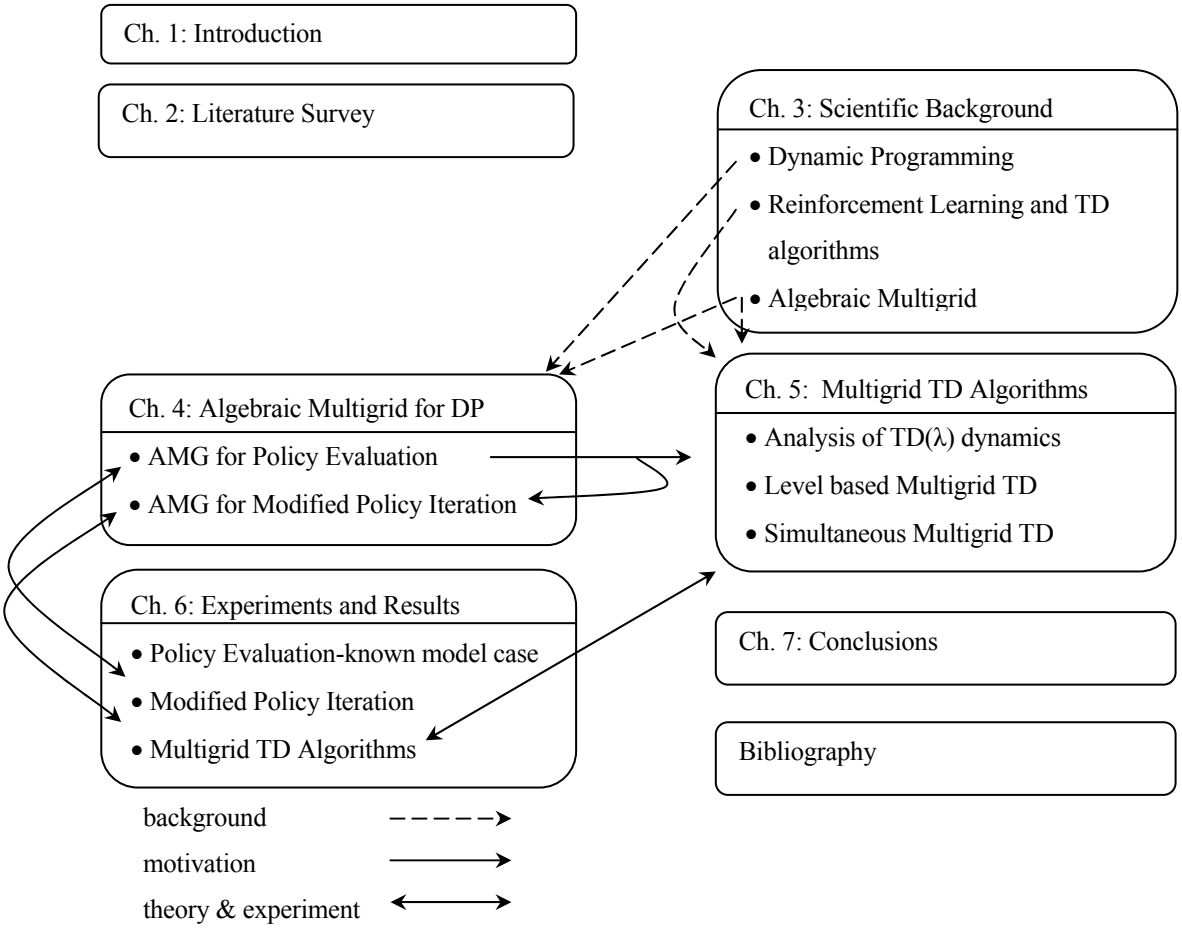
The contributions of this dissertation are

- An anlysis of the dynamics of TD(λ) reveals a deficiency that limits its asymptotic convergence rate.
- Two novel algorithms that speed up TD(λ) based on the Multigrid approach, named LB-MGTD and S-MGTD. To our knoweledge, this is the first time the link between the Multigrid approach and TD(λ) has been established.
- The LB-MGTD algorithm enables to apply the computationaly cheap TD(λ) on fine grids combined with the fast LSTD or λ-LSPE algorithms applied on the coarsest grid. Experimental results show that this is efficient in bottom-up initialization, however it less practical in a V-cycle scheme due to noise amplification.
- Seperate convergence of each level in LB-MGTD was established.
- Convergence proof of S-MGTD for the non-symmetric case was established. This is surprising since the convergence proof for standard AMG is limited to the symmetric case.
- A proof of TD(λ) convergence when used with modified eligibility traces. The linkage of this variant with S-MGTD suggests that this is expected to speed up convergence.

Chapter Layout

The dissertation is organized as follows. Chapter 2 provides a review on previous related work. Chapter 3 gives background on Reinforcement Learning and on Algebraic Multigrid. The review is rather detailed, since it is not common for researchers from one field to be familiar with the other. A reader familiar with either RL or AMG may skip the corresponding sections. In chapter 4, we propose and investigate the application of AMG to accelerate algorithms within the Dynamic Programming domain. The problem addressed in this chapter assumes complete knowledge of the MDP model. In chapter 5, we consider approximate policy evaluation for the unknown model case. We propose a Multigrid algorithm for TD(λ) and discuss its convergence. This algorithm shows how to combine TD(λ) and least squares based TD methods, such as LSTD(λ) or λ-LSPE. Our algorithm benefits from the advantages of each, enabling the acceleration of TD(λ) or dually, the refinement of least squares TD

methods. Experiments demonstrating issues discussed in chapters 4-5 are given in chapter 6. Chapter 7 concludes our dissertation. A graphical layout is given below.



Chapter 2

Literature Survey

2.1 Aggregation for Dynamic Programming

Markov decision processes (MDPs) have proven very useful for modeling sequential control problems in a stochastic environment [BT95] [B99] [FS02]. Though well based theoretically, traditional iterative methods of dynamic programming, which have a complexity polynomial in the number of states, remain computationally intractable for problems with very large state spaces.

It has long been realized that one of the reasons these methods are slow to converge is that they apply corrections of local nature only. For this reason several authors proposed the use of aggregation-disaggregation ideas to accelerate these iterative methods. A description of the aggregation technique is presented [CM82]. The idea is to partition the state space into disjoint groups called aggregates, and apply the same additive value correction to all the states in an aggregate. This allows for corrections of global nature along with local ones. The two-level scheme presented in [CM82] was extended in [AC88] to a Multigrid scheme. These papers do not however state how to construct the aggregates in practice and do not address convergence issues.

Several attempts were made to answer the question of how to form aggregates [SPK84], [FD02] [BC89]. In [SPK84] it was suggested to group states of similar one step rewards, with the hope that this will exploit some natural structure of the MDP. This heuristic fails in many applications, especially with delayed reward for which the one step reward is the same for all but a few goal states. In [FD02] value functions, which we define later in section 3.1, are used to reveal MDP structure. Aggregation is based on exploiting similarity between

value functions of different MDPs defined on the same state space. This is natural in maze navigation applications for which various goals are defined for the single maze. The main idea is to use correlations between various solutions to identify states that are likely to share similar values in a new MDP problem. This approach however does not suit aggregation for speeding up the evaluation of a single MDP. In [BC89] it is proposed to group states of similar residual into an aggregated state sharing the same value correction. Since residuals change during iterations, this approach results in a dynamic formation of aggregations. The paper motivates this aggregation method, as it minimizes one of two terms, whose sum determines a bound on the error (see equation (18) in [BC89]). Unfortunately, it is hard to justify the neglect of the second term. Moreover, the methods reviewed above lack strong theoretical grounds to guide aggregation construction. Still, a theory that links the iteration operator to aggregate formation has been well known in the Multigrid community since the early 1980s with the introduction of Algebraic Multigrid, which we review in section 2.4.

2.2 Multilevel approaches for Reinforcement Learning

Most RL algorithms are related to standard stochastic dynamic programming methods, adapting them to work on-line. By focusing computational effort along trajectories of high interest and applying function approximation, RL methods succeeded to solve complex problems, for which DP methods proved infeasible [T92]. Unfortunately, RL methods suffer from the same curse of dimensionality as DP methods, making the computational time required to solve real life problems with increasingly large state spaces, impractical.

To cope with this problem, much of the research during the last decade has been directed toward the exploitation of temporal abstraction, where the original problem is decomposed into a sequence of simpler tasks. We refer the reader to [DH92] [D00] and a comprehensive review of such principled methods in [BM03]. The methods presented are well based within the framework of semi-Markov decision processes (SMDP). However, in order to exploit temporal abstraction, the developer is required to manually decompose the problem based on prior knowledge. Though natural for some problems, a manual decomposition may be diffi-

cult if not impossible in many practical problems. This is especially true for learning in which problem structure may be unavailable. A few attempts have been made to automate the decomposition process [D98], [MMS02]. These methods focus on the automatic identification of sub-goal states, splitting the problem into the task of reaching the sub-goal, and reaching the goal state from the sub-goal state.

A different approach which adapts aggregates during learning was taken in [SJJ95], [KA99], [MMS03]. Instead of aggregates formed by disjoint partitioning of the state space, a "soft" aggregation is introduced in which a state may belong to several aggregates [SJJ95]. Weights determine how much each state belongs to every aggregate. This defines a linear mapping from a low dimensional space (aggregates) to the original state space. The mapping from an aggregate to its states is known as a basis function. The weights defining the basis functions are adapted using gradient descent in [SJJ95] [MMS03], and cross-entropy in [MMS03]. The approach closest to the one we use is given in [KA99]. Weights are adapted so that if the transition probability from one state to another is high, they are adapted to share an aggregate. Algebraic Multigrid analysis shows that standard iterations cause such states to share a common error [S01], so that applying the same correction makes sense.

2.3 TD learning algorithms - recent advances

Among TD learning algorithms, TD(λ) originally introduced by Sutton [S88] is probably the simplest and most popular one. The original algorithm used a "look up" table to represent the value function (see section 3.3.3). It was extended to adapt a parametric approximation of the value function as a linear combination of basis functions (see section 3.4.3). This extension offers tractability for problems with a large state space. The TD(λ) algorithm is a gradient-descent like algorithm. Its convergence with probability 1 was established in [TVR97] by rewriting it as a stochastic approximation. It was also shown that on-line sampling, that is sampling states according to the stationary probabilities of the Markov chain, and the linearity of the value approximator, are important to establish convergence, with counter examples included.

TD(λ) has two major drawbacks: it might make inefficient use of data, and it requires a handcrafted step-size schedule that heavily effects both convergence and performance. Based on the theory of least-squares estimation, two algorithm families were recently introduced. They overcome the above deficiencies of TD(λ) at the expense of increased computation per time-step from $O(K)$ to $O(K^3)$ and of memory resources from $O(K)$ to $O(K^2)$, where K is the number of parameters. The first is Least-Squares TD(λ) (LSTD(λ)) proposed by Boyan in [B02] and proved to converge with probability 1 in [NB03]. It uses data to generate an empirical equation system and solves it directly. Though this algorithm converges much faster than TD(λ) (see [B02]) it does not enable to incorporate a-prior solution. An efficient implementation of LSTD(λ) with a complexity of $O(K^2)$ per time-step was introduced in [KXZ03]. The inversion of the matrix is circumvented by using a recursive least-squares like algorithm. As in other RLS methods, this RLS-TD(λ) algorithm enables to incorporate an initial solution, and has a forgetting factor which may be used to adapt to a slowly varying Markov chain.

The second algorithm family is the λ-Least Squares Policy Evaluation (λ-LSPE) introduced in [NB03]. This algorithm family combines a gradient-like algorithm involving least-squares sub-problems. Its convergence with probability 1 for a diminishing size-step is proven in [NB03] and extended to a fixed size-step in [BBN03]. Its efficient implementation of $O(K^2)$ complexity is given in [NB03] including an analysis showing equivalence of performance to the LSTD(λ) algorithm. As it is a gradient-like algorithm, it requires an initial solution. We present technical detail on both LSTD(λ) and λ-LSPE later on in section 3.4.4.

2.4 Algebraic Multigrid

The Multigrid method was originally introduced by Fedorenko [F61]. The actual efficiency of Multigrid was first realized by Brandt [B72]. Independently, Multigrid methods have been introduced by Hackbusch [H76]. The original motivation for Multigrid was the efficient numerical solution of large systems of discretized elliptic partial differential equations, which requires algorithms to effectively reduce error components of both local and global nature.

This is achieved by (geometric) Multigrid, which is a family of methods that operate on a hierarchy of grids. Grids are defined a-priori by discretizing the continuous space at different scales. Though straightforward for many applications, the definition of a grid hierarchy may be highly difficult for complex, unstructured meshes. The Achilles heel is the use of geometric relations between grid points to define two inter-level operators: interpolation and restriction. In the 1980s Brandt [B86] introduced Algebraic Multigrid (AMG) further developed by McCormick, Ruge and Stüben in [MC87]. AMG combines operator-dependent interpolation and the Galerkin based coarse grid correction with Multigrid. The main idea is that a reasonable interpolator and the Galerkin operator can be derived directly from the underlying matrices, without any reference to a physical grid. Increased interest in AMG has risen in the 1990's with an increase in the complexity of geometrical applications such as computational fluid dynamics, and the need for efficient "plug-in" solvers [TOS01]. Consequently, there are now many different algebraic approaches, which mainly differ in the construction of inter-level operators [S01] [VH99]. We leave the technical presentation of AMG to section 3.6.

Chapter 3

Scientific Background

This chapter reviews *Dynamic Programming*, *Reinforcement Learning*, and *Algebraic Multigrid*: the first two provide methods for addressing problems formulated as *Markov Decision Problem* (MDPs); and the latter is a family of efficient numerical solvers. In the first section we described the MDP framework used to model our problem. Next, we review *Dynamic Programming* (DP) techniques, which are often used when the MDP model of the problem is fully known. We state the *Bellman equation*, which is a set of equations that define the solution of the MDP, and provide the basis for the subsequent methods. We continue by describing a method for evaluating any fixed policy and methods for finding an optimal policy.

Next, we review *Reinforcement Learning* (RL) techniques, which although inspired by DP methods, do not require an explicit model of the MDP. Instead, RL methods use measured data while exploring the policy space in a trial-and-error approach. We start with a general description of RL and continue with a review of a popular family of RL algorithms known as *Temporal-Difference* (TD) methods. We review TD methods that use linear function approximation. Approximation is used to circumvent a representational difficulty arising in complex real life problems. We provide detail on three popular algorithms in this family and compare their advantages and disadvantages.

The last two sections review *Algebraic Multigrid* (AMG). This is a family of multilevel numerical methods that speed-up iterative solution of large linear equation systems. The first of these two sections reviews AMG algorithms without going into technical detail, and addresses theoretical and practical issues regarding convergence. The last section covers technical detail of the *setup* phase in AMG.

For comprehensive reviews of Dynamic Programming and Reinforcement Learning we refer the reader to [BT95], [SB98], and [P94]. Comprehensive reviews on Multigrid and AMG are given in [TOS01] [BHMC00], and [MC87]. A basic tutorial on AMG is provided in [VH99] and an in-depth tutorial is given in [W99].

3.1 Problem modeling and the MDP framework

Consider an interaction between an *agent* and a *dynamic environment*. Interaction takes place at discrete time decision points, denoted $t = 0,1,2,\ldots$, in which the agent selects an *action* $a_t \in \mathcal{A}$ causing a change in the environment state and consequently receiving some reward signal. In many applications the environment may be modeled as a Markov Decision Process (MDP).

Formally, an MDP is defined by a 4-tuple $\{\mathcal{S},\mathcal{A},P,g\}$. $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces respectively. For brevity, we excluded the dependence of $\mathcal{A}(s)$ on the current state. P defines the environment dynamics via transition probabilities $p(s_{t+1} \mid s_t, a_t)$, from state s_t to state s_{t+1} given the action taken is a_t. The (scalar) *immediate reward* sent following such a transition is denoted $g(s_t, s_{t+1}, a_t)$ or g_t for short. The *Markov* property states that given the current state and action, the dynamics of the environment and the immediate reward are statistically independent of previous events. For future use we note that an MDP is said to be *finite* if its state space $\mathcal{S}$ is finite, and is called an *infinite MDP* if the number of states in $\mathcal{S}$ is infinite.

Following the agent's action a_t, the agent receives two signals: the new environment state s_{t+1} and the reward g_t. The agent chooses actions based on a *policy*, denoted by π, which is a mapping from the history of observations and actions up to time t, denoted $\mathcal{H}_t$, to the action to be taken, denoted a_t, i.e.

$$\pi : \mathcal{H}_t \to \mathcal{A} \tag{3.1.1}$$

where $\mathcal{H}_t \equiv \{s_0, s_1, \ldots, s_t, a_0, a_1, \ldots, a_{t-1}, g_0, g_1, \ldots, g_{t-1}\}$.

In some cases, the policy is stochastic, giving the probability for taking each action

$$\pi : \mathcal{H}_t \to \Delta(\mathcal{A}) \tag{3.1.2}$$

where $\Delta(\mathcal{A})$ is the set of probablilty distributions over the action set $\mathcal{A}$.

The objective of the agent is given in the form of a utility function, denoted $v^{\pi}(s)$, measuring the quality of any policy π when the agent starts from state s. This function is named the *value function* or *cost-to-go function*. The goal of the agent is to find an optimal policy denoted π^* that maximizes the value function for all states

$$\pi^* = \arg\max_{\pi} v^{\pi}(s), \quad \forall s \in \mathcal{S}. \tag{3.1.3}$$

A popular value function to be maximized is the expected total reward gained in a finite number of steps, denoted T,

$$v^{\pi}(s) = \mathbb{E}^{\pi}\left(\sum_{t=0}^{T-1} g_t \mid s_0 = s\right), \tag{3.1.1}$$

where the expectation is taken with respect to all trajectories of length T starting at state *s* and following the (random) policy π. This value function is appropriate when the number of steps pre-defined. If the number of steps is unknown or random, such as in the case of termination upon reaching a goal state, an infinite-horizon value function is better suited

$$v^{\pi}(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} g_t \mid s_0 = s\right). \tag{3.1.2}$$

For this value function to be bounded, certain technical conditions must be met [BT95]. Other popular value functions with infinite-horizon are the *discounted reward*

$$v^{\pi}(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t g_t \mid s_0 = s\right) \tag{3.1.3}$$

where $\gamma \in [0,1)$ is known as the *discount factor*, and the *average reward*

$$v^{\pi}(s) = \liminf_{T\to\infty} \mathbb{E}\left(\frac{1}{T} \sum_{t=0}^{T-1} g_t \mid s_0 = s \right). \tag{3.1.4}$$

The first emphasizes rewards at the beginning of the process while the latter emphasizes long-term revenues.

In this dissertation, we address either finite or infinite MDPs with infinite-horizon discount cost, i.e. (3.1.3). We address finite MDPs in the context of dynamic programming and infinite MDPs in the context of learning with function approximation both to be introduced later. We note that our approach applies to infinite MDPs in the dynamic programming context as well when introducing function approximation.

It is well known, that for such MDPs with discounted cost there exists a *deterministic stationary policy* which is optimal [BT95]. A *Markov policy* selects an action based solely on the current state. If the rule for selecting an action in a *Markov policy* does not change over time, the policy is said to be *stationary*. A *deterministic policy* specifies a single action $a = \pi(s)$, whereas a *randomized policy* defines a probability for selecting each action $p(a \mid s) = \pi(s,a)$ as a mapping $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. Applying a *stationary policy* to an MDP induces a *Markov chain*, with the transition probabilities

$$p(s' \mid s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a)\, p(s' \mid s,a). \tag{3.1.5}$$

We assume henceforth that this Markov chain is *irreducible*, *aperiodic* and has a unique distribution denoted $q(s)$, that satisfies

$$q(s') = \sum_{s \in \mathcal{S}} p(s' \mid s) q(s). \tag{3.1.6}$$

3.2 Dynamic Programming

Stochastic Dynamic Programming (DP) algorithms are methods for finding optimal policies when a complete model of the MDP is known. The optimal policy is known to satisfy

the *Bellman optimality equations* [BT95]. Dynamic Programming methods efficiently solve these equations by using two processes: one evaluating the cost-to-go function of the current policy, and the other improving the current policy based on the current cost-to-go estimate. After introducing the *Bellman equation*, we review the two processes above, and the various ways in which they are combined.

3.2.1 The Bellman equations

Consider a stationary policy π. We note that (3.1.3) can be used to write the cost-to-go of π for any state $s \in \mathcal{S}$ in terms of the values of its potential successive states $s' \in \mathcal{S}$ via the *Bellman equation* [BT95]:

$$v^{\mathrm{p}}(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \left[g(s,s',a) + \gamma v^{\mathrm{p}}(s') \right] \tag{3.2.7}$$

The value function of a given policy is the unique solution of this linear equation system. It will turn out convenient to write this equation system in vector notation

$$\mathbf{v}^{\mathrm{p}} = \mathbf{g}^{\mathrm{p}} + \gamma \mathbf{P}^{\pi} \mathbf{v}^{\mathrm{p}} \tag{3.2.8}$$

where $\mathbf{v}^{\mathrm{p}}$ is a vector of values of each state, $\mathbf{g}^{\mathrm{p}}$ a vector of the one step rewards, and $\mathbf{P}^{\pi}$ is the transition probability matrix, all defined below.

$$\mathbf{v}^{\mathrm{p}} = \begin{bmatrix} v^{\pi}(1) & v^{\pi}(2) & \cdots & v^{\pi}(N) \end{bmatrix}^{T} \tag{3.2.9}$$

$$\mathbf{g}^{\pi} = \begin{bmatrix} g^{\pi}(1) & g^{\pi}(2) & \cdots & g^{\pi}(N) \end{bmatrix}^{T} \tag{3.2.10}$$

$$g^{\pi}(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in S} p(s' \mid s,a) g(s,s',a) \tag{3.2.11}$$

$$\mathbf{P}^{\pi}_{s,s'} = \sum_{a \in \mathcal{A}(s)} \pi(s,a) p(s' \mid s,a) \tag{3.2.12}$$

The value function of the optimal policy is known to be the unique solution of the set of nonlinear equations named the *Bellman optimality equation* [BT95]:

$$v^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \left[g(s,s',a) + \gamma v^*(s') \right] \tag{3.2.13}$$

where v^* denotes the optimal value function. Given v^*, an optimal policy is derived by

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \left[g(s,s',a) + \gamma v^*(s') \right]. \tag{3.2.14}$$

We now turn to methods for efficiently solving the Bellman equation.

3.2.2 Policy Evaluation

In this section, we review a method for evaluating the cost-to-go of a given stationary policy denoted by π. This method is beneficial when the number of states is too large, making the direct solution of (3.2.7) intractable. Starting with an arbitrary function $v_0^\pi(s)$, we update all the states simultaneously by applying the iteration

$$v_{n+1}^\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \left[g(s,s',a) + \gamma v_n^\pi(s') \right] \tag{3.2.15}$$

or in vector notation

$$\mathbf{v}_{n+1}^\pi = \mathbf{g}^\pi + \gamma \mathbf{P}^\pi \mathbf{v}_n^\pi. \tag{3.2.16}$$

This iteration converges asymptotically to the value function of π.

3.2.3 Policy Iteration

The next two schemes find the optimal policy and value function. *Policy Iteration* separates the *policy evaluation* and *policy improvement* processes. Iterations are done in the space of deterministic policies. Starting with some arbitrary policy π_0, each iteration is carried out in two steps. The first evaluates the value function v^{π_n} of the current policy π_n by solving (3.2.7). In the second step, the current policy is improved using

$$\pi_{n+1}(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \left[g(s,s',a) + \gamma v^{\pi_n}(s') \right]. \tag{3.2.17}$$

It is well known that Policy Iteration for finite MDPs converges to the optimal policy in a finite number of iterations [BT95]. A drawback of this method is the high computational effort needed for solving (3.2.7) exactly.

3.2.4 Value Iteration

Value Iteration replaces the *policy evaluation* step, which is computationally expensive, by a single execution of (3.2.15). This can be compactly written as

$$v_{n+1}(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \left[g(s, s', a) + \gamma v_n(s') \right] \tag{3.2.18}$$

Starting from an arbitrary initial function $v_0(s)$, Value Iteration asymptotically converges to the optimal cost-to-go function. The *greedy policy* π_n^g defined below converges to the optimal policy after a finite number of iterations.

$$\pi_n^g(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \left[g(s, s', a) + \gamma v_n(s') \right] \tag{3.2.19}$$

3.2.5 Asynchronous Value Iteration

In Value Iteration described above, the values of all the states are updated simultaneously. In *Asynchronous Value Iteration*, states are updated according to some predefined order $s_1, s_2, s_3, \ldots$, using the most up-to-date values in each update. The update process takes the form

$$v_{n+1}(s) = \begin{cases} \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \left[g(s, s', a) + \gamma v_n(s') \right] & \text{if } s = s_n \\ v_n(s) & \text{otherwise} \end{cases}. \tag{3.2.20}$$

It is proven in [BT95], that this procedure converges to the optimal cost-to-go for arbitrary state ordering, as long as every state is updated *infinitely often*.

We mention *Gauss-Seidel methods* as a special case, in which states are swept in cyclic lexical order, i.e. $s_1 = 1, s_2 = 2, s_3 = 3, \ldots, s_1 = 1, s_2 = 2, \ldots$, while applying (3.2.15) for policy

evaluation and (3.2.18) for optimal policy search. In practice, this iteration usually converges faster than the one defined in (3.2.15) and (3.2.18).

3.2.6 Modified Policy Iteration

We conclude this review of Dynamic Programming methods by mentioning a blend of Value Iteration and Policy Iteration named *Modified Policy Iteration* or *Generalized Policy Iteration* [BT95]. It is an intermediate mixture of the *policy evaluation* and *policy improvement* processes, which applies (3.2.15) several times between executions of (3.2.17). Empirical evidence suggests it has faster convergence than either VI or PI when computational aspects are taken into account.

3.3 Introduction to Reinforcement Learning

3.3.1 What is Reinforcement Learning?

Reinforcement Learning (RL) is a branch of machine learning research, specialized for learning optimal control in an unknown environment using a trial-and-error approach. Instead of assuming that the MDP model is fully known, learning is done by measuring the outcomes of actions. After taking an action $a_t \in \mathcal{A}$, the agent observes the state transition $s_t \to s_{t+1}$ and the measured reward g_t, which are samples of the stochastic MDP. The agent then uses this information to update its current representation of either the cost-to-go function or the optimal policy. RL algorithms are designed to converge in some probabilistic sense, when the update rule is applied iteratively.

In this dissertation, we address a popular family of RL algorithms known as *Temporal-Difference* (TD) learning. TD learning is often used to replace the policy evaluation step of DP, in order to learn the value function of a fixed policy, π. In this section, we review algorithms that represent the estimated value function using a "lookup table" (that is, the estimated value of each state is stored separately). We address the use of function approximation for the value function in the next section. We first introduce the basic TD(0)

algorithm, followed by the extension to the TD(λ) algorithm family. For ease of notation, we omit the explicit indication of the dependency of the value function on π, and write v instead of v^{π}.

3.3.2 Temporal Difference Update – basics

Temporal-Difference (TD) methods use data obtained from simulated *trajectories* to estimate the value function of a stationary policy π. A *trajectory* is a series of states $s_0, s_1, s_2, \ldots$, dictated by transition probabilities of the MDP when actions are taken according to π, and the corresponding single step rewards $g(s_0, s_1), g(s_1, s_2), \ldots$. After each transition, the *temporal-difference*

$$d(s_t, s_{t+1}; \hat{v}) = \left[g(s_t, s_{t+1}) + \gamma \hat{v}(s_{t+1}) \right] - \hat{v}(s_t) \tag{3.3.21}$$

is calculated, with $\hat{v}(s)$ being an estimate of value function. The first term in (3.3.21) is a sample of the estimation of $v(s_t)$ using a one-step look ahead. Therefore, the temporal difference is a stochastic sample of the one-step inconsistency of our current estimation in the *Bellman equation*. We note that the mean of (3.3.21) is zero for all states, only if $\hat{v}(s)$ equals the true value function.

3.3.3 TD(λ)

TD(0) is a simple algorithm that iteratively applies the update rule

$$v(s_t) := v(s_t) + \alpha_t d(s_t, s_{t+1}; v) \tag{3.3.22}$$

where α_t is a sequence of learning steps. The state transition $s_t \to s_{t+1}$ is used to update the estimation at state s_t only, while keeping the estimation at other states fixed. This can be regarded as applying local corrections, making the value of each state more consistent with its one-step look ahead estimation. TD(0) is inefficient in the sense that it uses each measurement to update a single state.

TD(λ) is a family of algorithms that use each measured temporal difference to update all the states simultaneously using

$$v(s) := v(s) + \alpha_t d(s_t, s_{t+1}; v) z_t(s) \tag{3.3.23}$$

where the weight $z_t(s)$ determines the relevance of the temporal difference at time t to the updated state. The weights $z_t(s)$, named eligibility traces, provide more weight to recent states over states that were not visited for a long time. A common way for defining the eligibility traces is

$$z_t(s) := \begin{cases} \gamma\lambda z_{t-1}(s) + 1 & \text{if } s = s_t \\ \gamma\lambda z_{t-1}(s) & \text{otherwise} \end{cases} \tag{3.3.24}$$

where $z_t(s)$ is initialized as zero for all states. It is easy to see that $z_t(s)$ is relatively large when s is visited often, and state s_t is reached from s using a small number of steps. The weight $z_t(s)$ decays exponentially with the number of steps from the last time state s was visited, where the parameter $\lambda \in [0,1]$ defines the decay rate (beyond γ, the reward discount factor). As λ approaches 1, more states are influenced by each sample. For $\lambda = 0$ we obtain TD(0) as a special case.

TD(λ) converges with probability one to the true value function, under technical conditions discussed in [BT95]. The relation between λ and the convergence rate was studied in [SD98], where it was shown that an intermediate value of λ is optimal.

3.4 TD Methods for Linear Function Approximation

3.4.1 Linear Function Approximation

The number of states in practical problems may be very large, rendering the representation of the value function using a "lookup table" impractical. As an example we note that in Tesauro's TD-Gammon application [T92] the number of states is estimated at over than 10^{20}.

One way to tackle this problem is to approximate the value function using some parametric form. An approximation method that proved stable, both in theory and in practice [TVR97], is a linear combination of the form

$$\hat{v}_\theta(s) = \phi(s)^T \theta \tag{3.4.25}$$

where $\phi(s) \in \mathbb{R}^K$ is a vector valued function, with its elements weighted by the elements of the vector $\theta \in \mathbb{R}^K$. The elements $\phi_k(s)$ of $\phi(s)$ are known as *basis functions* or *features*. Applying approximation provides tractability and generalization at the expense of possible solution accuracy.

3.4.2 Temporal Difference using Function Approximation

When substituting (3.4.25), the temporal difference element of (3.3.21) takes the form

$$d(s_t, s_{t+1}; \theta) = g(s_t, s_{t+1}) - (\phi(s_t) - \gamma\phi(s_{t+1}))^T \theta \tag{3.4.26}$$

where θ is the parameter vector of the current estimate. The temporal differences are used to improve our current estimate θ_t to obtain a new estimate θ_{t+1}

3.4.3 TD(λ) with Function Approximation

Sutton's TD(λ) [S88] extends the tabular TD(λ) to the case of linear function approximation. The parameter vector θ_t is updated every time step using the recent temporal difference. The update rule is:

$$\mathbf{z}_t = \lambda\gamma\mathbf{z}_{t-1} + \phi(s_t) \tag{3.4.27}$$

$$\theta_t = \theta_{t-1} + \alpha_t \mathbf{z}_t d(s_t, s_{t+1}; \theta_{t-1}) \tag{3.4.28}$$

Similar to the tabular case, $\mathbf{z}_t$ is a vector of *eligibility traces* that measure the relevance of the recent temporal difference to each element of θ_t. It is initialized as $\mathbf{z}_{-1} = \mathbf{0}_{K\times 1}$. $\lambda \in [0,1]$ is an algorithm parameter, and α_t is a non-increasing positive step size sequence.

Under the assumption set stated below, TD(λ) with linear function approximation converges with probability one [TVR97] to a unique fixed point θ^*, which satisfies

$$\mathbf{A}\theta^* = \mathbf{b} \tag{3.4.29}$$

$$\mathbf{A} = \mathbf{\Phi}^T \mathbf{D}\mathbf{M}_\lambda \left(\mathbf{I} - \gamma \mathbf{P}\right)\mathbf{\Phi} \tag{3.4.30}$$

$$\mathbf{b} = \mathbf{\Phi}^T \mathbf{D}\mathbf{M}_\lambda \mathbf{g} \tag{3.4.31}$$

$$\mathbf{M}_\lambda = \left(\mathbf{I} - \gamma\lambda\mathbf{P}\right)^{-1} \tag{3.4.32}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{\Phi}$ is a matrix of basis functions, and $\mathbf{D}$ a diagonal matrix of stationary distributions, all defined below. For the definitions of $\mathbf{g}$ and $\mathbf{P}$ see (3.2.10)-(3.2.12).

$$\mathbf{\Phi} = \begin{bmatrix} | & | & & | \\ \phi_1(s) & \phi_2(s) & \cdots & \phi_K(s) \\ | & | & & | \end{bmatrix}_{N\times K}, \quad \mathbf{D} = \begin{bmatrix} q(1) & 0 & \cdots & 0 \\ 0 & q(2) & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & q(N) \end{bmatrix}_{N\times N} \tag{3.4.33}$$

$q(i)$ is the steady state (stationary) distribution probability of being in state $i \in \mathcal{S}$.

As opposed to TD(λ) in the tabular form that converges to the same unique value function for all values of λ, the convergence point of TD(λ) with function approximation depends on λ. We rewrite (3.4.29) as

$$\left(\mathbf{\Phi}^T \mathbf{D}\mathbf{M}_\lambda\right)\left(\mathbf{g} - \left(\mathbf{I} - \gamma\mathbf{P}\right)\left(\mathbf{\Phi}\theta^*\right)\right) = \mathbf{0}. \tag{3.4.34}$$

The second term is the residual of the Bellman equation (3.2.8), where $\mathbf{\Phi}\theta$ is the approximated value function. We observe that the algorithm projects the residual on a lower dimensional subspace, and equates the projection to zero. Since the projection depends on λ, so does the convergence point. In [TVR97] the following projection matrix is defined

$$\Pi = \mathbf{\Phi}\left(\mathbf{\Phi}^T \mathbf{D}\mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^T \mathbf{D}. \tag{3.4.35}$$

For the special case of $\lambda = 0$ it may be seen that this projection of the residual equals zero

$$\Pi\left(\mathbf{g} - \left(\mathbf{I} - \gamma\mathbf{P}\right)\mathbf{\Phi}\theta^*\right) = \mathbf{0} \tag{3.4.36}$$

and for the case of $\lambda = 1$ convergence is to the projection of the true value function

$$\theta^* = \Pi\mathbf{v} \tag{3.4.37}$$

where $\mathbf{v} = \left(\mathbf{I} - \gamma\mathbf{P}\right)^{-1}\mathbf{g}$.

For completeness, we state the assumptions used in [TVR97] to prove the convergence of TD(λ). We stress that assumptions 1(b), 2(b) and 3 are needed in the case of infinite (countable) state space, and are automatically satisfied for finite MDPs [TVR97].

TD(λ) assumption set

1(a). The Markov chain s_t is irreducible, aperiodic, and has a unique stationary distribution, $\mathbf{q}$ satisfying $\mathbf{q}^T\mathbf{P} = \mathbf{q}^T$. $\mathbb{E}_0(\cdot)$ denotes the expectation with respect to $\mathbf{q}$.

1(b). One step rewards satisfy $\mathbb{E}_0\left(g^2(s_t, s_{t+1})\right) < \infty$

2(a). The matrix $\mathbf{\Phi}$ is full rank

2(b). Basis functions satisfy $\mathbb{E}_0\left(\phi_k^2(s)\right) < \infty$

3. The effects of the initial state are bounded by a function $f: \mathcal{S} \to \mathbb{R}^+$ satisfying the following requirements:

3(a). For all s_0 and $m \geq 0$,

$$\sum_{\tau=0}^{\infty} \left\| \mathbb{E}\left(\phi(s_\tau)\phi(s_{\tau+m})^T \mid s_0\right) - \mathbb{E}_0\left(\phi(s_0)\phi(s_m)^T\right) \right\| \leq f(s_0)$$

$$\sum_{\tau=0}^{\infty} \left\| \mathbb{E}\left(\phi(s_\tau)g(s_{\tau+m}, s_{\tau+m+1}) \mid s_0\right) - \mathbb{E}_0\left(\phi(s_0)g(s_m, s_{m+1})\right) \right\| \leq f(s_0)$$

3(b). For any $n > 1$, there exists a constant μ_n such that for all s_0, t,

$$\mathbb{E}\left(f^n(s_t) \mid s_0\right) \leq \mu_n f^n(s_0)$$

4. The step-size sequence α_t is positive, non-increasing, predetermined, and satisfies $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$.

Throughout this dissertation, $\|\cdot\|$ without a subscript denotes the Euclidean norm for vectors and the Euclidean-induced norm for matrices.

3.4.4 Least-Squares based methods

When the number of unknowns *K* is small, it may be feasible to approximate and directly solve equation (3.4.29). In the LSTD(λ) algorithm, the matrix **A** and vector **b** in (3.4.30) and (3.4.31) respectively, are sampled by

$$\mathbf{A}_t = \sum_{n=0}^{t} \mathbf{z}_n \left(\phi(s_n) - \gamma\phi(s_{n+1})\right)^T \text{ and } \mathbf{b}_t = \sum_{n=0}^{t} \mathbf{z}_n g(s_n, s_{n+1}). \tag{3.4.38}$$

For time *t* in which $\mathbf{A}_t$ is non-singular, the estimate of θ_t is given by the solution

$$\theta_t = \mathbf{A}_t^{-1}\mathbf{b}_t . \tag{3.4.39}$$

Another algorithm is the λ-LSPE algorithm. It samples an additional matrix

$$\mathbf{B}_t = \sum_{n=0}^{t} \phi(s_n)\phi(s_n)^T , \tag{3.4.40}$$

and starting from some initial guess θ_0 applies the iteration

$$\theta_{t+1} = \theta_t + \alpha_t \mathbf{B}_t^{-1}\left(\mathbf{b}_t - \mathbf{A}_t\theta_t\right), \tag{3.4.41}$$

where α_t is a positive decreasing step size. In order to avoid singularities of $\mathbf{A}_t$ in LSTD(λ) or $\mathbf{B}_t$ in λ-LSPE, one can add a diagonal matrix $\delta\mathbf{I}$ to either $\mathbf{A}_t$ or $\mathbf{B}_t$, where δ is a positive real. We note that it is often useful to write the variables $\mathbf{A}_t$, $\mathbf{B}_t$, and $\mathbf{b}_t$ above as recursions:

$$\begin{aligned}
\mathbf{A}_t &= \mathbf{A}_{t-1} + \mathbf{z}_t\left(\phi(s_t) - \gamma\phi(s_{t+1})\right)^T \\
\mathbf{b}_t &= \mathbf{b}_{t-1} + \mathbf{z}_t g(s_t, s_{t+1}) \\
\mathbf{B}_t &= \mathbf{B}_t + \phi(s_t)\phi(s_t)^T
\end{aligned} \tag{3.4.42}$$

where $\mathbf{A}_{-1} = \mathbf{B}_{-1} = \mathbf{0}$ (or $\mathbf{A}_{-1} = \mathbf{B}_{-1} = \delta\mathbf{I}$) and $\mathbf{b}_{-1} = 0$.

The above formulations of LSTD(λ) and λ-LSPE are derived from the least squares approximation:

$$\theta_{t+1}^{LS} = \arg\min_{\theta} \sum_{n=0}^{t} \left(\phi(s_n)^T\theta - \left[\phi(s_n)^T\theta_t + \sum_{m=n}^{t} (\gamma\lambda)^{m-n} d\left(s_m, s_{m+1}; \hat{\theta}\right) \right] \right)^2 \tag{3.4.43}$$

The trajectories of temporal differences are used to build a value function estimation (right hand term in the above), which is then approximated in a least squares sense using linear approximation $\phi(s_n)^T\theta$. The above two algorithms differ in the choice of $\hat{\theta}$. LSTD(λ) minimizes the inconsistency of the new estimation θ_{t+1}^{LS}, by forcing $\hat{\theta} = \theta_{t+1}^{LS}$. The λ-LSPE

algorithm uses the current estimate $\hat{\theta} = \theta_t$ in (3.4.43). It then makes an adjustment in the direction of θ_{t+1}^{LS}, setting $\theta_{t+1} = \theta_t + \alpha_t \left(\theta_{t+1}^{LS} - \theta_t \right)$, using some positive stepsize α_t.

A detailed derivation of LSTD(λ) is presented in [BB99] and [B02], and convergence with probability one is proven in [NB03]. The complexity of each step of LSTD(λ) is $O(K^3)$. An efficient implementation of $O(K^2)$ complexity per step, named RLS-TD(λ), is derived in [XHH02]. The derivation of λ-LSPE(λ) is presented in [NB03], with its straightforward implementation of $O(K^3)$ complexity per step, and its efficient implementation of $O(K^2)$ complexity. An extension to its convergence properties for a constant step size is given in [BBN03].

3.4.5 Comparison of TD and Least Squares methods

In the subsection, we briefly compare the TD learning algorithms above. We start by stating that all the algorithms above converge with probability one to the same fixed point, which depends on λ. Furthermore, the efficient implementation of the λ-LSPE and RLS-TD(λ) are comparable in terms of computational load, memory requirements and asymptotic convergence rates. Their intermediate convergence behavior differs. The LS based algorithms are more data efficient than TD(λ) in the sense that they converge significantly faster in terms of the number of iterations required to achieve a certain error level. Furthermore, TD(λ) requires the specification of a learning stepsize which significantly effects the convergence rate. The drawback of the LS methods is the computational load per iteration and memory requirements, which are $O(K^2)$, as opposed to the complexity of TD(λ) which is $O(K)$. This renders LS methods impractical when the number of features is large.

3.5 Algebraic Preliminaries

In this section we give a few definitions which will serve us in subsequent sections.

Definition 1: A matrix $\mathbf{A}$ is called an *M-matrix* if $\mathbf{A}_{ii} > 0, \mathbf{A}_{ij} \le 0 \quad \forall i \ne j$, and $\mathbf{A}^{-1} > 0$ element-wise.

Definition 2: A matrix **A** is said to be *diagonally dominant* if $\mathbf{A}_{ii} > \sum_{j \neq i} \left| \mathbf{A}_{ij} \right|$.

Definition 3: An $n \times n$ real matrix **A** (not necessarily symmetric) is said to be *positive definite* if for all nonzero vectors $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$.

We use the notation $\mathbf{A} > 0$ ($\mathbf{A} < 0$) to denote that **A** is symmetric positive (negative) definite. It is well known that a real symmetric matrix **A**, is positive definite if and only if (iff) all its eigenvalues have positive real parts, i.e. $\mathrm{Re}\left(eig\left(\mathbf{A} \right) \right) > 0$. The following lemma does not assume **A** is symmetric.

Lemma 3.1 *Let* **A** *be a positive definite matrix (not necessarily symmetric). Then all the eigenvalues of* **A** *have a positive real part.*

Proof Denote by μ and $\mathbf{v} = \mathbf{x} + i\mathbf{y}$ any eigenvalue of **A** and its corresponding eigenvector, where $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$ are the real and imaginary parts of **v** respectively. By direct calculation we obtain

$$\mathbf{v}^{\dagger} \mathbf{A} \mathbf{v} = \left(\mathbf{x} - i\mathbf{y} \right)^T \mathbf{A} \left(\mathbf{x} + i\mathbf{y} \right) = \left(\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{y}^T \mathbf{A} \mathbf{y} \right) + i \left(\mathbf{x}^T \mathbf{A} \mathbf{y} - \mathbf{y}^T \mathbf{A} \mathbf{x} \right) \tag{3.5.44}$$

where † denotes a conjugate transpose. Since $\mathbf{v} \neq \mathbf{0}$ is an eigenvector of **A** we have

$$\mathbf{v}^{\dagger} \mathbf{A} \mathbf{v} = \mathbf{v}^{\dagger} \mu \mathbf{v} = \mu \left\| \mathbf{v} \right\|^2 \tag{3.5.45}$$

Using the last two equations we obtain

$$\mu = \left[\left(\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{y}^T \mathbf{A} \mathbf{y} \right) + i \left(\mathbf{x}^T \mathbf{A} \mathbf{y} - \mathbf{y}^T \mathbf{A} \mathbf{x} \right) \right] \Big/ \left\| \mathbf{v} \right\|^2 \tag{3.5.46}$$

Since **A** is positive definite it follows that $\mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$, $\mathbf{y}^T \mathbf{A} \mathbf{y} \geq 0$, with at least one of them being a strict inequality since $\mathbf{v} \neq \mathbf{0}$, and therefore

$$\mathrm{Real}\left(\mu \right) = \left(\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{y}^T \mathbf{A} \mathbf{y} \right) / \left\| \mathbf{v} \right\| > 0 . \tag{3.5.47}$$

■

Definition 4: A real matrix **A** is said to be *Hurwitz-stable* if all its eigenvalues have negative real parts, i.e. $\mathrm{Re}\left(eig\left(\mathbf{A}\right)\right) < 0$.

Theorem: A real matrix **A** is stable iff there exists a matrix $\mathbf{L} > 0$ such that

$$\mathbf{A}^T\mathbf{L} + \mathbf{L}\mathbf{A} < \mathbf{0}. \tag{3.5.48}$$

3.6 Algebraic Multigrid Review

3.6.1 What is Algebraic Multigrid?

Algebraic Multigrid (AMG) is a family of multilevel numerical methods for accelerating the iterative solution of large linear equation systems of the form

$$\mathbf{A}\mathbf{u} = \mathbf{b} \tag{3.6.49}$$

AMG convergence analysis assumes **A** is a symmetric positive definite matrix (*SPD*). We are not aware of such analysis under different assumptions. In practice, it is often further assumed that **A** is an M-matrix or is strictly diagonally dominant. For AMG to be computationally efficient, it is further required that **A** is sparse.

AMG accelerates standard iterative methods, by eliminating error components at various resolution scales. A standard iterative method is used on the original *fine grid,* which results in reducing one type of error components. The complementary type, which is slow to converge under iteration, is named *algebraically smooth*. Smooth errors are approximated on a *coarse grid* and eliminated after interpolation. Since *iteration* and *coarse grid correction* are effective for complementary types of error components, applying them alternately results in a synergetic fast reduction of any error component.

We now make this description more concrete. Consider an iterative method of the form

$$\mathbf{u} := \mathbf{u} + \mathbf{Q}^{-1}\left(\mathbf{b} - \mathbf{A}\mathbf{u}\right) \tag{3.6.50}$$

where the matrix **Q** determines the iteration method. Common iteration methods are Richardson for which $\mathbf{Q} = \mathbf{I}$, ω-Jacobi for which **Q** is the diagonal of **A** divided by a

damping factor $\omega \in [0,1]$, and Gauss-Seidel for which **Q** is the lower triangular part of **A** including the diagonal. Notice that this form of iteration has much in common with Dynamic Programming methods presented in section 3.2. The policy evaluation method in (3.2.15) is in fact a Richardson iteration for **A**=**I**-**P**. Applying the iteration (3.6.50) multiplies the error vector $\mathbf{e} \equiv \mathbf{u}^* - \mathbf{u}$, where $\mathbf{u}^*$ is the exact solution of (3.6.49), by the matrix

$$\mathbf{S} = \mathbf{I} - \mathbf{Q}^{-1}\mathbf{A}, \tag{3.6.51}$$

(see straightforward derivation in [S01]). The matrix **S** is known as the *smoothing operator*. Denote the eigenvalue, eigenvector pairs of **S** by $\mu_k, \mathbf{v}_k$, respectively. Error vectors spanned by eigenvectors with near zero eigenvectors will be effectively reduced by the iteration. However, an error vector aligned with $\mathbf{v}_k$ for which $|\mu_k| \approx 1$ will converge very slowly. The latter is said to be *algebraically smooth*. Any initial error can be represented as a linear combination of $\mathbf{v}_k$. As we apply (3.6.50) repeatedly, non-smooth components decay more rapidly and smooth components remain, making the resulting error more and more smooth. Eventually, the smooth error may be approximated as a linear combination of eigenvectors $\mathbf{v}_k$ for which $|\mu_k| \approx 1$, while neglecting components for which $|\mu_k|$ is close to zero. For a numerical demonstration of differences in convergence rates we refer the reader to section 5.1.2. AMG methods eliminate the remaining smooth error by a coarse grid correction step, which applies an additive correction of the form

$$\mathbf{u} := \mathbf{u} + \hat{\mathbf{e}}. \tag{3.6.52}$$

The error is approximated by

$$\hat{\mathbf{e}} \approx \mathbf{I}_1^0 \mathbf{v}. \tag{3.6.53}$$

where $\mathbf{I}_1^0$ is an $n_0 \times n_1$ matrix ($n_1 < n_0$) and $\mathbf{v}$ is some $n_1 \times 1$ vector. AMG builds $\mathbf{I}_1^0$ so that its columns roughly span the subspace spanned by n_1 eigenvectors for which $|\mu_k|$ is closest to 1. Smooth errors are therefore well represented by (3.6.53). Plugging in (3.6.52) into (3.6.49) generates an over-determined equation for $\mathbf{v}$

$$\mathbf{A}\left(\mathbf{u}+\mathbf{I}_1^0\mathbf{v}\right)\approx\mathbf{b}\,. \tag{3.6.54}$$

Let $\mathbf{I}_0^1$ be some $n_0\times n_1$ (projection) matrix. A solution for $\mathbf{v}$ is obtained by left multiplying the above equation by $\mathbf{I}_0^1$

$$\left(\mathbf{I}_0^1\mathbf{A}\mathbf{I}_1^0\right)\mathbf{v}=\mathbf{I}_0^1\left(\mathbf{b}-\mathbf{A}\mathbf{u}\right). \tag{3.6.55}$$

We can now solve for $\mathbf{v}$, and apply the correction in (3.6.52), completing the coarse correction step. The basic idea described above is extended in AMG to multiple grid levels.

We now define some terms and indexing that will serve us in describing the AMG routine below. The term *grid* in AMG refers to a vector of unknowns, whose elements are virtual *points*. In classical AMG, a *coarse grid* is a vector, whose elements are a subset of elements of the *fine grid* vector. AMG forms a hierarchy of grids. We index these levels by $\ell\in\{0,1,\ldots,\ell_{\max}\}$, where $\ell=0$ is the original finest grid and $\ell_{\max}$ is the coarsest. We denote the number unknown variables (points) in grid level ℓ by n_ℓ, with n_0 the dimensionality of $\mathbf{u}$. Since coarse grids are contained within fine grids we have $n_{\ell_{\max}}<n_{\ell_{\max-1}}\ldots<n_1<n_0$.

3.6.2 General AMG Routine

AMG algorithms have two phases: a preliminary *setup* phase which is applied once to define level operators, and a *solve* phase which is applied iteratively until a satisfactory precision is obtained. A general AMG routine which includes these two phases is given in Algorithm 3.1 below. The setup phase builds three types of operators: $n_\ell\times n_\ell$ equation matrices $\mathbf{A}_\ell$, interpolators $\mathbf{I}_{\ell+1}^{\ell}$ which are $n_\ell\times n_{\ell+1}$ rectangular matrices, and restrictors $\mathbf{I}_{\ell}^{\ell+1}$ which are $n_{\ell+1}\times n_\ell$ matrices. The restrictors are used to project the residual error vector resulting from smoothing iterations at level ℓ onto a coarse grid at level $\ell+1$. Interpolators are used to transfer a vector that approximates the smooth error on a coarse grid $\ell+1$ back to the fine grid ℓ. Since the *setup* phase is rather technical, we delay its presentation to section 3.7.

The solve phase implements a standard Multigrid algorithm based on the operators defined in the setup phase. The *V-cycle* scheme outlined in Algorithm 3.1 is a standard implementation of the solve phase, with other schemes given in subsequent sections. The V-cycle scheme is a recursive implementation of a three step scheme used to approximate the solution of an equation system $\mathbf{A}_\ell \mathbf{u}_\ell = \mathbf{b}_\ell$ at level ℓ. The first step uses standard iteration to smooth the error vector (step 3). In the second step, the resultant smooth error is approximated on a coarse grid via a reduced equation system known as the *residual equations*, which take the form $\mathbf{A}_{\ell+1}\mathbf{u}_{\ell+1} = \mathbf{b}_{\ell+1}$. The coarse approximation of the smooth error is interpolated to the fine grid and used to make an additive correction (step 4). In the third step, additional iterations are used to reduce interpolation errors (step 5). At the coarsest grid, the number of unknowns is small enough to allow a direct solution of $\mathbf{A}_{\ell_{\max}} \mathbf{u}_{\ell_{\max}} = \mathbf{b}_{\ell_{\max}}$ (step 1). The schematics of the V-cycle are presented in Figure 3.1.

Algorithm 3.1: General AMG routine

Setup phase:

Denote $\mathbf{A}_0 := \mathbf{A}$ for the finest grid, $\ell = 0$. We denote its dimensions by $n_0 \times n_0$.

For $\ell = 0$ to $\ell_{\max} - 1$ do:

1. Given $\mathbf{A}_\ell$, build the $n_\ell \times n_{\ell+1}$ interpolator, $\mathbf{I}_{\ell+1}^{\ell}$ by applying the following steps:
   1.1. Select a subset of $n_{\ell+1}$ ($< n_\ell$) coarse grid variables (see section 3.6.4).
   1.2. Calculate the elements $\mathbf{I}_{\ell+1}^{\ell}$ which serve as interpolation weights (section 3.6.3).
2. Build the $n_{\ell+1} \times n_\ell$ restrictor $\mathbf{I}_{\ell}^{\ell+1}$, based on the interpolator. Common choices are the transpose $\mathbf{I}_{\ell}^{\ell+1} = \left(\mathbf{I}_{\ell+1}^{\ell}\right)^T$, and the pseudo inverse $\mathbf{I}_{\ell}^{\ell+1} = \left[\left(\mathbf{I}_{\ell+1}^{\ell}\right)^T \mathbf{I}_{\ell+1}^{\ell}\right]^{-1} \left(\mathbf{I}_{\ell+1}^{\ell}\right)^T$.
3. Build the $n_{\ell+1} \times n_{\ell+1}$ coarse-grid operator as $\mathbf{A}_{\ell+1} := \mathbf{I}_{\ell}^{\ell+1} \mathbf{A}_\ell \mathbf{I}_{\ell+1}^{\ell}$

Solve phase: V-cycle

V-cycle$(\mathbf{b}_\ell, \ell)$

If on the coarsest grid, i.e. $\ell = \ell_{\max}$

1. Directly solve and return $\mathbf{u}_{\ell_{\max}} := \mathbf{A}_{\ell_{\max}}^{-1} \mathbf{b}_{\ell_{\max}}$

Otherwise do steps 2-5

2. Initialize level correction $\mathbf{u}_\ell := \mathbf{0}$
3. Apply $\beta_{pre} \geq 0$ smoothing iterations $\mathbf{u}_\ell := \mathbf{u}_\ell + \mathbf{Q}_\ell^{-1}(\mathbf{b}_\ell - \mathbf{A}_\ell \mathbf{u}_\ell)$
4. Coarse grid correction step:
   - 4.1. Restrict the residual to the coarser grid $\mathbf{b}_{\ell+1} := \mathbf{I}_\ell^{\ell+1}(\mathbf{b}_\ell - \mathbf{A}_\ell \mathbf{u}_\ell)$
   - 4.2. Recursive call $\mathbf{v}_{\ell+1} := \text{V-cycle}(\mathbf{b}_{\ell+1}, \ell+1)$
   - 4.3. Interpolate the error to the current grid $\hat{\mathbf{e}}_\ell := \mathbf{I}_{\ell+1}^{\ell} \mathbf{v}_{\ell+1}$.
   - 4.4. Correct the current grid estimation $\mathbf{u}_\ell := \mathbf{u}_\ell + \hat{\mathbf{e}}_\ell$
5. Apply $\beta_{post} \geq 0$ smoothing iterations $\mathbf{u}_\ell := \mathbf{u}_\ell + \mathbf{Q}_\ell^{-1}(\mathbf{b}_\ell - \mathbf{A}_\ell \mathbf{u}_\ell)$

Return $\mathbf{u}_\ell$

The operator defined as $\mathbf{A}_{\ell+1} := \mathbf{I}_\ell^{\ell+1} \mathbf{A}_\ell \mathbf{I}_{\ell+1}^{\ell}$ is named a *Galerkin operator*. At least one of the algorithm parameters β_{pre} and β_{post} is non-zero, and they are often chosen such that $\beta_{pre} + \beta_{post} \in \{2,3,4\}$,. Further detail on the steps above is available in [MC87], [TOS01] and [VH99].

3.6.3 Grid Refinement

Grid Refinement is another multilevel scheme we address in our work. It uses a "one way" coarse-to-fine approach, by interpolating the coarse grid solution to the fine grid. The schematics are presented in Figure 3.1. This scheme enables to obtain an application dependent initial solution, at a low computational cost. Unfortunately, this solution usually contains algebraically smooth error components that decay slowly under iterations.

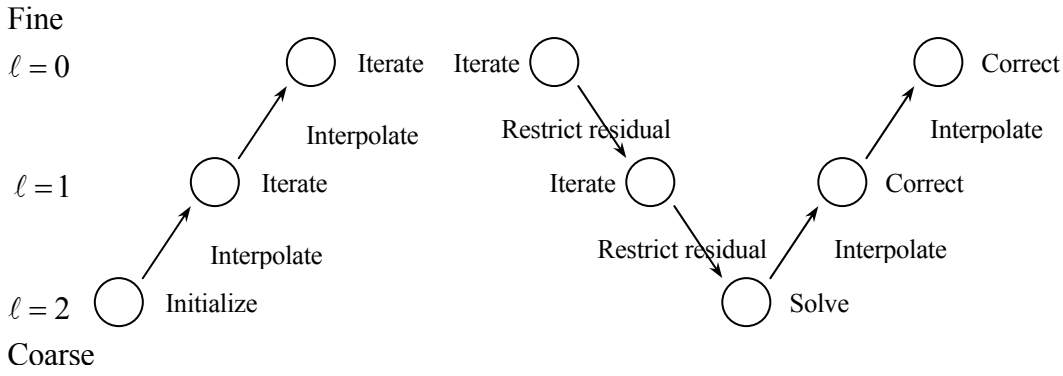


Figure 3.1: Schematics of Grid Refinement (left) and V-cycle (right)

3.6.4 Full Multigrid

Full Multigrid (FMG) combines Grid Refinement and Multigrid. A Grid Refinement step is used in the initialization of every level, followed by a V-cycle in order to eliminate smooth errors generated during interpolation. The schematics of FMG are shown in Figure 3.2.

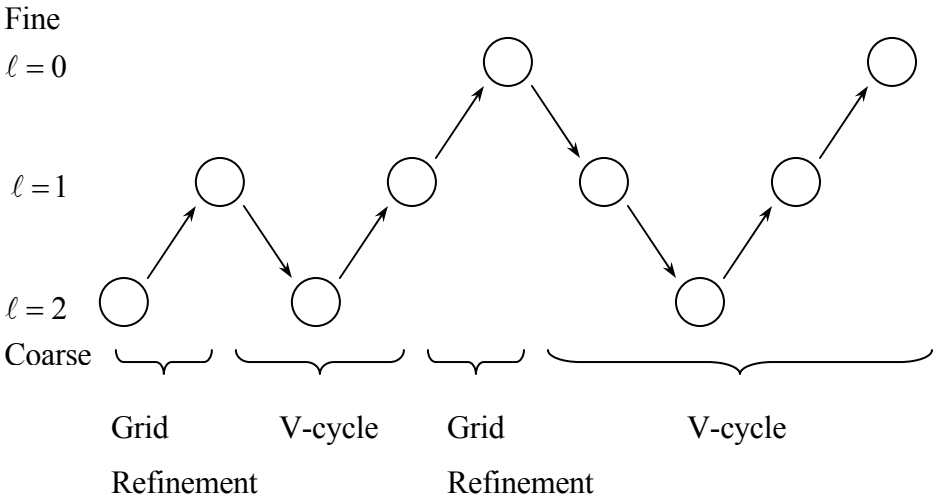


Figure 3.2: Schematics of a Full Multigrid cycle

3.6.5 Convergence Issues

The convergence of Algorithm 3.1 and similar AMG algorithms is well established when the matrix **A** is symmetric positive definite [S01] [MC87]. Theorem A.2.2 and Corollary A.2.1 in [S01] immediately imply the following theorem.

Theorem *Let* $\mathbf{A}_0$ *be SPD and let any full rank interpolation* $\mathbf{I}_{\ell+1}^{\ell}$ *be given. Let* $\mathbf{A}_\ell$ *be defined as Galerkin operators* $\mathbf{A}_{\ell+1} = \mathbf{I}_{\ell}^{\ell+1}\mathbf{A}_\ell\mathbf{I}_{\ell+1}^{\ell}$. *Then V-cycle given in Algorithm 3.1 with Gauss-Seidel iterations used in steps 3 and 5 converges to the solution of* $\mathbf{A}_0\mathbf{u} = \mathbf{b}_0$.

Proof The proof in a nutshell is based on the definition of two operators:

smoothing operator $$\mathbf{S}_\ell \equiv \mathbf{I} - \mathbf{Q}_\ell^{-1}\mathbf{A}_\ell \tag{3.6.56}$$

coarse-grid correction $$\mathbf{K}_{\ell,\ell+1} \equiv \mathbf{I} - \mathbf{I}_{\ell+1}^{\ell}\mathbf{A}_{\ell+1}^{-1}\mathbf{I}_{\ell}^{\ell+1}\mathbf{A}_\ell . \tag{3.6.57}$$

Each iteration in steps 3 or 5 modifies the error $\mathbf{e}_\ell = \mathbf{u}_\ell^* - \mathbf{u}_\ell$, where $\mathbf{u}_\ell^*$ is the solution of $\mathbf{A}_\ell\mathbf{u}_\ell = \mathbf{b}_\ell$, according to $\mathbf{e}_\ell := \mathbf{S}_\ell\mathbf{e}_\ell$; and applying the correction step (step 4) at level $\ell_{\max} - 1$ changes the error according to $\mathbf{e}_\ell := \mathbf{K}_{\ell,\ell+1}\mathbf{e}_\ell$. The proof follows by showing that $\mathbf{K}_{\ell,\ell+1}$ is an orthogonal projector with respect to the inner product $\langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{A}} = \mathbf{u}^T\mathbf{A}\mathbf{v}$, and therefore cannot increase the induced norm of the error $\|\mathbf{K}_{\ell,\ell+1}\mathbf{e}_\ell\|_{\mathbf{A}} \le \|\mathbf{e}_\ell\|_{\mathbf{A}}$, while $\mathbf{S}_\ell$ for Gauss-Seidel is a contraction with respect to the same norm. It follows that the composite operator $\mathbf{S}_\ell^{\beta_{post}}\mathbf{K}_{\ell,\ell+1}\mathbf{S}_\ell^{\beta_{post}}$ is a contraction. This immediately implies convergence of a two level scheme, with an extension to multi-level by induction.

■

A few remarks are in order. The SPD property of **A** is required to define the induced norm $\|\cdot\|_{\mathbf{A}}$, which is used to show contraction properties. There are other iteration methods, despite Gauss-Seidel, that guarantee convergence, such as ω-Jacobi (see [S01]). Asymptotic convergence is guaranteed whatever choice of interpolators or restrictors, as long as the interpolators are full rank. However, the rate of convergence heavily depends on a proper

choice of interpolators. This dependence is presented in [MC87] as a bound on the number of iterations required to reduce the error norm by some factor, assuming some bound on the interpolation error. We note that this measure of convergence rate does not consider computational load. An interesting result is that under certain technical conditions the convergence rate in [MC87] (Theorem 3.1) is independent of the size of **A**. We do not specify these bounds here since they are not suited for the explicit construction of realistic AMG processes. We only mention that the interpolator should approximate well smooth errors, while non-smooth errors are less important. Finally, convergence is also guaranteed when replacing the exact solution on the coarsest grid with an approximate one, if it fulfills a tolerance condition given in [S01]. This result gives some sense of the algorithm robustness.

3.6.6 Practical Issues

While convergence guarantees give a sense of robustness, the main objective of AMG is computational efficiency. This efficiency is measured in terms of computational effort required to reach a certain error level. To gain efficiency, interpolators should be computationally cheap as well as good approximators of smooth errors. In addition, interpolators should be constructed in a way to keep $\mathbf{A}_\ell$ sparse in order to keep coarse grid computations cheap. Such interpolators are constructed based on heuristics. The construction of interpolators will be discussed in section 3.7.3. Another parameter which effects the efficiency is the number of iterations performed per correction step. Applying too few iterations will deteriorate convergence rate due to large interpolation errors, while too many will be computationally inefficient. Usually, a total of 2-5 iterations per level are a good default choice.

3.7 Technical notes on the *setup* phase

3.7.1 A few words on motivation

The goal of the setup phase is to generate inter-level operators such that convergence rate during the solve phase will be high, i.e. the computational effort required to obtain certain accuracy should be small as possible. To do so, smooth errors should be well approximated

on the coarse grid by interpolation (see (3.6.53)). On the other hand, it is not required to approximate non-smooth errors since they are smoothed out by iterations. In fact, the main idea behind the setup phase is to compromise between interpolation accuracy which dominates convergence speed (in terms of the number of iterations) and the numerical work together with memory requirements which rely on keeping the interpolator sparse and strongly reducing the number of unknown variables in the coarse level.

To motivate the general approach, taken in the construction of interpolators, we show a property that is approximately satisfied by smooth errors. We refer the interested reader to a more rigor and comprehensive discussion in [S01]. Consider a vector $\mathbf{v}$, such that

$$\mathbf{A}\mathbf{v} \approx \mathbf{0}. \tag{3.7.58}$$

Applying the smoothing operator (3.6.51) to $\mathbf{v}$ approximately gives

$$\mathbf{S}\mathbf{v} = (\mathbf{I} - \mathbf{Q}^{-1}\mathbf{A})\mathbf{v} = \mathbf{v} - \mathbf{Q}^{-1}\left(\mathbf{A}\mathbf{v}\right) \approx 1 \cdot \mathbf{v}, \tag{3.7.59}$$

which means that an error $\mathbf{v}$ is slow to converge under iteration. In other words, algebraically smooth errors satisfy

$$\sum_j \mathbf{A}_{ij} v_j \approx 0, \quad \forall i. \tag{3.7.60}$$

There are many ways to build an interpolation operator that may serve to approximate $\mathbf{v}$ given it is smooth. A major advantage of AMG lies in the fully automation of the interpolator creation. A very common approach, and also the one we take here, builds the interpolation by splitting the variables of the elements v_j of the unknown vector $\mathbf{v}$ into two disjoint groups: variables named *C-variables* which are evaluated on the coarse grid, and variables named *F-variables* that are interpolated from the *C-variables*. The interpolation of *F-variables* takes the form

$$v_i = \sum_{j \in C} w_{ij} v_j, \quad i \in F, \tag{3.7.61}$$

where w_{ij} are interpolation weights, chosen so that (3.7.60) is approximately satisfied. In order for the interpolator to be computationally cheap the number of non-zero weights is

kept small as possible. The splitting procedure is known as the *coarsening* procedure. The coarsening procedure and two methods we use for choosing weights are described in sections 3.7.4 and 3.7.3 respectively. Together they form an automatic procedure which takes a matrix $\mathbf{A}_\ell$ and constructs a computationally cheap interpolator $\mathbf{I}_{\ell+1}^{\ell}$, which approximates smooth errors at level ℓ via (3.6.53).

The *setup* phase in Algorithm 3.1 applies this heuristic procedure to build $\mathbf{I}_{\ell+1}^{\ell}$. The specific choice of the restrictor is not as important to guarantee fast convergence and is often simply taken as the transpose of $\mathbf{I}_{\ell+1}^{\ell}$. However, it is very important to take the coarse-grid operator as a *Galerkin* operator. For this special choice the coarse grid correction operator $\mathbf{K}_{\ell,\ell+1}$ is a projection operator (see (3.6.57)), which is required in proving convergence properties of the scheme [S01]. In general, various setup procedures differ mainly in the interpolator construction procedure, and only slightly in the choice the restrictor. Though technically different, the objective in the interpolator construction is similar: build $\mathbf{I}_{\ell+1}^{\ell}$ such that any algebraically smooth errors with respect to $\mathbf{A}_\ell$ (or **S**) is well approximated as $\mathbf{I}_{\ell+1}^{\ell}\mathbf{v}_{\ell+1}$, while keeping computational costs low.

3.7.2 Notations and Definitions

In the description of the *setup* phase, we use the following definitions and notations: With respect to (3.6.49), the i^{th} variable in **u,** denoted u_i, is said to *depend* on variable u_j, or u_j to *influence* u_i if $\mathbf{A}_{ij} \neq 0$. Variable u_i is said to *strongly depend* on u_j if

$$-\mathbf{A}_{ij} \geq \varepsilon \max_{k \neq i} \{-\mathbf{A}_{ik}\} \tag{3.7.62}$$

where $\varepsilon \in [0,1]$ is a threshold parameter. Throughout this dissertation, we chose $\varepsilon = 0.15$. Variable u_j is said to *strongly influence* u_i if u_i *strongly depends* on u_j.

The variables of every level ℓ are split using the *coarsening* process into two disjoint (coarse and fine) subsets: C denotes the set of *C-variables* evaluated on level $\ell+1$, and F denotes the complementary set of *F-variables*, which are interpolated from *C-variables*.

Given a C-F splitting, each variable that *influences* $u_i \in F$ is assigned to one of three disjoint groups: $C_i \subset C$ denotes the group of *C-variables* used to interpolate u_i, $D_i^s \subset F$ the *F-variables* that *strongly influence* u_i, and D_i^w the remaining variables that are said to *weakly influence* u_i.

3.7.3 Construction of Interpolators

In this dissertation we use a common two-step procedure to build $\mathbf{I}_{\ell+1}^{\ell}$. First, we apply standard *coarsening* as described in [TOS01] section A.7.1.1. *Coarsening*, described below, splits the variables of $\mathbf{u}_\ell$ into C-variables that construct $\mathbf{v}_{\ell+1}$, and F-variables, which are interpolated from $\mathbf{v}_{\ell+1}$. For now, assume such a split is given. We would like to build an interplator, i.e. choose interpolation weights, such that smooth errors are well approximated. Without lose of generality, we assume that C-variables are located at the upper part of $\mathbf{u}_\ell$, followed by F-variables. The interpolator we seek takes the form

$$u_i = \left(I_{\ell+1}^{\ell}\mathbf{v}_\ell\right)_i = \begin{cases} v_i & \text{if } u_i \in C \\ \sum_{j \in C_i} w_{ij} v_j & \text{if } u_i \in F \end{cases} \tag{3.7.63}$$

The interpolation weights w_{ij} for the F-variables are defined in the second step. We use two interpolation methods. The strict aggregation interpolation presented in (3.7.64), aims to generate an extremely cheap interpolator by assigning the value of a single *C-variable* to each $u_i \in F$. In order to approximate (3.7.60) as well as possible, u_i is set to the *C-variable* that mostly *influences* it. If the most influencing variable is not unique, the first one in lexical order is chosen.

$$\textit{Strict aggr.:}\quad w_{ij} = \begin{cases} 1 & \text{for a single } j \in C_i, \text{ for which } \left(-\mathbf{A}_{ij}\right) = \max_{k \in C}\left(-\mathbf{A}_{ik}\right) \\ 0 & \text{otherwise} \end{cases} \tag{3.7.64}$$

The second weighting scheme we use named Ruge-Stüben is presented in (3.7.69). It better approximates (3.7.60) at a price of being computationaly more expensive. The weight derivation is obtained by first rewritting (3.7.60) as

$$\mathbf{A}_{ii}\nu_i + \sum_{n\in D_i^w} \mathbf{A}_{in}\nu_n \approx -\sum_{j\in C_i} \mathbf{A}_{ij}\nu_j - \sum_{m\in D_i^s} \mathbf{A}_{im}\nu_m \ . \tag{3.7.65}$$

Note that ν_j are not known for $j \in D_i^s$ or $j \in D_i^w$. They are therefore approximated respectively by:

$$\begin{aligned} \nu_m &\approx \sum_{j\in C_i} \frac{\mathbf{A}_{mj}}{\sum_{k\in C_i} \mathbf{A}_{mk}} \nu_j, \qquad && \forall m \in D_i^s \\ \nu_n &\approx \nu_i, && \forall n \in D_i^w \end{aligned} \tag{3.7.66}$$

where strongly connected *F-varialbes* are replaced with a weighted average of C-variables, and weakly connected variables are simply replaced with the interpolated variable ν_i. By substitution in (3.7.65) and changing the summation order we have

$$\left(\mathbf{A}_{ii} + \sum_{n\in D_i^w} \mathbf{A}_{in}\right)\nu_i \approx -\sum_{j\in C_i}\sum_{m\in D_i^s} \frac{\mathbf{A}_{im}\mathbf{A}_{mj}}{\sum_{k\in C_i} \mathbf{A}_{mk}} \nu_j - \sum_{j\in C_i} \mathbf{A}_{ij}\nu_j \ , \tag{3.7.67}$$

leading to

$$\nu_i \approx -\frac{\sum_{j\in C_i}\left[\mathbf{A}_{ij} + \sum_{m\in D_i^s} \frac{\mathbf{A}_{im}\mathbf{A}_{mj}}{\sum_{k\in C_i} \mathbf{A}_{mk}}\right]}{\left(\mathbf{A}_{ii} + \sum_{n\in D_i^w} \mathbf{A}_{in}\right)} \nu_j \ . \tag{3.7.68}$$

The weights in the Ruge-Stüben method are given by

$$\textit{Ruge- Stüben:} \qquad w_{ij} = -\frac{\mathbf{A}_{ij} + \sum_{m \in D_i^s} \left(\frac{\mathbf{A}_{im}\mathbf{A}_{mj}}{\sum_{k \in C_i} \mathbf{A}_{mk}} \right)}{\mathbf{A}_{ii} + \sum_{n \in D_i^w} \mathbf{A}_{in}}. \tag{3.7.69}$$

For a detailed disucssion and justification of Ruge-Stüben weigting see [TOS01] section A.7.2.1.

3.7.4 Coarsening

As described before, the *coarsening* procedure splits the variables u_i into two disjoint groups: C-variables and F-variables. The previous section described methods for choosing interpolation weights such that smooth errors are approximated by interpolation (3.6.53). This was achieved by neglecting and approximating terms in (3.7.60). In order to minimize the error caused by these approximations we would like to interpolate each *F-variable*, u_i, from as many variables as possible, or at least from those that strongly influence it. This will require to assign all the stongly influencing variables to set C. A weaker requirement is to be able to approximate well any variable that strongly influences u_i, whether we choose it to be a C-variable or an F-variable. On the other hand, computational efficiency requires to keep the number of C-variables minimal. The above considerations are formally represented by the following criterions [VH99]:

(C1) For every $u_i \in F$, any variable, u_j that strongly influences u_i should either be in C or share a strongly influencing *C-variable*, i.e. $C_i \cap C_j \neq \varnothing$.

(C2) C should be a maximal subset with the property that no *C-variable* strongly depends on another *C-variable*.

In this dissertation we follow the coarsesning procedure described in [TOS01]. It is a heuristic procedure aimed to satisfy both criterions. Whenever these criterions contrast, the second one is relaxed. Coarsening is done in three sweeps:

(1) Define strong dependencies and influences between variables.

(2) Split into C-variables and F-variables to satisfy (C2) by applying Algorithm 3.2.

(3) Scan F-variables to make sure (C1) is satisfied. This is done using Algorithm 3.3.

For further detail see [TOS01].

Algorithm 3.2: Split into C-variables and F-variables

1. Assign to each variable a value equal to the number of variables that it strongly influences. Denote this value by λ_i.
2. Choose a variable with the maximal λ_i, as a *C-variable*. Denote this variable by u_c.
3. Set all unassigned variables that strongly depend on u_c, to be F-variables. Denote these variables by u_{fj}.
4. Increase λ_i of each unassigned variables, u_i, by one for every u_{fj} variable it strongly depends on. Decrease λ_i by one if u_c strongly depends on u_i.

Repeat steps 2-4 until all variables are either C-variables or F-variables.

Algorithm 3.3: Enforce (C1)

1. For each *F-variable*, u_i, do:

   1.1. For every $u_j \in D_i^s$ do

   i) If C_i and C_j share a *C-variable* then continue to the next u_j in step 1.1.

   ii) If for any other $u_k \in D_i^s / u_j$, C_i and C_k do not share a *C-variable*
   then reassign u_i to be a *C-variable*,
   otherwise reassign u_j to be a *C-variable*.

   iii) Continue to the next u_i in step 1.

Chapter 4

AMG for Policy Evaluation and Iteration

In this chapter we discuss the application of AMG to the problems of policy evaluation and policy optimization, when the model of the MDP is fully known. Throughout this chapter, we assume the MDP is *irreducible, aperiodic* and *finite*, with an *infinite-horizon discounted cost*. For AMG to be efficient, we further assume that $\mathbf{P}$ is sparse. This assumption is reasonable for many complex real life problems in which the number of states reached from any state $s \in \mathcal{S}$ within a single step is limited.

Starting with policy evaluation, we show a deficiency of standard iterative methods that limits their convergence rate. We then apply AMG as a "black box" solver to accelerate convergence for the PE problem. Next, we review the validity of assumptions made in AMG theory to prove convergence, in the context of policy evaluation. All assumptions except for symmetry are validated for the finest grid. Since symmetry is needed to guarantee convergence, we show how the problem can be reformulated to impose symmetry. For reasons discussed later on, convergence may be faster in practice when applying AMG as a "black box" solver, ignoring the symmetry assumption violation. Next, we discuss a special kind of interpolator-restrictor setting for AMG, which forms aggregates in the state space. We prove that this special setting enables to write coarse grid equations as the solution of MDPs of reduced state space sizes. General interpolator-restrictor settings do not support this property. This property is convenient when one wishes to apply extra processing in low resolution, such as a coarse policy search. We conclude this chapter by addressing the application of AMG to the problem of policy optimization, specifically by policy evaluation combined with policy improvement.

4.1 AMG for Policy Evaluation

In this section, we address the problem of evaluating a fixed stationary policy π. To do so we need to solve the linear equation defined in (3.2.8). We rewrite this as

$$\left(\mathbf{I}-\gamma\mathbf{P}^{\pi}\right)\mathbf{v}^{\mathrm{p}}=\mathbf{g}^{\mathrm{p}}, \tag{4.1.1}$$

and define

$$\mathbf{A}=\mathbf{I}-\gamma\mathbf{P}^{\pi} \tag{4.1.2}$$

$$\mathbf{b}=\mathbf{g}^{\pi}. \tag{4.1.3}$$

For notational ease, we drop the dependence notation on the fixed policy π, writing $\mathbf{P}$ and $\mathbf{g}$ instead of $\mathbf{P}^{\pi}$ and $\mathbf{g}^{\pi}$ respectively.

4.1.1 A deficiency of standard iterative methods

To understand why AMG should prove beneficial to policy evaluation, we demonstrate when standard iterative methods have poor convergence rates. We remind that standard iterative methods take the form (see section 3.6.1 for definitions)

$$\mathbf{u}:=\mathbf{u}+\mathbf{Q}^{-1}\left(\mathbf{b}-\mathbf{A}\mathbf{u}\right). \tag{4.1.4}$$

This iteration propagates the error according to

$$\mathbf{e}:=\left(\mathbf{I}-\mathbf{Q}^{-1}\mathbf{A}\right)\mathbf{e}. \tag{4.1.5}$$

If the operator $\mathbf{I}-\mathbf{Q}^{-1}\mathbf{A}$ has little effect on some error vector $\mathbf{e}$, convergence will be slow. Such error vectors are said to be *algebraically smooth*. For example, if $\mathbf{e}$ is an eigenvector of $\mathbf{I}-\mathbf{Q}^{-1}\mathbf{A}$ with an eigenvalue close to 1, convergence rate will be poor. To make this more concrete in the context of policy evaluation, we demonstrate this for the Richardson method, i.e. $\mathbf{Q}=\mathbf{I}$. Substituting (4.1.2) into (4.1.5) we obtain

$$\mathbf{e}:=\gamma\mathbf{P}\mathbf{e}. \tag{4.1.6}$$

The error after n iterations, denoted $\mathbf{e}_n$, is therefore

$$\mathbf{e}_n = (\gamma \mathbf{P})^n \mathbf{e}_0 . \tag{4.1.7}$$

Since $\mathbf{P}$ is a probability matrix it has an eigenvalue equal to 1, with a corresponding eigenvector $\underline{\mathbf{1}} = [1 \quad 1 \quad \cdots \quad 1]^T$. Assuming the initial error is $\mathbf{e}_0 = \underline{\mathbf{1}}$, the error after n iterations is

$$\mathbf{e}_n = (\gamma \mathbf{P})^n \mathbf{e}_0 = \gamma^n \mathbf{P}^n \underline{\mathbf{1}} = \gamma^n \underline{\mathbf{1}} = \gamma^n \mathbf{e}_0 . \tag{4.1.8}$$

Often γ is chosen close to one (e.g. 0.999), in order to account for long-term revenues. Consequently, convergence will be very slow. Though demonstrated for the Richardson method only, similar results apply to all iteration methods discussed in this dissertation. The reason is that iterations make local adjustments and therefore global corrections propagate slowly. Convergence is slowed down when the error vector lies in the space of smooth errors. The error can then be well approximated as done in AMG algorithms.

4.1.2 AMG as a "black box" solver for PE

Treating AMG as a “black box” solver, we may directly apply it to solve (4.1.1) and obtain the value function, $\mathbf{v}$. Algorithm 4.1 below is the application of the general AMG scheme (Algorithm 3.1) to policy evaluation. In step 2, the V-cycle takes the residual error as an argument and the result is used to correct $\mathbf{v}_n$. This modification is equivalent to initializing V-cycle with the current estimate on the finest grid instead of zero, i.e. $\mathbf{u}_0 := \mathbf{v}_n$ (step 2 in Algorithm 3.1).

Algorithm 4.1: AMG for Policy Evaluation

1.	Apply the *Setup phase* with respect to $\mathbf{A} = \mathbf{I} - \gamma \mathbf{P}$
2.	Starting with some initial guess, usually $\mathbf{v}_0 = \mathbf{0}$, iteratively apply,
	2.1. Calculate residual $\mathbf{res}_n := \mathbf{g} - (\mathbf{I} - \gamma \mathbf{P})\mathbf{v}_n$
	2.2. Correct $\mathbf{v}_{n+1} := \mathbf{v}_n + \text{V-cycle}(\mathbf{res}_n, \ell = 0)$

4.1.3 AMG for policy evaluation - review of assumptions

AMG theory ensures the convergence of Algorithm 4.1 if $\mathbf{A}$ is symmetric and positive definite (SPD). Efficiency relies on **A** being sparse. Interpolator construction often further assumes that **A** is an M-matrix (see definition in section 3.5). We now address the validity of these assumptions in the context of policy evaluation. First, we show in Lemma 4.2 that for the finest grid, **A** as defined in (4.1.2) is indeed an M-matrix.

Lemma 4.2 *If* $\mathbf{P}$ *is a transition probability matrix and* $\gamma \in [0,1)$*, then* **A** *as defined in (4.1.2) is an M-matrix with a strictly dominant diagonal.*

Proof First, we show that **A** has a positive diagonal and non-positive off diagonal elements. The matrix **P** is a transition probability matrix, i.e. $\mathbf{P}_{ij} \in [0,1]$ and $\sum_j \mathbf{P}_{ij} = 1$. Since $\mathbf{P}_{ij} \in [0,1]$ and $\gamma \in [0,1)$ we have

$$0 \le \gamma \mathbf{P}_{ij} < 1. \tag{4.1.9}$$

Applying this to the definition of $\mathbf{A}$ in (4.1.2) we obtain

$$\mathbf{A}_{ii} = 1 - \gamma \mathbf{P}_{ij} \ge 1 - \gamma > 0 \tag{4.1.10}$$

$$\mathbf{A}_{ij} = -\gamma \mathbf{P}_{ij} \le 0 \quad \forall i \ne j. \tag{4.1.11}$$

Next, we show that **A** is strictly diagonally dominant. We verify that **A** has a positive row sum

$$\sum_j \mathbf{A}_{ij} = 1-\gamma\sum_j \mathbf{P}_{ij} = 1-\gamma > 0 \quad \forall i, \tag{4.1.12}$$

where we used the equality $\sum_j \mathbf{P}_{ij} = 1$. By moving all off diagonal elements to the right hand side, and using (4.1.11) we prove strict diagonal dominance

$$\mathbf{A}_{ii} > -\sum_{j\neq i} \mathbf{A}_{ij} = \sum_{j\neq i} \left|\mathbf{A}_{ij}\right| \qquad \forall i. \tag{4.1.13}$$

It is well known that a strictly diagonally dominant matrix satisfying (4.1.10) and (4.1.11) is an M-matrix [W99].

■

As stated in section 3.6.5, if the M-matrix $\mathbf{A}$ is symmetric and positive definite (SPD), then convergence is guaranteed, and choosing $\mathbf{A}_{\ell+1} := \mathbf{I}_{\ell}^{\ell+1}\mathbf{A}_{\ell}\mathbf{I}_{\ell+1}^{\ell}$ (Galerkin operators) ensures that $\mathbf{A}_{\ell}$ are SPD at all levels. Unfortunately, there is no reason to assume that $\mathbf{A}$ is symmetric, since $\mathbf{P}$ seldom is. Consequently, direct application of AMG to (4.1.1) may diverge. More over, for general interpolators, even if $\mathbf{A}_0$ is an M-matrix, $\mathbf{A}_{\ell}$ on coarser levels are not generally so. Ensuring that $\mathbf{A}_{\ell}$ is an M-matrix in the non-symmetric case can be done for the special case of state aggregation, to be discussed in 4.1.5. It goes without saying that $\mathbf{A}$ is sparse, since $\mathbf{P}$ is sparse as stated at the beginning of this chapter. We conclude by noting that even though the symmetry assumption is generally invalid, our experiments suggest that in practice direct application of AMG often converges and may be advantageous in terms of convergence rate (see chapter 6).

4.1.4 Imposing symmetry

We now present two approaches to circumvent the violation of the symmetry assumption, in order to guarantee convergence. One is to multiply both sides of (4.1.1) by $\mathbf{A}^T$, to obtain

$$\left(\mathbf{A}^T\mathbf{A}\right)\mathbf{v} = \left(\mathbf{A}^T\mathbf{b}\right). \tag{4.1.14}$$

Equation (4.1.14) takes the form of a least-squares problem. Since $\mathbf{A}$ is an M-matrix and therefore non-singular, $\mathbf{A}^T$ is non-singular and (4.1.14) has the same solution as the original

equation (4.1.1). The matrix $\mathbf{A}^T\mathbf{A}$ is positive definiteness since $\mathbf{u}^T\left(\mathbf{A}^T\mathbf{A}\right)\mathbf{u} = \|\mathbf{A}\mathbf{u}\|^2 \geq 0$, and symmetric. We can therefore apply AMG to (4.1.14) while guaranteeing convergence to the true value function. We denote $\tilde{\mathbf{A}} = \mathbf{A}^T\mathbf{A}$ and $\tilde{\mathbf{b}} = \mathbf{A}^T\mathbf{b}$, and apply AMG to solve

$$\tilde{\mathbf{A}}\mathbf{v} = \tilde{\mathbf{b}}. \tag{4.1.15}$$

Another way to circumvent the symmetry violation is to use a *Kaczmarz* form. Since $\mathbf{A}^T$ is invertible we may define a vector $\mathbf{x}$ as the solution of $\mathbf{A}^T\mathbf{x} = \mathbf{v}$. Substituting into (4.1.1) we obtain

$$\left(\mathbf{A}\mathbf{A}^T\right)\mathbf{x} = \mathbf{b}. \tag{4.1.16}$$

The matrix $\mathbf{A}\mathbf{A}^T$ is symmetric and positive definite. Again, we can apply AMG to solve (4.1.16) with guaranteed convergence. The benefit of these approaches is the applicability of AMG convergence theory. Unfortunately, as demonstrated in chapter 6, the rate of convergence for both approaches is inferior in practice to that of directly applying AMG to (4.1.1). This result is not surprising since the condition numbers of $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$ are greater than the condition number of $\mathbf{A}$, rendering the solution of (4.1.14) and (4.1.16) harder than (4.1.1). Furthermore, $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$ are less sparse than $\mathbf{A}$ which increases the computational load. In general they are not *M-matrices*, even if $\mathbf{A}$ is. We note that the M-matrix property is useful in the construction of interpolators and refer the interested reader to [TOS01].

4.1.5 Preservation of the Markov chain interpretation under strict state aggregation

When using AMG to evaluate a policy, one may wish to preserve a *Markov chain (MC) interpretation* on the coarse grids. We say that $\mathbf{A}_\ell$ has an *MC interpretation* if it can be written as

$$\mathbf{A}_\ell = \mathbf{I}_\ell - \gamma\mathbf{P}_\ell, \tag{4.1.17}$$

where $\mathbf{I}_\ell$ is the identity matrix of dimension matching level ℓ, and $\mathbf{P}_\ell$ is a transition probability matrix. We say that the MC interpretation is *preserved* if given that $\mathbf{A}_\ell$ has a MC interpretation, it follows that the *Galerkin* operator $\mathbf{A}_{\ell+1} = \mathbf{I}_\ell^{\ell+1}\mathbf{A}_\ell\mathbf{I}_{\ell+1}^{\ell}$ has a MC interpretation. When MC interpretation is preserved, coarse grids may be viewed as Markov chains defined on a reduced state space. This is beneficial since by Lemma 4.2, this results in $\mathbf{A}_\ell$ being *M-matrices* at all levels, making interpolation more stable. This also enables to define a policy search at a low resolution. This variant is left for future work.

We use two different methods to construct the inter-level operators. The method proposed by Ruge and Stüben [S01] has proven effective in practice. Unfortunately, it does not preserve MC interpretation or the M-matrix property over grid levels. As an example, consider a Markov chain defined by $\mathbf{P}_0$ below with a discount factor of $\gamma = 0.9$. We generate the Ruge-Stüben interpolator $\mathbf{I}_1^0$ using (3.7.69) and $\varepsilon = 0.15$ in (3.7.62), and take the restrictor as its transpose.

$$\mathbf{P}_0 = \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{I}_1^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.4286 & 0.4286 \\ 0.3 & 0 & 0.6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.1.18}$$

Writing down the *Galerkin* operator we obtain the following coarse grid equations

$$\mathbf{A}_1 = \mathbf{I}_0^1\mathbf{A}_0\mathbf{I}_1^0 = \mathbf{I}_0^1\left(\mathbf{I} - \gamma\mathbf{P}_0\right)\mathbf{I}_1^0 = \begin{bmatrix} 0.55 & 0 & \underline{0.0514} & -0.4886 \\ 0 & 1 & -0.3857 & -0.3857 \\ -0.3 & -0.3 & 0.8029 & -0.0771 \\ -0.27 & 0 & -0.54 & 1 \end{bmatrix}, \tag{4.1.19}$$

where $\mathbf{A}_1$ is not an M-matrix since it has a positive off diagonal element (underlined above). Since $\mathbf{I} - \gamma\mathbf{P}_1$ is an M-matrix for any $\gamma \in [0,1), \mathbf{P}_1 > 0$, there are no such $\gamma, \mathbf{P}_1$ satisfying $\mathbf{A}_1 = \mathbf{I} - \gamma\mathbf{P}_1$, therefore $\mathbf{A}_1$ has no MC interpretation.

The second method uses *state aggregation*. Here the state space is partitioned into disjoint groups, denoted $\mathcal{G}_k, k = 1, 2, \ldots, K$. Each group, called an *aggregated state*, is treated as an abstract state. The interpolator is defined

$$\begin{aligned} \left(\mathbf{I}_{\ell+1}^{\ell}\right)_{ik} &= 1 \text{ if } i \in \mathcal{G}_k \\ &= 0 \text{ otherwise} \end{aligned}, \tag{4.1.20}$$

and the restrictor is defined as its pseudo-inverse

$$\mathbf{I}_{\ell}^{\ell+1} = \left[\left(\mathbf{I}_{\ell+1}^{\ell}\right)^T \mathbf{I}_{\ell+1}^{\ell}\right]^{-1} \left(\mathbf{I}_{\ell+1}^{\ell}\right)^T. \tag{4.1.21}$$

Lemma 4.3 *State aggregation using (4.1.20) and (4.1.21) preserves MC interpretation.*

Proof Follows directly from Lemma 1 in [BC89]. The lemma states that if $\mathbf{I}_{\ell+1}^{\ell}$ and $\mathbf{I}_{\ell}^{\ell+1}$ satisfy (4.1.20) and (4.1.21), then

a) $\mathbf{I}_{\ell}^{\ell+1}\mathbf{I}_{\ell+1}^{\ell} = \mathbf{I}_{\ell+1}$

b) $\mathbf{P}_{\ell+1} \triangleq \mathbf{I}_{\ell}^{\ell+1}\mathbf{P}_{\ell}\mathbf{I}_{\ell+1}^{\ell}$ is a transition probability matrix.

Writing the *Galerkin* operator, we verify the MC interpretation,

$$\mathbf{A}_{\ell+1} = \mathbf{I}_{\ell}^{\ell+1}\mathbf{A}_{\ell}\mathbf{I}_{\ell+1}^{\ell} = \mathbf{I}_{\ell}^{\ell+1}\left(\mathbf{I} - \gamma\mathbf{P}_{\ell}\right)\mathbf{I}_{\ell+1}^{\ell} = \mathbf{I}_{\ell}^{\ell+1}\mathbf{I}_{\ell+1}^{\ell} - \gamma\mathbf{I}_{\ell}^{\ell+1}\mathbf{P}_{\ell}\mathbf{I}_{\ell+1}^{\ell} = \mathbf{I}_{\ell+1} - \gamma\mathbf{P}_{\ell+1}. \tag{4.1.22}$$

The last equality follows directly from the lemma. Since $\mathbf{P}_{\ell+1}$ is a transition probability matrix, $\mathbf{A}_{\ell+1}$ has a MC interpretation.

■

As noted in [BC89], $\left(\mathbf{I}_{\ell+1}^{\ell}\right)^T \mathbf{I}_{\ell+1}^{\ell}$ is a diagonal matrix with the k^{h} diagonal entry equal to the number of elements in $\mathcal{G}_k$, denoted $\left|\mathcal{G}_k\right|$. The inverse is therefore easy to compute, and

$$\left[\mathbf{P}_{\ell+1}\right]_{ij} = \frac{1}{\left|\mathcal{G}_i\right|} \sum_{k \in \mathcal{G}_i, m \in \mathcal{G}_j} \left[\mathbf{P}_{\ell}\right]_{km}. \tag{4.1.23}$$

In [BC89] aggregates are formed for a two level scheme by grouping states of similar residuals. We take a different approach by gradually grouping states based on state adjacency. A subset of representative states (C-variables) is selected using the coarsening procedure described in section 3.7.4. Each remaining state i is aggregated with a representative state j, to which transition probability is highest, i.e. $j = \arg\max_k p\left(s' = k \mid s = i\right)$. On the first coarse level, only adjacent states are grouped. On the next coarse level, adjacent states relative to coarse grid are grouped, grouping states that are two steps apart on the fine grid.

For completeness, we present Algorithm 4.2 as a version of the V-cycle function presented in Algorithm 3.1, specialized to the case of state aggregation. Interpolators, restrictors and transition probability matrices are defined according to (4.1.20), (4.1.21) and (4.1.23) respectively. In steps 3 and 8, updating the value for all states simultaneously results in a Richardson iteration. If values are updated one by one using the most up-to-date values, we obtain an iteration method similar to Gauss-Seidel. The latter is preferred for efficiency reasons.

Algorithm 4.2: AMG for Policy Evaluation - detailed scheme

V-cycle$\left(\mathbf{g}_\ell, \ell\right)$

If on the coarsest grid, i.e. $\ell = \ell_{\max}$

1. Directly Solve and Return $\mathbf{v}_{\ell_{\max}}$: $\mathbf{v}_{\ell_{\max}} = \mathbf{g}_{\ell_{\max}} + \gamma \mathbf{P}_{\ell_{\max}} \mathbf{v}_{\ell_{\max}}$

Otherwise do steps 2-8

2. Initialize $\mathbf{v}_\ell := \mathbf{0}$
3. Apply β_{pre} times, $\forall s \in \mathcal{S}$ $v_\ell(s) := g_\ell(s) + \gamma \sum_{s' \in \mathcal{S}_\ell} p_\ell(s' \mid s) \mathbf{v}_\ell(s')$
4. Restrict the residual to the coarser grid $\mathbf{g}_{\ell+1} := \mathbf{I}_\ell^{\ell+1}\left(\mathbf{g}_\ell - \left(\mathbf{I}_\ell - \gamma \mathbf{P}_\ell\right)\mathbf{v}_\ell\right)$
5. Recursive call $\mathbf{v}_{\ell+1} := \text{V-cycle}\left(\mathbf{g}_{\ell+1}, \ell+1\right)$
6. Interpolate the error to the current grid $\hat{\mathbf{e}}_\ell := \mathbf{I}_{\ell+1}^{\ell} \mathbf{v}_{\ell+1}$.
7. Correct the current grid estimation $\mathbf{v}_\ell := \mathbf{v}_\ell + \hat{\mathbf{e}}_\ell$
8. Apply β_{post} times, $\forall s \in \mathcal{S}$ $v_\ell(s) := g_\ell(s) + \gamma \sum_{s' \in \mathcal{S}_\ell} p_\ell(s' \mid s) v_\ell(s')$

Return $\mathbf{v}_\ell$

4.2 AMG for Modified Policy Iteration

In previous sections, we discussed the usage of AMG to evaluate a given policy, in order to accelerate standard iterative methods. We now turn to the problem of finding optimal policies. Based on the policy iteration algorithm, we replace the costly policy evaluation step with an efficient AMG algorithm. Algorithm 4.3 alternates between finding the greedy policy with respect to the current value estimate (step 1), and an AMG policy evaluation step (steps 2,3). This algorithm may be regarded as an efficient implementation of a modified policy iteration algorithm. As in Algorithm 4.1, using the residual error in step 3 is equivalent to initializing AMG on the finest grid with the current estimate of $\mathbf{v}_n$.

Algorithm 4.3: AMG for Policy Iteration

Initialize $\mathbf{v}_0 = \mathbf{0}$

1. Derive the greedy policy* $\pi_n^g(s) := \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} p(s' \mid s,a)\left[g(s,s',a) + \gamma v_n(s') \right]$

2. Apply the *Setup phase* with respect to $\mathbf{A} = \mathbf{I} - \gamma \mathbf{P}^{\pi_n^g}$

3. Policy evaluation step:

   3.1. Calculate residual $\mathbf{res}_n := \mathbf{g}^{\pi_n^g} - \left(\mathbf{I} - \gamma \mathbf{P}^{\pi_n^g} \right) \mathbf{v}_n$

   3.2. Correct $\mathbf{v}_{n+1} := \mathbf{v}_n + \text{V-cycle}\left(\mathbf{res}_n, \ell = 0 \right)$

* An additional iteration can be gained at no extra computational cost by assigning the value of the maximum to $\mathbf{v}_{n+1}$.

Chapter 5

Multigrid Temporal Difference Algorithms

In the previous chapter, we discussed the application of AMG to the problem of evaluating a fixed policy, and to the problem of searching for an optimal policy, when the MDP model is fully known. In this chapter, we extend these ideas to the context of learning, when the model is not explicitly available. We propose Multigrid versions of Temporal Difference learning algorithms for policy evaluation. Throughout this chapter we assume that the Markov chains involved have either finite or infinite countable state spaces, and consider the *infinite-horizon discounted cost*. For convergence analysis purposes we further assume the Markov chains are *irreducible* and *aperiodic*.

This dissertation focuses on TD versions of the *solve* phase of AMG. We provide a short discussion about the *setup* phase in section 5.4. In all other sections we assume that the required inter-level operators are available beforehand. We denote by $\mathbf{I}_{\ell+1}^{\ell}$ the interpolator from level $\ell+1$ to level ℓ. The restrictor from level ℓ to level $\ell+1$ is defined as $\mathbf{I}_{\ell}^{\ell+1}=\left(\mathbf{I}_{\ell+1}^{\ell}\right)^{T}$. Assigning, as before, the index zero to the finest level, we denote the feature vector of this level as $\phi_0(s)=\phi(s)$. We define the feature vector at levels $\ell \in 1,2,\ldots,\ell_{\max}$ recursively by

$$\phi_{\ell+1}(s)^{T}=\phi_{\ell}(s)^{T}\mathbf{I}_{\ell+1}^{\ell} \tag{5.1.1}$$

or in vector form

$$\mathbf{\Phi}_{\ell+1}=\mathbf{\Phi}_{\ell}\mathbf{I}_{\ell+1}^{\ell}. \tag{5.1.2}$$

Note that the basis functions of a coarse grid are linear combination of basis functions of finer grids. Consequently, the set of basis functions from all levels, $\{\phi_\ell(s)\}_{\ell=0,\ldots,\ell_{\max}}$, forms a redundant set. We will make use of this property in subsequent convergence analysis.

In the first section we analyze the dynamics of the TD(λ) algorithm with linear function approximation, to reveal a deficiency that limits its asymptotic convergence rate. This analysis shows that TD(λ) may be regarded as a stochastic variant of a Richardson smoother, and therefore convergence slows down as the error becomes algebraically smooth. Motivated by the analogy to the known model case, we propose a level-based Multigrid TD learning algorithm in section 5.2. We prove the convergence with probability 1 of each level separately. Next, we propose a Multigrid TD algorithm that simultaneously updates parameter vectors at all resolution level, making it more data efficient. We discuss two special cases: one is equivalent to the level-based algorithm; the other has a special structure that enables to prove its convergence. In the last section we briefly discuss the *setup* phase in the context of learning.

5.1 Analysis of TD(λ) dynamics

5.1.1 Analytic derivation

We remind that the TD(λ) update rule

$$\theta_t = \theta_{t-1} + \alpha_t \mathbf{z}_t \left(g(s_t, s_{t+1}) - \left(\phi(s_t) - \gamma\phi(s_{t+1})\right)^T \theta_{t-1} \right) \tag{5.1.3}$$

$$\mathbf{z}_t = \lambda\gamma\mathbf{z}_{t-1} + \phi(s_t). \tag{5.1.4}$$

The stochastic dynamics of the TD(λ) algorithm are expressed in [BBN03] as

$$\theta_t = \theta_{t-1} + \alpha_t \left(\mathbf{b} - \mathbf{A}\theta_{t-1}\right) + \alpha_t \left(\Xi_t \theta_{t-1} + \xi_t\right) \tag{5.1.5}$$

where

$$\mathbf{A} = \mathbf{\Phi}^T \mathbf{D} \mathbf{M}_\lambda \left(\mathbf{I} - \gamma\mathbf{P}\right)\mathbf{\Phi} \tag{5.1.6}$$

$$\mathbf{b} = \mathbf{\Phi}^T \mathbf{D} \mathbf{M}_\lambda \mathbf{g} \tag{5.1.7}$$

$$\mathbf{\Xi}_t = \mathbf{A} - \mathbf{z}_t \left(\phi(s_t) - \gamma \phi(s_{t+1}) \right)^T \tag{5.1.8}$$

$$\xi_t = \mathbf{b} - \mathbf{z}_t \mathbf{g}(s_t, s_{t+1}) . \tag{5.1.9}$$

The matrices $\mathbf{\Phi}, \mathbf{P}, \mathbf{D}, \mathbf{M}_\lambda$ and the vector $\mathbf{g}$ are defined in chapter 3 in equations (3.2.10)-(3.2.12) and (3.4.32)-(3.4.33). $\mathbf{\Xi}_t$ and ξ_t are random sequences of matrices and vectors respectively that have an asymptotic mean equal to zero. Convergence of the means to zero is faster for Markov chains with a short mixing times and small $\gamma\lambda$ value.

Our analysis is focused on the convergence of the mean[1] of the parameter vector $\mathbb{E}\{\theta_t\}$. Taking the expectation of (5.1.5) and denoting $\overline{\theta}_t = \mathbb{E}\{\theta_t\}$, we approximate the dynamics of the mean by

$$\overline{\theta}_{t+1} = \overline{\theta}_t + \alpha_t \left(\mathbf{b} - \mathbf{A} \overline{\theta}_t \right), \tag{5.1.10}$$

where we neglected the expectation of the third term in (5.1.5). This approximation is reasonable when the means of $\mathbf{\Xi}_t$ and ξ_t are close to zero, and the correlation between $\mathbf{\Xi}_t$ and θ_t is negligible. This correlation is small if θ_t varies slowly.

In [TVR97] it is proven that if the TD(λ) assumption set in section 3.4.3 is valid, TD(λ) converges with probability 1 to the unique vector

$$\theta^* = \mathbf{A}^{-1} \mathbf{b} . \tag{5.1.11}$$

Denoting the deterministic mean error

$$\overline{\mathbf{e}}_t = \theta^* - \overline{\theta}_t , \tag{5.1.12}$$

and subtracting both sides of (5.1.10) from θ^*, we obtain the error propagation equations

[1] A similar analysis for both mean and variance yet restricted to the case of no function approximation is given in [SD98].

$$\bar{\mathbf{e}}_t = \bar{\mathbf{e}}_{t-1} - \alpha_t \mathbf{A} \bar{\mathbf{e}}_{t-1}. \tag{5.1.13}$$

We denote the eigenvalue-eigenvector pairs of **A,** by $\mu_k, \mathbf{v}_k$ for $k = 1, \ldots, N$, respectively. In [TVR97] it is shown that **A** is positive definite and bounded, and therefore $\text{Real}(\mu_k) \geq 0$ and bounded (see Lemma 3.1). If the error vector equals $\mathbf{v}_k$, applying the update rule multiplies the error vector by $1 - \alpha_t \mu_k$. Since $\alpha_t > 0$ is a sequence decreasing to zero, and $\text{Real}(\mu_k) \geq 0$ and bounded, there exists some T, such that for all $t \geq T$,

$$0 < \text{Real}(1 - \alpha_t \mu_k) \leq 1. \tag{5.1.14}$$

Addressing real valued μ_k, for a large enough t the factor $1 - \alpha_t \mu_k$ is smaller for large values of μ_k. Therefore, the asymptotic convergence rate is fast if μ_k is large and very slow for $\mu_k \approx 0$. If $\bar{\mathbf{e}}_t$ is a linear combination of the eigenvectors, convergence would be relatively fast at first, due to effective reduction of components corresponding to large eigenvalues. Eventually, convergence rate deteriorates when resultant error components correspond to small eigenvalues. The smallest eigenvalue dominates the asymptotic convergence rate. While this discussion assumed μ_k is real, it applies to complex eigenvalues as well. Since $\bar{\mathbf{e}}_t$ is real, its linear combination representation contains complex eigenvectors and their conjugates with equal weights. The imaginary part is thus canceled.

Our analysis shows that TD(λ) effectively reduces one type of error components, while leaving another type relatively unchanged. Adopting terminology from standard AMG we name the latter an *algebraically smooth error*. This calls for a Multigrid scheme, which approximates and eliminates the *smooth error* on a coarse grid. This is the subject of the next section.

For future reference, we use the ODE approach [BT95] to approximate (5.1.13) for large enough t. This approach enables to avoid the dependence on the learning step α_t. We define a continuous time process $\bar{\mathbf{e}}(\tau)$, by rescaling time $\tau_t = \sum_{k=0}^{t-1} \alpha_k$ and interpolating $\bar{\mathbf{e}}_t$ linearly

between τ_t and τ_{t+1}. The ODE that governs the asymptotic convergence of the mean error is then given by

$$\frac{d}{d\tau}\bar{\mathbf{e}}_\tau = -\mathbf{A}\bar{\mathbf{e}}_\tau . \tag{5.1.15}$$

We note that if the error equals to $\mathbf{v}_k$, the error derivative is $-\mu_k$, and as before, asymptotic convergence is slow for $\mu_k \approx 0$.

5.1.2 Empirical demonstration

To make the derivations more concrete we offer the following example. We use the 1-D random walk Markov chain defined in Figure 5.1 with *N*=64 states. Detail on this chain is given later on in section 6.1.1.

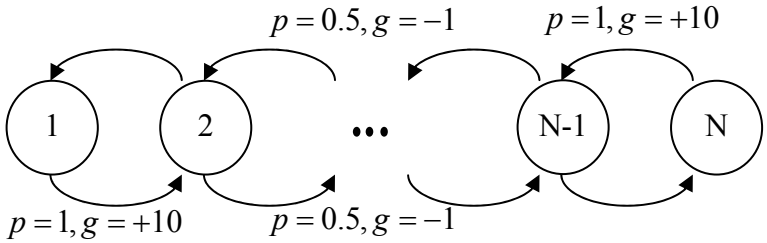


Figure 5.1: 1-D random walk Markov chain

Figure 5.2 shows convergence curves of the norm of the residual, for TD(0) when initialized from different initial errors. In the first three cases, the initial error is an eigenvector $\mathbf{v}_k$ of $\mathbf{A}$. As predicted by the analysis above, convergence is exponential with a rate of $-|\mu|_k$. In the bottom row the initial error is an average of the three eigenvectors, normalized to unit length. It shows that when the initial error is a combination of eigenvectors, convergence deteriorates with a limit rate that equals the smallest eigenvalue.

Table 5.1 demonstrates this effect numerically. Denote the initial error by $\mathbf{e}_0$ and the error after applying TD(0) by $\mathbf{e}_T$. For each of the four cases above, the table shows the correlation between $\mathbf{e}_0$ and each of the eigenvectors, and the correlation between $\mathbf{e}_T$ and $\mathbf{v}_k$, $k = 1,2,3$. We observe the following:

- When $\mathbf{e}_0 = \mathbf{v}_k$ is an eigenvector of **A**, TD(0) attenuates the error without changing its orientation. As seen from Figure 5.2 the attenuation rate is determined by μ_k.
- When $\mathbf{e}_0 = \frac{1}{c}\sum_i \mathbf{v}_i$, TD(0) attenuates each component differently, making the error more and more algebraically smooth, ultimately making it lie parallel to the eigenvector with the smallest eigenvector.

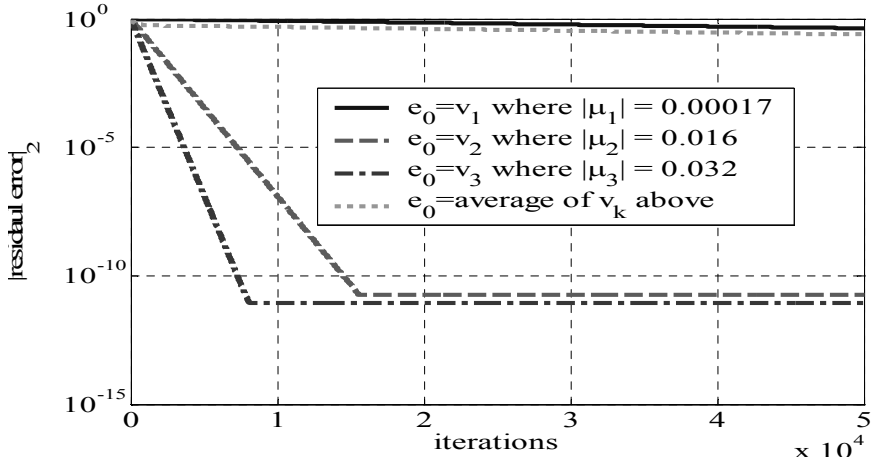


Figure 5.2: Convergence curves for TD(0), starting from different initial values

Table 5.1: Correlations between $\mathbf{e}_t$ and $\mathbf{v}_k$ before and after applying TD(0) when initialized from different initial errors

	Initial correlation $\rho(\mathbf{e}_0, \mathbf{v}_k)$			Final correlation $\rho(\mathbf{e}_T, \mathbf{v}_k)$		
	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$
$\mathbf{e}_0 = \mathbf{v}_1$	1	0	0	1	-2E-14	-9E-15
$\mathbf{e}_0 = \mathbf{v}_2$	0	1	0	-3E-3	1	-2E-4
$\mathbf{e}_0 = \mathbf{v}_3$	0	0	1	-3E-3	1E-3	0.92
$\mathbf{e}_0 = \frac{1}{c}\sum_i \mathbf{v}_i$	0.58	0.58	0.58	1	9E-14	3E-15

5.2 A level-based Multigrid TD algorithm

5.2.1 The Main Algorithm

In this section we propose a level-based Multigrid TD algorithm (*LB-MGTD(λ)*) that accelerates the convergence of TD(λ). As seen in the previous section, TD(λ) may be regarded as a stochastic on-line smoother with the dynamics of a Richardson iteration (5.1.10), solving

$$\mathbf{A}\theta = \mathbf{b} \tag{5.2.16}$$

where $\mathbf{A}$ and $\mathbf{b}$ are previously defined in (5.1.6)-(5.1.7). As such, the convergence rate of TD(λ) is expected to deteriorate as the error becomes algebraically smooth. Based on the Multigrid approach we would like to apply a correction of the form

$$\theta_\ell := \theta_\ell + \mathbf{I}_{\ell+1}^{\ell}\vartheta_{\ell+1} \tag{5.2.17}$$

where $\vartheta_{\ell+1}$ is the solution of the residual equation

$$\left(\mathbf{I}_{\ell}^{\ell+1}\mathbf{A}_\ell\mathbf{I}_{\ell+1}^{\ell}\right)\vartheta_{\ell+1} = \mathbf{I}_{\ell}^{\ell+1}\left(\mathbf{b}_\ell - \mathbf{A}_\ell\theta_\ell\right). \tag{5.2.18}$$

In the learning context, this equation cannot be solved directly nor represented explicitly since the model of the Markov chain is not known. $\mathbf{A}_\ell$ and $\mathbf{b}_\ell$ are unavailable, because $\mathbf{P}, \mathbf{D}$ and $\mathbf{g}$ are not known. The level-based Multigrid TD algorithm we propose in Algorithm 5.1, avoids this obstacle by applying a TD(λ) variant that converges to the solution of (5.2.18). The idea is to replace the unavailable explicit variables with measured samples: for example, the vector $\mathbf{g}$ with scalar rewards $g\left(s_t, s_{t+1}\right)$ that are sampled over time, and the residual error $\mathbf{b}_\ell - \mathbf{A}_\ell\theta_\ell$ by a sampled version. A description of the algorithm is given in the next sub-section. The algorithm is an on-line implementation of the V-cycle scheme presented in section 3.6.2 (see schematics in Figure 3.1). The algorithm starts with some initial guess at the fine level $\ell = 0$ (usually $\theta_0 = \mathbf{0}$), and iteratively applies the function

$$\theta_0 := \text{LB-MGTD-V-cycle}\left(\theta_0, 0\right). \tag{5.2.19}$$

The approximated value function at the end of a complete cycle is given by

$$v(s) = \phi_0(s)^T \theta_0 . \tag{5.2.20}$$

In Algorithm 5.1 below and henceforth we omit the iteration indexing *t* for ease of notation, and write $\mathbf{z}_\ell$, r_ℓ and d_ℓ instead of $\mathbf{z}_{\ell,t}$, $r_{\ell,t}$ and $d_{\ell,t}$, respectively. We use a dual notation $\vartheta_{\ell+1} = \theta_{\ell+1}$ to denote the correction, in order to distinguish between $\vartheta_{\ell+1}$ a fixed result vector when the active level is ℓ, and $\theta_{\ell+1}$ a varying vector when in level $\ell+1$.

Algorithm 5.1: Level-based Multigrid-TD(λ) at level ℓ (LB-MGTD)

LB-MGTD-V-cycle$\left(\vartheta_\ell, \ell, \theta_0^-, \theta_1^-, \ldots, \theta_{\ell-1}^-\right)$

1. **Initialize level correction** $\theta_\ell := \vartheta_\ell, \mathbf{z}_\ell := \mathbf{0}$

2. **Pre-iterate at level ℓ with residual rewards**:

   2.1. Observe the transition $s_t \to s_{t+1}$ and the reward g_t at time t.

   2.2. Update the eligibility traces $\mathbf{z}_\ell := \lambda\gamma\mathbf{z}_\ell + \phi_\ell\left(s_t\right)$

   2.3. Sample the residual $r_\ell := g_t - \sum_{m=0}^{\ell-1}\left(\phi_m\left(s_t\right) - \gamma\phi_m\left(s_{t+1}\right)\right)^T \theta_m^-$

   2.4. Calculate the temporal difference $d_\ell = r_\ell - \left(\phi_\ell\left(s_t\right) - \gamma\phi_\ell\left(s_{t+1}\right)\right)^T \theta_\ell$

   2.5. Update $\theta_\ell := \theta_\ell + \alpha_{\ell,t}\mathbf{z}_\ell d_\ell$

   2.6. If the switching criterion is met then continue, otherwise repeat from 2.1.

If operating at the coarsest grid, i.e. $\ell = \ell_{\max}$, return $\theta_{\ell_{\max}}$. Otherwise continue with steps 3-4.

3. **Apply coarse grid correction**:

   3.1 Set $\theta_\ell^- := \theta_\ell$

   3.2. Recursive call $\vartheta_{\ell+1} :=$ LB-MGTD-V-cycle$\left(\vartheta_{\ell+1} = \mathbf{0}, \ell+1, \theta_0^-, \theta_1^-, \ldots, \theta_\ell^-\right)$

   3.3. Correction using the interpolated error $\theta_\ell := \theta_\ell^- + \mathbf{I}_{\ell+1}^\ell \vartheta_{\ell+1}$

4. **Post-iterate**: repeat step 2 until meeting the switching criterion.

Return θ_ℓ

5.2.2 Algorithm description

As stated at the beginning of the chapter, we assume that interpolators $\mathbf{I}_{\ell+1}^\ell$ and feature vectors $\phi_\ell\left(s\right)$ are available at all levels. We also assume having switching criteria between levels. In our experiments we used fixed times for switching between levels. The derivation of more refined criteria, such as ones dependant on an estimate of the residual decay rate, is kept for future work.

The LB-MGTD(λ) algorithm above uses a sampled trajectory to estimate the value function. Each sample consists of a state transition $s_t \to s_{t+1}$ and a reward g_t (step 2.1). At each time t, a parameter vector θ_ℓ of a single level ℓ is updated, while others denoted θ_m^- are kept fixed. We call the updated level the *active* level.

When the function LB-MGTD-V-cycle is first called, steps 1-2 simply apply standard TD(λ) iterations at the active level $\ell = 0$. Notice that the sampled residual at this level is no other than the one step reward, i.e. $r_\ell = g_t$. However, as we have shown in section 5.1, convergence rate is not uniform over all error components. Some error components are effectively reduced by TD(λ) iterations, while others are not. Ultimately, convergence rate deteriorates as the error becomes *algebraically smooth*. A switching criterion pauses TD(λ) at level $\ell = 0$, and the coarse grid correction is initiated (step 3). The resultant smooth error is approximated at grid level $\ell = 1$ via recursion in step 3.2, and the correction is applied in step 3.3. Finally, post smoothing is done by further applying the standard TD(λ) iterations in step 4 to reduce interpolation error, and the result θ_0 is returned.

The most complex step to comprehend is probably the recursion call (step 3.2), which serves to approximate the smooth error. To better understand this step, lets assume for now that the algorithm uses only a single coarse grid, i.e. $\ell_{\max} = 1$. In this case step 3.2 actually applies steps 1-2 at level $\ell = 1$ and returns. This time, step 2 implements a variant of TD(λ) with two modifications: the feature vector $\phi_0(s)$ is replaced with $\phi_1(s)$, and the sampled reward g_t with the sampled residual $r_1(s_t, s_{t+1}) := g_t - \left(\phi_0(s_t) - \gamma\phi_0(s_{t+1})\right)^T \theta_0^-$. We show in Proposition 1 below, that if allowed to continue indefinitely, the TD(λ) variant converges to the solution of the coarse grid equations (5.2.18). For this reason it makes sense to use $\mathbf{I}_1^0\theta_1$ as an approximation of the smooth error in θ_0. Extension of the two-level scheme to a multi-level one via recursion is straightforward, since step 2 at level ℓ is simply TD(λ) with a feature vector $\phi_\ell(s)$ and a one step reward $r_\ell(s_t, s_{t+1})$. For future use we rewrite the sampled residual

$$r_\ell\left(s_t, s_{t+1}\right) := g_t - \sum_{m=0}^{\ell-1}\left(\phi_m\left(s_t\right) - \gamma\phi_m\left(s_{t+1}\right)\right)^T \theta_m^- \tag{5.2.21}$$

as a recursion

$$r_{\ell+1}\left(s_t, s_{t+1}\right) = r_\ell\left(s_t, s_{t+1}\right) - \left(\phi_\ell\left(s_t\right) - \gamma\phi_\ell\left(s_{t+1}\right)\right)^T \theta_\ell^- . \tag{5.2.22}$$

Finally, we note that the value function for intermediate times (before level ℓ and higher were computed) is available as

$$v(s) = \sum_{m=0}^{\ell-1}\phi_m\left(s\right)^T \theta_m^- + \phi_\ell\left(s\right)^T \theta_\ell . \tag{5.2.23}$$

5.2.3 Analogy to the classical Multigrid algorithm

The LB-MGTD(λ) algorithm is closely related to the solve phase of the classical Multigrid algorithm (see the V-cycle of Algorithm 3.1 in chapter 3). The major steps 1-4 in LB-MGTD(λ) correspond to steps 2-5 in the V-cycle, sharing similar functionality. These include level correction initialization, pre- and post- iteration, and coarse grid correction. Step 1 in the V-cycle, which provides direct solution at $\ell = \ell_{\max}$ is approximated in LB-MGTD(λ) with iterations of the TD(λ) variant at $\ell = \ell_{\max}$ (step 2). Notational differences include: $\mathbf{b} \leftrightarrow g$, $\mathbf{u} \leftrightarrow \theta$, $\mathbf{v} \leftrightarrow \vartheta$. We use the notation ϑ for the error approximated on the coarse grid instead of $\mathbf{v}$ to distinguish it from the notation of the value function **v**. The switching criteria in LB-MGTD(λ) have similar role to the parameters $\beta_{pre}, \beta_{post}$ in the classical algorithm. The purpose of both algorithms is the fast convergence to a solution of a linear equation system: for LB-MGTD(λ) the equation is that of TD(λ) (see equations and definitions in (3.4.29)-(3.4.33))

$$\mathbf{\Phi}_0^T\mathbf{D}\mathbf{M}_\lambda\left(\mathbf{I} - \gamma\mathbf{P}\right)\mathbf{\Phi}_0\theta_0 = \mathbf{\Phi}_0^T\mathbf{D}\mathbf{M}_\lambda\mathbf{g}, \tag{5.2.24}$$

and for the classical Multigrid algorithm it is

$$\mathbf{A}_0\mathbf{u}_0 = \mathbf{b}_0 . \tag{5.2.25}$$

The matrix $\mathbf{\Phi}_0^T \mathbf{D} \mathbf{M}_\lambda \left(\mathbf{I} - \gamma \mathbf{P}\right) \mathbf{\Phi}_0$ and the vector $\mathbf{\Phi}_0^T \mathbf{D} \mathbf{M}_\lambda \mathbf{g}$ in LB-MGTD play the role of $\mathbf{A}_0$ and $\mathbf{b}_0$ in classical Multigrid, respectively. θ_0 and $\mathbf{u}_0$ are the respective unknown vectors.

However, despite structural similarities, the classical Multigrid algorithm cannot be implemented in the learning context. The coarse grid equations (5.2.18) cannot be explicitly represented nor directly solved without an explicit model of the Markov chain. Another major problem arises from the need to restrict the residual onto the coarse grid. Even if the model were known, it would be infeasible to represent the residual since its dimension equals the number of states $|S|$, which is often extremely large or infinite in problems which require function approximation. The LB-MGTD(λ) overcomes both these obstacles by introducing the TD(λ) variant.

5.2.4 Fast TD solvers for the coarsest level

In LB-MGTD(λ), the solution of the coarse grid equations is obtained by recursively applying TD(λ) variants at different resolution levels. As the algorithm progresses to coarser grids the dimension of the unknown parameter vector θ_ℓ decreases. This opens the opportunity to use algorithms such as LSTD(λ) or λ-LSPE that, while being more resource demanding, provide faster convergence (see section 3.4.5).

To solve the coarsest level using these algorithms, one should replace steps 1-2 in LB-MGTD(λ) when in the coarsest level, with variants of LSTD(λ) or λ-LSPE. The variants include the following modifications:

- If the algorithm requires initialization, use $\theta_{\ell_{\max}} := \vartheta_{\ell_{\max}}$.
- Use $\phi_{\ell_{\max}}\left(s_t\right)$ as the feature vector.
- Use $r_{\ell_{\max}}\left(s_t, s_{t+1}\right) := g_t - \sum_{m=0}^{\ell_{\max}-1} \left(\phi_m\left(s_t\right) - \gamma \phi_m\left(s_{t+1}\right)\right)^T \theta_m^-$ as the one step reward.

The frame below presents the λ-LSPE variant used to solve the coarse grid equations. We show in Proposition 1 below, that if allowed to continue indefinitely, these variants applied at the coarsest level converge to the solution of the coarse grid equations (5.2.18).

1. **Apply λ-LSPE on the coarsest grid**:

   1.1 Initialize: $\theta_{\ell_{\max}} := \vartheta_{\ell_{\max}}$, $\mathbf{B} = \delta \mathbf{I}$, $\mathbf{A} = \mathbf{0}$, $\mathbf{b} = \mathbf{0}$, $\mathbf{z}_{\ell_{\max}} = \mathbf{0}$, for some small[2] $\delta > 0$.

   1.2. Observe the transition $s_t \to s_{t+1}$, and the reward g_t at time t.

   1.3. Update the eligibility traces $\mathbf{z}_{\ell_{\max}} := \lambda\gamma \mathbf{z}_{\ell_{\max}} + \phi_{\ell_{\max}}(s_t)$

   1.4. Sample the residual $r_{\ell_{\max}} := g_t - \sum_{m=0}^{\ell_{\max}-1} \left(\phi_m(s_t) - \gamma\phi_m(s_{t+1})\right)^T \theta_m^-$

   1.5. Update

$$\mathbf{B} := \mathbf{B} + \phi_{\ell_{\max}}(s_t)\phi_{\ell_{\max}}(s_t)^T$$

$$\mathbf{A} := \mathbf{A} + \mathbf{z}\left(\phi_{\ell_{\max}}(s_t) - \gamma\phi_{\ell_{\max}}(s_{t+1})\right)^T$$

$$\mathbf{b} := \mathbf{b} + \mathbf{z} r_{\ell_{\max}}$$

$$\theta_{\ell_{\max}} := \theta_{\ell_{\max}} + \alpha_{\ell_{\max},t} \mathbf{B}^{-1}\left(\mathbf{b} - \mathbf{A}\theta_{\ell_{\max}}\right)$$

   1.6. If the switching criterion is met then return $\theta_{\ell_{\max}}$, otherwise repeat from 1.2.

5.2.5 Convergence of the coarse level algorithms

In Proposition 1 below we prove that the TD algorithms applied on coarse levels in LB-MGTD(λ) (Algorithm 5.1), converge with probability 1 to the solution of the coarse grid equations. Our proof uses Theorem 1 in [TVR97], section VI. To avoid confusion, from here on we refer to Theorems 1 and 2 in [TVR97] as Theorems A and B respectively. Lemma 5.4 validates Theorem A's assumptions on the coarse grid levels. In the first part of Proposition 1 we apply Theorem A to prove convergence with probability 1, calculate the convergence point, and show that this point is the solution of the coarse grid equations presented in chapter 3. We restate these equations for convenience:

[2] See [NB03] for a discussion on how to choose δ.

$$\mathbf{A}_\ell \theta_\ell = \mathbf{b}_\ell \tag{5.2.26}$$

$$\mathbf{A}_{\ell+1} \coloneqq \mathbf{I}_\ell^{\ell+1} \mathbf{A}_\ell \mathbf{I}_{\ell+1}^\ell \tag{5.2.27}$$

$$\mathbf{b}_{\ell+1} = \mathbf{I}_\ell^{\ell+1} \left(\mathbf{b}_\ell - \mathbf{A}_\ell \theta_\ell^- \right) \tag{5.2.28}$$

and remind that on the finest grid we have $\mathbf{A}_0 = \mathbf{\Phi}_0^T \mathbf{D} \mathbf{M}_\lambda \left(\mathbf{I} - \gamma \mathbf{P} \right) \mathbf{\Phi}_0$ and $\mathbf{b}_0 = \mathbf{\Phi}_0^T \mathbf{D} \mathbf{M}_\lambda \mathbf{g}$. In the second part, we prove that λ-LSPE and LSTD(λ) converge to the same solution, and can thus serve as fast solvers for the coarsest grid.

We note that Lemma 5.4 is required for infinite MDPs, since for finite state spaces, all assumptions are trivially satisfied.

Lemma 5.4 *Assume the TD(λ) assumption set of section 3.4.3 is satisfied for the finest level* $\ell = 0$. *Further assume that coarse grid feature vectors* $\phi_\ell(s)$ *satisfy (5.1.1), that* $\mathbf{I}_{m+1}^m$, $m = 0, \ldots, \ell - 1$ *have full rank and that the reward functions are defined by (5.2.21). Then the TD(λ) assumption set in section 2.4.3 is satisfied for all grid levels.*

Proof Theorem A defines a metric space $L_2(\mathcal{S}, \mathbf{D})$, of vectors defined over the state space with the norm induced by the inner product $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{D}} \equiv \mathbf{x}^T \mathbf{D} \mathbf{y}$. $\mathbf{D}$ is defined in (3.4.33). $L_2(\mathcal{S}, \mathbf{D})$ is the space of all vectors with a finite norm, i.e. $\|\mathbf{J}\|_{\mathbf{D}} < \infty$. We first note that all levels relate to the same Markov chain and therefore property 1(a) and 4 in the TD(λ) assumption set of section 3.4.3 are trivially satisfied for every ℓ. Assumption 2(b) states that all basis functions are in $L_2(\mathcal{S}, \mathbf{D})$. By definition (see (5.1.1)), coarse grid basis functions at all levels are finite linear combinations of fine grid basis functions, and therefore are in $L_2(\mathcal{S}, \mathbf{D})$. Assumption 2(b) is therefore satisfied at all levels. We now turn to verify 1(b) for the modified reward (5.2.21)

$$\begin{aligned}\mathbb{E}_0\left[r_\ell\left(s_t,s_{t+1}\right)^2\right]&=\mathbb{E}_0\left[\left(g\left(s_t,s_{t+1}\right)-\sum_{m=0}^{\ell-1}\left(\phi_m\left(s_t\right)-\gamma\phi_m\left(s_{t+1}\right)\right)^T\theta_m^-\right)^2\right]=\\&=\mathbb{E}_0\left[g\left(s_t,s_{t+1}\right)^2\right]+\sum_{m=0}^{\ell-1}\mathbb{E}_0\left[g\left(s_t,s_{t+1}\right)\phi_m\left(s_t\right)^T\right]\theta_m^-\\&-\gamma\sum_{m=0}^{\ell-1}\mathbb{E}_0\left[g\left(s_t,s_{t+1}\right)\phi_m\left(s_{t+1}\right)^T\right]\theta_m^-\\&+\mathbb{E}_0\left[\left(\sum_{m=0}^{\ell-1}\left(\phi_m\left(s_t\right)-\gamma\phi_m\left(s_{t+1}\right)\right)^T\theta_m^-\right)^2\right]\end{aligned}\tag{5.2.29}$$

It is readily seen that this expression is a finite sum of scalar elements of the form

$$\mathbb{E}_0\left[J_1\left(s_t\right)J_2\left(s_{t+k}\right)\right]\tag{5.2.30}$$

with $\mathbf{J}_1,\mathbf{J}_2\in L_2\left(\mathcal{S},\mathbf{D}\right)$, since g and the basis functions in the vector $\phi_m\left(s\right)$ are all in $L_2\left(\mathcal{S},\mathbf{D}\right)$. As part of Lemma 7 in [TVR97] it is shown that for any $\mathbf{J}_1,\mathbf{J}_2\in L_2\left(\mathcal{S},\mathbf{D}\right)$ and $k\geq 0$ we have

$$\mathbb{E}_0\left[J_1\left(s_t\right)J_2\left(s_{t+k}\right)\right]=\mathbf{J}_1{}^T\mathbf{D}\mathbf{P}^k\mathbf{J}_2<\infty.\tag{5.2.31}$$

Similarly

$$\mathbb{E}_0\left[J_1\left(s_{t+k}\right)J_2\left(s_t\right)\right]=\mathbf{J}_1{}^T\left(\mathbf{P}^T\right)^k\mathbf{D}\mathbf{J}_2<\infty.\tag{5.2.32}$$

Since (5.2.29) is a finite sum of finite terms, it is finite. Therefore, $r_\ell\left(s_t,s_{t+1}\right)$ satisfies 1(b).

We now turn to verify assumption 2(a) on all levels. On the finest grid $\mathbf{\Phi}_0$ is full rank by assumption 2(a), i.e. its columns are linearly independent. By assumption $\mathbf{I}_{\ell+1}^{\ell}$ are full rank too. Observing (5.1.2), it follows by induction that $\mathbf{\Phi}_{\ell+1}$ is full rank, since it is a multiplication of matrices with linearly independent columns. This proves that 2(a) is valid at all levels.

We proceed to verify assumption 3(a) and 3(b) by induction. It is assumed that at level $\ell=0$ these assumptions are valid. Assuming these assumptions are valid at level ℓ for some function $f_\ell:\mathcal{S}\to\mathbb{R}^+$, we will show they are valid level $\ell+1$. For brevity of notation we shall write

$$\phi_{\ell+1,\tau}=\phi_{\ell+1}\left(s_\tau\right)\quad,\quad g_t=g\left(s_t,s_{t+1}\right)\quad,\quad r_{\ell,t}=r_\ell\left(s_t,s_{t+1}\right). \tag{5.2.33}$$

Using (5.1.2) we obtain

$$\begin{aligned}
&\sum_{\tau=0}^{\infty}\left\|\mathbb{E}\left(\phi_{\ell+1,\tau}\phi_{\ell+1,\tau+m}^{T}\mid s_0\right)-\mathbb{E}_0\left(\phi_{\ell+1,0}\phi_{\ell+1,m}^{T}\right)\right\| \\
&=\sum_{\tau=0}^{\infty}\left\|\left(\mathbf{I}_{\ell+1}^{\ell}\right)^T\left[\mathbb{E}\left(\phi_{\ell,\tau}\phi_{\ell,\tau+m}^{T}\mid s_0\right)-\mathbb{E}_0\left(\phi_{\ell,0}\phi_{\ell,m}^{T}\right)\right]\mathbf{I}_{\ell+1}^{\ell}\right\| \\
&\le\left\|\left(\mathbf{I}_{\ell+1}^{\ell}\right)^T\right\|\left\|\mathbf{I}_{\ell+1}^{\ell}\right\|f_\ell\left(s_0\right)
\end{aligned} \tag{5.2.34}$$

where we remind that $\|\cdot\|$ denotes the Euclidean-induced matrix norm.

For the second bound we substitute (5.2.22) in the following:

$$\begin{aligned}
&\sum_{\tau=0}^{\infty}\left\|\mathbb{E}\left(\phi_{\ell+1,\tau}r_{\ell+1,\tau+m}\mid s_0\right)-\mathbb{E}_0\left(\phi_{\ell+1,0}r_{\ell+1,m}\right)\right\|= \\
&=\sum_{\tau=0}^{\infty}\left\|\mathbf{I}_{\ell}^{\ell+1}\left[\mathbb{E}\left(\phi_{\ell,\tau}r_{\ell+1,\tau+m}\mid s_0\right)-\mathbb{E}_0\left(\phi_{\ell,0}r_{\ell+1,m}\right)\right]\right\| \\
&\le\left\|\mathbf{I}_{\ell}^{\ell+1}\right\|\sum_{\tau=0}^{\infty}\left\|\mathbb{E}\left(\phi_{\ell,\tau}\left[r_{\ell,\tau+m}-\left(\phi_{\ell,\tau+m}-\gamma\phi_{\ell,\tau+m+1}\right)^T\theta_\ell^-\right]\mid s_0\right)\right. \\
&\qquad\left.-\mathbb{E}_0\left(\phi_{\ell,t}\left[r_{\ell,m}-\left(\phi_{\ell,m}-\gamma\phi_{\ell,m+1}\right)^T\theta_\ell^-\right]\right)\right\| \\
&\le\left\|\mathbf{I}_{\ell}^{\ell+1}\right\|\left[\sum_{\tau=0}^{\infty}\left\|\mathbb{E}\left(\phi_{\ell,\tau}r_{\ell,\tau+m}\mid s_0\right)-\mathbb{E}_0\left(\phi_{\ell,0}r_{\ell,m}\right)\right\|\right. \\
&\qquad+\sum_{\tau=0}^{\infty}\left\|\mathbb{E}\left(\phi_{\ell,\tau}\phi_{\ell,\tau+m}^{T}\mid s_0\right)-\mathbb{E}_0\left(\phi_{\ell,t}\phi_{\ell,t+m}^{T}\right)\right\|\left\|\theta_\ell^-\right\| \\
&\qquad\left.+\gamma\sum_{\tau=0}^{\infty}\left\|\mathbb{E}\left(\phi_{\ell,\tau}\phi_{\ell,\tau+m+1}^{T}\mid s_0\right)-\mathbb{E}_0\left(\phi_{\ell,0}\phi_{\ell,m+1}^{T}\right)\right\|\left\|\theta_\ell^-\right\|\right] \\
&\le\left\|\mathbf{I}_{\ell}^{\ell+1}\right\|\left(1+\left(1+\gamma\right)\left\|\theta_\ell^-\right\|\right)f_\ell\left(s_0\right)
\end{aligned} \tag{5.2.35}$$

We validate 3(a) by defining

$$f_{\ell+1}\left(s\right)=c\cdot f_\ell\left(s\right) \tag{5.2.36}$$

$$c = \max\left\{\left\|\left(\mathbf{I}_{\ell+1}^{\ell}\right)^T\right\|\left\|\mathbf{I}_{\ell+1}^{\ell}\right\|, \left\|\mathbf{I}_{\ell+1}^{\ell}\right\|\left(1+(1+\gamma)\left\|\theta_{\ell}^{-}\right\|\right)\right\}. \tag{5.2.37}$$

Since c is finite we validate 3(b) by multiplying both side by c^n

$$c^n \cdot \mathbb{E}\left(f_{\ell}^n(s_t) \mid s_0\right) \le c^n \cdot \mu_n f_{\ell}^n(s_0) \quad \Rightarrow \quad \mathbb{E}\left(f_{\ell+1}^n(s_t) \mid s_0\right) \le \mu_n f_{\ell+1}^n(s_0). \tag{5.2.38}$$

This concludes the validation of the assumption set for any level ℓ.

■

Proposition 1 *Consider a Markov chain with either a finite or infinite countable state space, with discounted cost, and the TD(λ) assumption set in section 2.4.3 for level* $\ell = 0$. *Then, the following Temporal Difference algorithms applied at level* ℓ *using* $\phi_{\ell}(s)$ *and the reward function defined in (5.2.21) converge with probability 1 to the solution of the coarse grid equations defined in (5.2.26)-(5.2.28):*

(a) TD(λ) under the TD(λ) assumption set in section 2.4.3.

(b) λ-LSPE under assumptions 1(a), 2(a) and 4 in the assumption set.

(c) LSTD(λ) under assumptions 1(a), 2(a) in the assumption set.

Proof We start by proving *(a)*. Lemma 5.4 validates the assumptions of Theorem A on the coarse grid level ℓ. By Theorem A, TD(λ) applied using $\phi_{\ell}(s)$ and $r_{\ell}(s_t, s_{t+1})$ converges with probability 1, to the vector θ^* satisfying

$$\hat{\mathbf{A}}_{\ell}\theta_{\ell}^* = \hat{\mathbf{b}}_{\ell}. \tag{5.2.39}$$

$$\hat{\mathbf{A}}_{\ell} = \mathbb{E}_0\left[\hat{\mathbf{z}}_{\ell,t}\left(\phi_{\ell}(s_t) - \gamma\phi_{\ell}(s_{t+1})\right)^T\right] \tag{5.2.40}$$

$$\hat{\mathbf{b}}_{\ell} = \mathbb{E}_0\left[\hat{\mathbf{z}}_{\ell,t} r_{\ell}(s_t, s_{t+1})\right] \tag{5.2.41}$$

$$\hat{\mathbf{z}}_{\ell,t} = \sum_{\tau=-\infty}^{t} (\gamma\lambda)^{t-\tau}\phi_{\ell}(s_{\tau}) \tag{5.2.42}$$

We used hats to temporarily distinguish these from $\mathbf{A}_\ell$, $\mathbf{b}_\ell$ and $\mathbf{z}_\ell$ defined previously.

First, we show that $\hat{\mathbf{A}}_\ell = \mathbf{A}_\ell$. Using (5.1.1) and $\mathbf{I}_\ell^{\ell+1} = \left(\mathbf{I}_{\ell+1}^{\ell}\right)^T$ we have

$$\hat{\mathbf{z}}_{\ell+1,t} = \sum_{\tau=-\infty}^{t} (\gamma\lambda)^{t-\tau} \phi_{\ell+1}(s_\tau) = \sum_{\tau=-\infty}^{t} (\gamma\lambda)^{t-\tau} \left(\mathbf{I}_{\ell+1}^{\ell}\right)^T \phi_\ell(s_\tau) = \left(\mathbf{I}_{\ell+1}^{\ell}\right)^T \hat{\mathbf{z}}_{\ell,t} = \mathbf{I}_\ell^{\ell+1} \hat{\mathbf{z}}_{\ell,t}. \tag{5.2.43}$$

We substitute this in (5.2.40) and verify that $\hat{\mathbf{A}}_\ell$ satisfies (5.2.27),

$$\begin{aligned} \hat{\mathbf{A}}_{\ell+1} &= \mathbb{E}_0\left[\hat{\mathbf{z}}_{\ell+1,t}\left(\phi_{\ell+1}(s_t) - \gamma\phi_{\ell+1}(s_{t+1})\right)^T\right] \\ &= \mathbf{I}_\ell^{\ell+1}\mathbb{E}_0\left[\hat{\mathbf{z}}_{\ell,t}\left(\phi_\ell(s_t) - \gamma\phi_\ell(s_{t+1})\right)^T\right]\mathbf{I}_{\ell+1}^{\ell} = \mathbf{I}_\ell^{\ell+1}\hat{\mathbf{A}}_\ell\mathbf{I}_{\ell+1}^{\ell} \end{aligned}. \tag{5.2.44}$$

Since by definition $\hat{\mathbf{A}}_0 = \mathbf{A}_0$ it follows that $\hat{\mathbf{A}}_\ell = \mathbf{A}_\ell$. We now show that $\hat{\mathbf{b}}_\ell = \mathbf{b}_\ell$. By substituting (5.2.21) in (5.2.41) we verify that $\hat{\mathbf{b}}_\ell$ satisfies (5.2.28),

$$\begin{aligned} \hat{\mathbf{b}}_{\ell+1} &= \mathbb{E}_0\left[\hat{\mathbf{z}}_{\ell+1,t} r_{\ell+1}(s_t, s_{t+1})\right] = \left(\mathbf{I}_{\ell+1}^{\ell}\right)^T \mathbb{E}_0\left[\hat{\mathbf{z}}_{\ell,t} r_{\ell+1}(s_t, s_{t+1})\right] \\ &= \mathbf{I}_\ell^{\ell+1}\mathbb{E}_0\left[\hat{\mathbf{z}}_{\ell,t}\left(r_\ell(s_t, s_{t+1}) - \left(\phi_\ell(s_t) - \gamma\phi_\ell(s_{t+1})\right)^T \theta_\ell^-\right)\right] \\ &= \mathbf{I}_\ell^{\ell+1}\left(\mathbb{E}_0\left[\hat{\mathbf{z}}_{\ell,t} r_\ell(s_t, s_{t+1})\right] - \mathbb{E}_0\left[\hat{\mathbf{z}}_{\ell,t}\left(\phi_\ell(s_t) - \gamma\phi_\ell(s_{t+1})\right)^T\right]\theta_\ell^-\right) \\ &= \mathbf{I}_\ell^{\ell+1}\left(\hat{\mathbf{b}}_\ell - \hat{\mathbf{A}}_\ell\theta_\ell^-\right) \end{aligned} \tag{5.2.45}$$

Again, since $\hat{\mathbf{b}}_0 = \mathbf{b}_0 = \mathbf{g}$ it follows that $\hat{\mathbf{b}}_\ell = \mathbf{b}_\ell$. We have shown that (5.2.26) and (5.2.39) are the same, hence the convergence point satisfies the coarse grid equations (5.2.26)-(5.2.28), thus proving *(a)*.

Convergence with probability 1 of λ-LSPE and LSTD(λ) to the same point as TD(λ) was proven in [NB03]. Since all assumptions are valid on the coarse grid ℓ by Lemma 5.4, convergence is guaranteed. By part *(a)* the convergence point satisfies (5.2.26)-(5.2.28).

■

Proposition 1 shows convergence of each level separately, indicating that Algorithm 5.1 is admissible. However, as in the known model case discussed in section 4.1.3, this does not

imply convergence of the entire scheme. Since $\mathbf{A}$ is not symmetric, even if all coarse grid equations are solved exactly, divergence may occur.

5.3 A simultaneous Multigrid TD algorithm

5.3.1 Main idea and purpose

In LB-MGTD(λ) (Algorithm 5.1) we used distinct separation between resolution levels. At every point in time, only a single level is active, receiving a trajectory sample and updating its parameter vector, while parameter vectors for all other levels are kept fixed. When obtaining data samples is costly, it may be considered wasteful to use each sample in order to update only a single level.

One option is to store a trajectory section sampled on the finest grid, and reapply it on coarser grids. This procedure does not require the generation of new samples when updating the coarse levels. This approach is similar to the well-known experience replay approach [L93].

We propose another approach that uses a single trajectory sample to simultaneously update multiple grid levels. Although grid separation is convenient in the known model case, it is not necessary in the learning scenario. In Algorithm 5.2 below, we propose to update the parameter vectors at all levels simultaneously, using a single temporal-difference value. We name this algorithm Simultaneous Multigrid TD (*S-MGTD(λ)*).

5.3.2 Algorithm presentation

As stated at the beginning of the chapter, we assume that the feature vectors $\phi_\ell(s)$ are available at all levels. We further assume that learning step sequences are defined for all levels and denoted by $\{\alpha_{\ell,t}\}_{\ell=0}^{\ell_{\max}}$. The parameter vector of level ℓ to be learned is denoted by ω_ℓ, to avoid confusion with the parameters θ_ℓ of LB-MGTD(λ) which have a different meaning. In our experiments we initialize $\omega_\ell = \mathbf{0}$ for $\ell = 0,\ldots,\ell_{\max}$. The S-MGTD(λ) algorithm is activated by

$$\left[\omega_0, \omega_1, \ldots, \omega_{\ell_{\max}}\right] := \text{S-MGTD}\left(\omega_0, \omega_1, \ldots, \omega_{\ell_{\max}}\right). \tag{5.3.46}$$

Algorithm 5.2: Simultaneous Multigrid-TD(λ) (S-MGTD)

S-MGTD$\left(\omega_0, \omega_1, \ldots, \omega_{\ell_{\max}}\right)$

1. Initialize: $\mathbf{z}_\ell = \mathbf{0}$ for all $\ell = 0, \ldots, \ell_{\max}$
2. Iterate steps 2.1-2.3, over time t:
   2.1. Observe the transition $s_t \to s_{t+1}$ and the reward g_t at time t.
   2.2. Calculate the temporal difference $d = g_t - \sum_{\ell=0}^{\ell_{\max}} \left(\phi_\ell\left(s_t\right) - \gamma\phi_\ell\left(s_{t+1}\right)\right)^T \omega_\ell$
   2.3. For $\ell = 0$ to $\ell_{\max}$ do:
      Update the eligibility traces $\mathbf{z}_\ell := \lambda\gamma\mathbf{z}_\ell + \phi_\ell\left(s_t\right)$
      Update $\omega_\ell := \omega_\ell + \alpha_{\ell,t}\mathbf{z}_\ell d$

The approximate cost-to-go function is given at all times by $v\left(s\right) = \sum_{\ell=0}^{\ell_{\max}} \phi_\ell\left(s\right)^T \omega_\ell$.

The value function in S-MGTD(λ) (Algorithm 5.2) is approximated as a linear combination of basis functions from all resolution levels

$$v\left(s\right) = \sum_{\ell=0}^{\ell_{\max}} \phi_\ell\left(s\right)^T \omega_\ell . \tag{5.3.47}$$

This algorithm has the same form as standard TD(λ) algorithm with two differences: the basis functions are now dependant since $\phi_{\ell+1}\left(s\right)$ depends on $\phi_\ell\left(s\right)$ via (5.1.1) and therefore form an over-complete basis, and the learnable parameters at different levels have different learning step sequences.

To gain more insight into this algorithm, we show in section 5.3.3 that for a proper choice of learning step sequences, this algorithm is equivalent to the LB-MGTD(λ) algorithm presented previously in section 5.2. Next in section 5.3.4, we interpret this algorithm as

TD(λ) applied at the finest level $\ell = 0$ with a matrix sequence for learning steps instead of a scalar sequence. We use this in section 5.3.5 to analyze the case in which the learning steps of different levels proportional to each other.

5.3.3 Level-based as a special case of simultaneous Multigrid TD

We now discuss a special case for which S-MGTD(λ) resembles LB-MGTD(λ) (Algorithm 5.1). For now, we distinguish between the variables of the algorithms, by applying hats to variables of S-MGTD(λ), denoting $\hat{\mathbf{z}}_\ell, \hat{v}(s), \hat{\alpha}_t$ instead of $\mathbf{z}_\ell, v(s), \alpha_t$. We show below that these elements are identical respectively for both algorithms.

Consider the execution of both algorithms in parallel, feeding them the same trajectory samples. Starting with $T_0 = 0$, denote by T_i, $i \in \{0,1,2,\ldots\}$ the time instances in which the active level ℓ in LB-MGTL(λ) changes (when MG-TD-V-cycle is called are returns), and by ℓ_i the active level during the interval $[T_i, T_{i+1})$. In particular we have $\ell_i = i$ for $i \in \{0,1,\ldots,\ell_{\max}\}$, and $\ell_i = 2\ell_{\max} - i$ for $i \in \{\ell_{\max}+1, \ell_{\max}+2, \ldots, 2\ell_{\max}\}$. We define the learning-step scheme for S-MGTD(λ) during the interval $t \in [T_i, T_{i+1})$ according to

$$\hat{\alpha}_{m,t} = \begin{cases} \alpha_t & \text{if } m = \ell_i \\ 0 & \text{otherwise} \end{cases}. \tag{5.3.48}$$

Since the learning steps for all levels but the active one are zero, only the parameter vector of the active level changes. To emphasize this, we denote the fixed parameter vectors by $\omega_m^- = \omega_m$ for $m \neq \ell$.

Lemma 5.5 *Consider a parallel execution of LB-MGTD(λ) with TD(λ) on the coarsest level and a modified version of S-MGTD(λ), which applies a reset $\mathbf{z}_\ell = \mathbf{0}$ at the active level when $t = T_i$. Assume the learning scheme for S-MGTD(λ) is defined by (5.3.48). Then the algorithms are equivalent in the sense that $\hat{v}(s) = v(s)$ for all t.*

Proof In order to easily compare $\hat{v}(s)$ and $v(s)$, we first write them both in terms of $\phi_0(s)$. Substituting (5.1.1) in itself recursively we obtain the following relation for $m < \ell$

$$\phi_\ell(s)^T = \phi_m(s)^T \mathbf{I}_\ell^m \tag{5.3.49}$$

where $\mathbf{I}_\ell^m$ denotes the level ℓ to level m interpolator. It is defined by

$$\mathbf{I}_\ell^m = \mathbf{I}_{m-1}^m \mathbf{I}_{m-2}^{m-1} \ldots \mathbf{I}_\ell^{\ell-1}. \tag{5.3.50}$$

As a convention we define $\mathbf{I}_\ell^\ell$ to be the identity matrix. For future use we use (5.3.49) to re-write the eligibility traces as

$$\mathbf{z}_{\ell,t} = \sum_{\tau=0}^{t} (\gamma\lambda)^{t-\tau} \phi_\ell(s_\tau) = \sum_{\tau=0}^{t} (\gamma\lambda)^{t-\tau} \left(\mathbf{I}_\ell^0\right)^T \phi_0(s_\tau) = \left(\mathbf{I}_\ell^0\right)^T \mathbf{z}_{0,t}. \tag{5.3.51}$$

We now apply (5.3.49) to (5.2.23) and to (5.3.47) to obtain

$$v(s) = \sum_{m=0}^{\ell-1} \phi_m(s)^T \theta_m^- + \phi_\ell(s)^T \theta_\ell = \phi_0(s)^T \left(\sum_{m=0}^{\ell-1} \mathbf{I}_m^0 \theta_m^- + \mathbf{I}_\ell^0 \theta_\ell \right) \tag{5.3.52}$$

$$\hat{v}(s) = \sum_{m=0}^{\ell_{\max}} \phi_m(s)^T \omega_m = \phi_0(s)^T \left(\sum_{m=0}^{\ell_{\max}} \mathbf{I}_m^0 \omega_m \right). \tag{5.3.53}$$

We proof the lemma by showing next that at all times

$$\sum_{m=0}^{\ell-1} \mathbf{I}_m^0 \theta_m^- + \mathbf{I}_\ell^0 \theta_\ell = \sum_{m=0}^{\ell_{\max}} \mathbf{I}_m^0 \omega_m. \tag{5.3.54}$$

The proof follows by induction. At $t = 0$ the equation (5.3.54) is trivially satisfied since all parameter vectors are initialized as zero. At time t, both algorithms observe the transition $s_t \to s_{t+1}$ and the reward g_t, and add a term of the form $\alpha_\ell \mathbf{z}_\ell d$ to a parameter vector. In LB-MGTD(λ), θ_ℓ is updated, changing the left side of (5.3.54) according to

$$\sum_{m=0}^{\ell-1} \mathbf{I}_m^0 \theta_m^- + \mathbf{I}_\ell^0 \theta_\ell \coloneqq \sum_{m=0}^{\ell-1} \mathbf{I}_m^0 \theta_m^- + \mathbf{I}_\ell^0 \left(\theta_\ell + \alpha_t \mathbf{z}_\ell d_\ell\right) = \left[\sum_{m=0}^{\ell-1} \mathbf{I}_m^0 \theta_m^- + \mathbf{I}_\ell^0 \theta_\ell \right] + \alpha_t \mathbf{I}_\ell^0 \mathbf{z}_\ell d_\ell. \tag{5.3.55}$$

In S-MGTD(λ), $\{\omega_m\}_{m=0}^{\ell_{\max}}$ are all updated, changing the left side of (5.3.54) according to

$$\sum_{m=0}^{\ell_{\max}} \mathbf{I}_m^0 \omega_m \coloneqq \sum_{m=0}^{\ell_{\max}} \mathbf{I}_m^0 \left(\omega_m + \hat{\alpha}_{m,t} \hat{\mathbf{z}}_m \hat{d}\right) = \left[\sum_{m=0}^{\ell_{\max}} \mathbf{I}_m^0 \omega_m\right] + \hat{\alpha}_{\ell,t} \mathbf{I}_\ell^0 \hat{\mathbf{z}}_\ell \hat{d} \tag{5.3.56}$$

where we used (5.3.51) and the fact that $\hat{\alpha}_{m,t} \neq 0$ only at the active level. We argue that the terms added to both sides of (5.3.54) are equal, i.e.

$$\hat{\alpha}_{\ell,t} \mathbf{I}_\ell^0 \hat{\mathbf{z}}_\ell \hat{d}_\ell = \alpha_t \mathbf{I}_\ell^0 \mathbf{z}_\ell d \,. \tag{5.3.57}$$

First, notice that at the active level we have $\hat{\mathbf{z}}_\ell = \mathbf{z}_\ell$ at all times, which follows trivially from resetting them at the same times T_i, and applying the same update rule until T_{i+1}. Both algorithms calculate the temporal difference defined in (3.3.21) restated here for convenience

$$d\left(s_t, s_{t+1}; \hat{v}\right) = \left[g\left(s_t, s_{t+1}\right) + \gamma \hat{v}\left(s_{t+1}\right)\right] - \hat{v}\left(s_t\right). \tag{5.3.58}$$

By the induction assumption, (5.3.54) is satisfied, therefore $\hat{v}(s) = v(s)$ and $\hat{d} = d_\ell$. Finally, $\hat{\alpha}_{\ell,t} = \alpha_t$ by definition. The above shows that the update rules of both algorithms add the same term to both sides of (5.3.54), satisfying the induction for $t+1$. To conclude the proof we show that (5.3.54) is satisfied after applying the correction step in LB-MGTD(λ) (step 3.3). This follows directly from the equality

$$\begin{aligned} &\sum_{m=0}^{\ell} \mathbf{I}_m^0 \theta_m^- + \mathbf{I}_{\ell+1}^0 \theta_{\ell+1} = \sum_{m=0}^{\ell-1} \mathbf{I}_m^0 \theta_m^- + \mathbf{I}_\ell^0 \theta_\ell^- + \mathbf{I}_\ell^0 \mathbf{I}_{\ell+1}^\ell \theta_{\ell+1} \\ &= \sum_{m=0}^{\ell-1} \mathbf{I}_m^0 \theta_m^- + \mathbf{I}_\ell^0 \left(\theta_\ell^- + \mathbf{I}_{\ell+1}^\ell \theta_{\ell+1}\right) = \sum_{m=0}^{\ell-1} \mathbf{I}_m^0 \theta_m^- + \mathbf{I}_\ell^0 \theta_\ell \end{aligned} \tag{5.3.59}$$

which shows that the left hand side of (5.3.54) remains unchanged under the correction step, when the active level changes from $\ell+1$ to ℓ. Since (5.3.54) is satisfied at all times, it follows that $\hat{v}(s) = v(s)$, which concludes the proof.

■

In light of Lemma 5.5, we interpret the update rule of S-MGTD(λ) as a merger of the correction and the iteration steps of LB-MGTD(λ) (steps 2-4). The vectors ω_m separately store

the coarse corrections to finer levels in memory. They are implicitly added in the calculations of the value function as in (5.3.53) and the temporal differences.

We note that although we usually assume that the basis functions within each level ℓ are linearly independent, the collection of all basis functions from all the levels forms a linearly dependent set. As a result, we cannot expect S-MGTD(λ) to have a unique convergence point in terms of $\{\omega_m\}_{m=0}^{\ell_{\max}}$, rather hope for uniqueness of the approximated function (5.3.47).

5.3.4 S-MGTD(λ) - finest grid formulation

In this section we formulate the solution of S-MGTD(λ) as a function of the finest grid basis functions only. We show that the update rule takes the form of a TD(λ) algorithm operating at the finest grid, with a varying matrix taking the place of the learning step. This form will turn out useful in the subsequent theoretical analysis. We write (5.3.53) as

$$v(s) = \phi_0(s)^T \omega . \tag{5.3.60}$$

where

$$\omega = \sum_{\ell=0}^{\ell_{\max}} \mathbf{I}_\ell^0 \omega_\ell . \tag{5.3.61}$$

The update rules of S-MGTD(λ) (step 2.3) changes ω according to

$$\omega := \sum_{\ell=0}^{\ell_{\max}} \mathbf{I}_\ell^0 \left(\omega_\ell + \alpha_{\ell,t} \mathbf{z}_\ell d\right) = \omega + \sum_{\ell=0}^{\ell_{\max}} \mathbf{I}_\ell^0 \alpha_{\ell,t} \mathbf{z}_\ell d = \omega + \left[\sum_{\ell=0}^{\ell_{\max}} \alpha_{\ell,t} \mathbf{I}_\ell^0 \left(\mathbf{I}_\ell^0\right)^T\right] \mathbf{z}_0 d \tag{5.3.62}$$

where we used (5.3.51). We can now rewrite S-MGTD(λ) as an equivalent TD(λ) algorithm. For ease of notation we omit the zero level index and write $\mathbf{z}$ instead of $\mathbf{z}_0$ in the following.

Algorithm 5.3: S-MGTD - equivalent algorithm

1.	Initialize:	$\mathbf{z} \coloneqq \mathbf{0}, \omega \coloneqq \mathbf{0}$
2.	Iterate:	
	2.1. Calculate the temporal difference	$d \coloneqq g_t - \left(\phi_0\left(s_t\right) - \gamma\phi_0\left(s_{t+1}\right)\right)^T \omega$
	2.2. Update the eligibility traces	$\mathbf{z} \coloneqq \lambda\gamma\mathbf{z} + \phi_0\left(s_t\right)$
	2.3. Update	$\omega \coloneqq \omega + \mathbf{\Lambda}_t \mathbf{z} d$

where the learning step is now the matrix sequence

$$\mathbf{\Lambda}_t = \sum_{\ell=0}^{\ell_{\max}} \alpha_{\ell,t} \mathbf{I}_\ell^0 \left(\mathbf{I}_\ell^0\right)^T . \tag{5.3.63}$$

The value functions estimated by S-MGTD(λ) and Algorithm 5.3 are the same. We emphasize that Algorithm 5.3 is inferior to S-MGTD(λ) in terms of computational load, since its update rule takes $O(K^2)$ operations where K is the dimension of $\phi_0\left(s\right)$, as the learning step matrix is not sparse. However, its TD(λ) like form assists in theoretical analysis of S-MGTD(λ), as demonstrated in the following section.

5.3.5 Convergence analysis for proportional learning steps

In this section we address the convergence issue of S-MGTD(λ) for learning step schemes of the form

$$\alpha_{\ell,t} = \beta_\ell \alpha_t \tag{5.3.64}$$

where $\beta_0 > 0$, $\beta_\ell \geq 0$ for $\ell \in 1,\ldots,\ell_{\max}$ are constants, and α_t is a predetermined positive, non-increasing sequence that satisfies $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$ (assumption 4 of the TD(λ) assumption set in section 3.4.3). We say that such schemes have *proportional learning steps*.

We start our analysis of these schemes with a proof of convergence, when the transition matrix is symmetric (Theorem 3 below). As in the known model case, convergence for the

non-symmetric case is not guaranteed for general choice of β_ℓ. In the second part of this section we support this claim with a counter example. In the convergence proof to follow we use the following lemma, which considers a generalized form of TD(λ).

Proposition 2 *Let a symmetric positive definite matrix* **C** *be given. Then under the TD(λ) assumption set in section 2.4.3 the following variants of TD(λ) converge with probability 1 to the same solution as TD(λ).*

(a) $$\mathbf{z}_t = \lambda\gamma\mathbf{z}_{t-1} + \phi(s_t), \quad \omega := \omega + \alpha_t \mathbf{C}\mathbf{z}d \tag{5.3.65}$$

where the correction term in TD(λ) is now multiplied by the matrix **C**.

(b) $$\tilde{\phi}(s) = \mathbf{C}\phi(s), \quad \tilde{\mathbf{z}}_t = \lambda\gamma\tilde{\mathbf{z}}_{t-1} + \tilde{\phi}(s_t), \quad \omega_t := \omega_{t-1} + \alpha_t \tilde{\mathbf{z}}_t d_t . \tag{5.3.66}$$

where the basis functions used in the eligibility traces of TD(λ) are now multiplied by **C**, and are therefore different than the basis functions used in the approximation of the value function.

Proof Our proof of part ***(a)*** follows the line of proof of Theorem A, which constructs the Markov process $\mathbf{X}_t = (s_t, s_{t+1}, \mathbf{z}_t)$, and rewrites the TD(λ) algorithm as a stochastic approximation (see section III part C in [TVR97])

$$\omega_t = \omega_{t-1} + \alpha_t \left(\mathbf{b}(\mathbf{X}_t) - \mathbf{A}(\mathbf{X}_t)\omega_{t-1}\right) \tag{5.3.67}$$

where

$$\begin{aligned} \mathbf{A}(\mathbf{X}_t) &= \mathbf{z}_t \left(\phi(\mathbf{X}_t) - \gamma\phi(\mathbf{X}_{t+1})\right)^T \\ \mathbf{b}(\mathbf{X}_t) &= \mathbf{z}_t g(s_t, s_{t+1}) \quad . \end{aligned} \tag{5.3.68}$$

The TD(λ) assumption set is shown to be sufficient to satisfy the assumptions of Theorem B stated below, proving convergence of (5.3.67) with probability 1. Notice that opposed to AMG convergence theory, there is no need here for **A** to be symmetric.

Theorem B (Theorem 2 in [TVR97])

Consider an iterative algorithm of the form (5.3.67) where

1. the step-size sequence α_t is positive, non-increasing, predetermined, and satisfies $\sum_{t=0}^{\infty}\alpha_t = \infty$ and $\sum_{t=0}^{\infty}\alpha_t^2 < \infty$;
2. $\mathbf{X}_t$ is a Markov process with a unique invariant distribution, and there exists a mapping h from the states of the Markov process to the positive reals, satisfying the remaining conditions. Let $\mathbb{E}_0[\cdot]$ stand for expectation with respect to this invariant distribution;
3. the following matrix and vector are well defined and finite
$$\begin{aligned}\mathbf{A} &= \mathbb{E}_0\left[\mathbf{A}(\mathbf{X}_t)\right] \\ \mathbf{b} &= \mathbb{E}_0\left[\mathbf{b}(\mathbf{X}_t)\right]\end{aligned};$$
4. the matrix $-\mathbf{A}$ is stable;
5. there exist constants c and k such that for all $\mathbf{X}$
$$\sum_{t=0}^{\infty}\left\|\mathbb{E}\left[\mathbf{A}(X_t)\,|\,\mathbf{X}_0=\mathbf{X}\right]-\mathbf{A}\right\| \le c\left(1+h^k(\mathbf{X})\right)$$
and
$$\sum_{t=0}^{\infty}\left\|\mathbb{E}\left[\mathbf{b}(X_t)\,|\,\mathbf{X}_0=\mathbf{X}\right]-\mathbf{b}\right\| \le c\left(1+h^k(\mathbf{X})\right);$$
6. for any $k>1$ there exist a constant μ_k such that for all $\mathbf{X}, t$
$$\mathbb{E}\left[h^k(\mathbf{X}_t\,|\,\mathbf{X}_0=\mathbf{X})\right] \le \mu_k\left(1+h^k(\mathbf{X})\right).$$

Then, θ_t converges to θ^* with probability 1, where θ^* is the unique vector that satisfies
$$\mathbf{A}\theta^* = \mathbf{b}.$$

In assumption 4 above the original requirement in [TVR97] is for $-\mathbf{A}$ to be negative definite. However, it suffices to require that $-\mathbf{A}$ is stable, since this is only needed to ensure that the equation $\frac{d}{dt}\theta = -\mathbf{A}\theta + \mathbf{b}$ is stable[3].

Following the above line of proof, we rewrite (5.3.65) as a stochastic approximation

[3] Theorem B was deduced from Theorem 17 (page 239) in [BMP90], by using the potential function $U(\theta) = \left\|\theta-\theta^*\right\|^2$. The same can be deduced by using the Lyapunov function $U(\theta) = (\theta-\theta^*)^T\mathbf{P}(\theta-\theta^*)$, with $\mathbf{P}$ the solution of the Lyapunov equation $(-\mathbf{A})^T\mathbf{P}+\mathbf{P}(-\mathbf{A}) = -\mathbf{Q}$, for any choice of $\mathbf{Q}>0$.

$$\omega_t = \omega_{t-1} + \alpha_t \left(\tilde{\mathbf{b}}(\mathbf{X}_t) - \tilde{\mathbf{A}}(\mathbf{X}_t)\omega_{t-1} \right) \tag{5.3.69}$$

where

$$\begin{aligned} \tilde{\mathbf{A}}(\mathbf{X}_t) &= \mathbf{C}\mathbf{z}_t \left(\phi(\mathbf{X}_t) - \gamma\phi(\mathbf{X}_{t+1}) \right)^T \\ \tilde{\mathbf{b}}(\mathbf{X}_t) &= \mathbf{C}\mathbf{z}_t g(s_t, s_{t+1}) \end{aligned} . \tag{5.3.70}$$

Next, we show that if the assumptions of Theorem B are satisfied for standard TD(λ) in (5.3.67), they are also satisfied for our variant (5.3.69). By transitivity this will show that the TD(λ) assumption set is sufficient to satisfy the assumptions of Theorem B for (5.3.69), proving convergence. Assumptions 1,2 and 6 are the same for both cases. Showing that satisfying assumptions 3-5 for (5.3.67) implies their validity for (5.3.69) is straightforward as follows:

3) By substitution

$$\tilde{\mathbf{A}} = \mathbb{E}_0 \left[\mathbf{C}\mathbf{A}(\mathbf{X}_t) \right] = \mathbf{C}\mathbb{E}_0 \left[\mathbf{A}(\mathbf{X}_t) \right] = \mathbf{C}\mathbf{A} \tag{5.3.71}$$

and similarly $\tilde{\mathbf{b}} = \mathbf{C}\mathbf{b}$. Since $\mathbf{A}, \mathbf{b}$ and $\mathbf{C}$ are well defined and finite, so are $\tilde{\mathbf{A}}, \tilde{\mathbf{b}}$.

4) Consider the real matrix $\tilde{\mathbf{A}} = \mathbf{C}\mathbf{A}$. It is straightforward from Lyapunov's first theorem (see section 3.5) that if there exists a matrix $\mathbf{L} > 0$ such that $\tilde{\mathbf{A}}^T\mathbf{L} + \mathbf{L}\tilde{\mathbf{A}} > \mathbf{0}$, then $-\tilde{\mathbf{A}}$ is stable. By assumption $\mathbf{C} > 0$, and so we can take $\mathbf{L} = \mathbf{C}^{-1} > 0$. This gives

$$\tilde{\mathbf{A}}^T\mathbf{L} + \mathbf{L}\tilde{\mathbf{A}} = \mathbf{A}^T\mathbf{C}^T\mathbf{L} + \mathbf{L}\mathbf{C}\mathbf{A} = \mathbf{A}^T\mathbf{C}^T\mathbf{C}^{-1} + \mathbf{C}^{-1}\mathbf{C}\mathbf{A} = \mathbf{A}^T + \mathbf{A} > 0, \tag{5.3.72}$$

where the last inequality follows from $-\mathbf{A}$ being stable, showing that $-\tilde{\mathbf{A}}$ is stable as well.

5) By substitution we have

$$\begin{aligned} &\sum_{t=0}^{\infty} \left\| \mathbb{E}\left[\tilde{\mathbf{A}}(X_t) \mid \mathbf{X}_0 = \mathbf{X} \right] - \tilde{\mathbf{A}} \right\| = \sum_{t=0}^{\infty} \left\| \mathbf{C}\left[\mathbb{E}\left[\mathbf{A}(X_t) \mid \mathbf{X}_0 = \mathbf{X} \right] - \mathbf{A} \right] \right\| \\ &\le \|\mathbf{C}\| \sum_{t=0}^{\infty} \left\| \left[\mathbb{E}\left[\mathbf{A}(X_t) \mid \mathbf{X}_0 = \mathbf{X} \right] - \mathbf{A} \right] \right\| \le \|\mathbf{C}\| c\left(1 + h^k(\mathbf{X})\right) = \tilde{c}\left(1 + h^k(\mathbf{X})\right), \end{aligned} \tag{5.3.73}$$

for $\tilde{c} = \|\mathbf{C}\| c$. Proving the second inequality is similar.

We now apply Theorem B to (5.3.69), proving convergence with probability 1 to the unique solution of

$$\tilde{\mathbf{A}}\omega^* = \tilde{\mathbf{b}}. \tag{5.3.74}$$

By assumption $\mathbf{C} > 0$ and therefore invertible. We multiply both sides by $\mathbf{C}^{-1}$ showing that the convergence point satisfies

$$\mathbf{A}\omega^* = \mathbf{b} \tag{5.3.75}$$

which is the same convergence point as for TD(λ). This concludes part ***(a)***.

Part ***(b)*** follows trivially by noticing that

$$\tilde{\mathbf{z}}_t = \sum_{\tau=0}^{t} (\gamma\lambda)^{t-\tau} \tilde{\phi}(s_\tau) = \sum_{\tau=0}^{t} (\gamma\lambda)^{t-\tau} \mathbf{C}\phi(s_\tau) = \mathbf{C}\mathbf{z}_t$$

which shows that the update rules of ***(a)*** and ***(b)*** are the same.

■

Next, we make use of the above proposition to prove the convergence of S-MGTD(λ) with proportional learning steps, as shown in the following proposition.

Theorem 3 *Consider a Markov chain with either a finite or infinite (countable) state space, with discounted cost, and assume the TD(λ) assumption set in section 2.4.3 is satisfied. Then S-MGTD(λ) (Algorithm 5.2) with proportional learning steps (5.3.64) converges with probability 1 to the same value function as TD(λ) applied at* $\ell = 0$.

Proof For proportional learning steps, the learning step matrix defined in (5.3.63) is

$$\mathbf{\Lambda}_t = \alpha_t \left[\sum_{\ell=0}^{\ell_{\max}} \beta_\ell \mathbf{I}_\ell^0 \left(\mathbf{I}_\ell^0\right)^T \right]. \tag{5.3.76}$$

and we denote

$$\mathbf{C} = \sum_{\ell=0}^{\ell_{\max}} \beta_\ell \mathbf{I}_\ell^0 \left(\mathbf{I}_\ell^0\right)^T . \tag{5.3.77}$$

The matrix $\mathbf{C}$ is SPD where positive definiteness follows since $\mathbf{I}_\ell^0 = \mathbf{I}$ (the unit matrix), $\beta_0 > 0$, and $\beta_\ell \geq 0$. The update rule of Algorithm 5.3 takes the form

$$\omega := \omega + \alpha_t \mathbf{C}\mathbf{z}d , \tag{5.3.78}$$

which is exactly as in (5.3.65). Convergence with probability 1 of Algorithm 5.3 follows directly from Proposition 2 part (a), and the solution is the same as TD(λ) applied at the finest level. Since Algorithm 5.3 and S-MGTD(λ) are equivalent, convergence with probability 1 of S-MGTD(λ) to the same solution follows.

■

5.4 A Few Words on On-Line Grid Construction

Throughout this chapter we assumed that the required inter-level operators are available beforehand. In this section we demonstrate how to generate such operators on-line through samples. This demonstration is provided for completeness only, since this is not the focus of our research.

We seek to apply the *setup* step of AMG based on the matrix $\mathbf{A}$ defined in (5.1.6) when the MDP model is not known. We remind that $\mathbf{A}$ can be sampled using (3.4.38)

$$\mathbf{A}_t = \sum_{\tau=0}^{t} \mathbf{z}_\tau \left(\phi\left(s_\tau\right) - \gamma\phi\left(s_{\tau+1}\right)\right)^T . \tag{5.4.79}$$

In principle, we can sample $\mathbf{A}_t$ before execution of the *solve* step, and directly implement the *setup* step to generate $\mathbf{A}_\ell$ at all levels based on $\mathbf{A}_t$. This approach is data inefficient since all samples obtained for the *setup* step are not used in the *solve* step. In the following we show how the *setup* step can be combined with the *solve* step, in LB-MGTD(λ) (Algorithm 5.1).

We suggest to initialize $\mathbf{A}_{\ell,0} = 0$, and update $\mathbf{A}_{\ell,t}$ during the execution of the TD(λ) iterations at level ℓ (step 2) according to

$$\mathbf{A}_{\ell,t} := \mathbf{A}_{\ell,t-1} + \mathbf{z}_{\ell,t}\left(\phi_\ell\left(s_t\right) - \gamma\phi_\ell\left(s_{t+1}\right)\right)^T. \tag{5.4.80}$$

Before moving to the coarser grid $\ell+1$ (step 3.2), we apply the *setup* step based on the sampled $\mathbf{A}_{\ell,t}$ to generate $\mathbf{I}_{\ell+1}^{\ell}$, $\mathbf{I}_{\ell}^{\ell+1}$, $\phi_{\ell+1}\left(s\right)$ and initialize

$$\mathbf{z}_{\ell+1,t} := \left(\mathbf{I}_{\ell+1}^{\ell}\right)^T \mathbf{z}_{\ell,t}, \tag{5.4.81}$$

$$\mathbf{A}_{\ell+1,t} := \mathbf{I}_{\ell}^{\ell+1}\mathbf{A}_{\ell,t}\mathbf{I}_{\ell+1}^{\ell}. \tag{5.4.82}$$

where in (5.4.81) we used (5.3.51). In this way, the *setup* step uses all the information gathered until the time in which the inter-level operators for the next coarse level are required.

A deficiency of this particular algorithm is the resources needed to store $\mathbf{A}_{0,t}$, which for practical problems with many features may be infeasible. We remind that LSTD(λ) and λ-LSPE use $\mathbf{A}_{0,t}$ as well and therefore suffer the same deficiency. However, in contrary to these LS algorithms, the *setup* step does not require $\mathbf{A}_{0,t}$ to be exact, since it is only used to locate the strongest dependencies between variables. We therefore suspect that more efficient algorithms for the *setup* step can be derived. Some indication for this can be found in the algorithm presented in [KA99] that clusters states based on temporal neighborhood without explicitly generating $\mathbf{A}_{0,t}$. The derivation of such algorithms is left for future work.

Chapter 6

Experiments and Results

In this chapter we bring experimental results on the previously proposed algorithms compared to traditional ones. We start by describing two test bed problems, as well as some notations and conventions used throughout the chapter. We present results for the known model case (chapter 4), starting with the problem of Policy Evaluation (section 4.1) and followed by the problem of seeking the optimal policy using Modified Policy Iteration (section 4.2). The next section considers the learning scenario (chapter 5), and shows results on the performance of LB-MGTD and S-MGTD (sections 5.2-5.3). The last section concludes our observations.

6.1 Problem Definitions

6.1.1 A simple example: 1-D random walk

The random walk problem described in Figure 5.1, consists of a *Markov* chain with N states, ordered on a 1-D line and indexed $s \in \{1,2,\ldots,N\}$. The transition probability from any inner state to its right or left neighbor is 0.5 and the transition probability from an edge state to its inner neighbor is 1, i.e.

$$p(s'=j+1 \mid s=j) = p(s'=j-1 \mid s=j) = 0.5, \qquad \forall j \in \{2,3,\ldots N-1\} \tag{6.1.1}$$

$$p(s'=2 \mid s=1) = p(s'=N-1 \mid s=N) = 1 \tag{6.1.2}$$

where s is the current state and s' is the following state. The cost inflicted upon transition from any inner step is -1, while a reward of +10 is provided when "bouncing" from an edge state. The discount factor is chosen such that the total discount from edge to edge is 0.5, i.e.

$$\gamma = 0.5^{\frac{1}{N-1}} \tag{6.1.3}$$

The problem of policy evaluation is that of efficiently obtaining the value function for the infinite horizon discounted case. This function is given in Figure 6.2. This simple problem is similar to the hop-world problem in [XHH02]. These problems are helpful for testing scalability issues, because the propagation time of information from one end to the other depends on N.

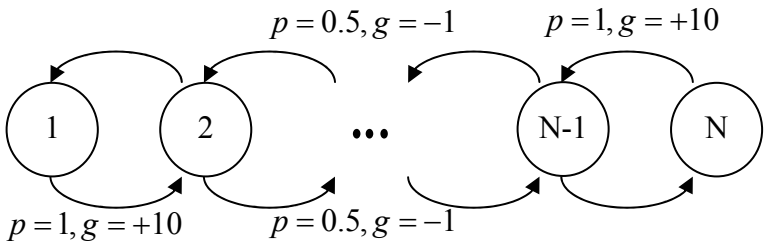


Figure 6.1: 1-D random walk Markov chain

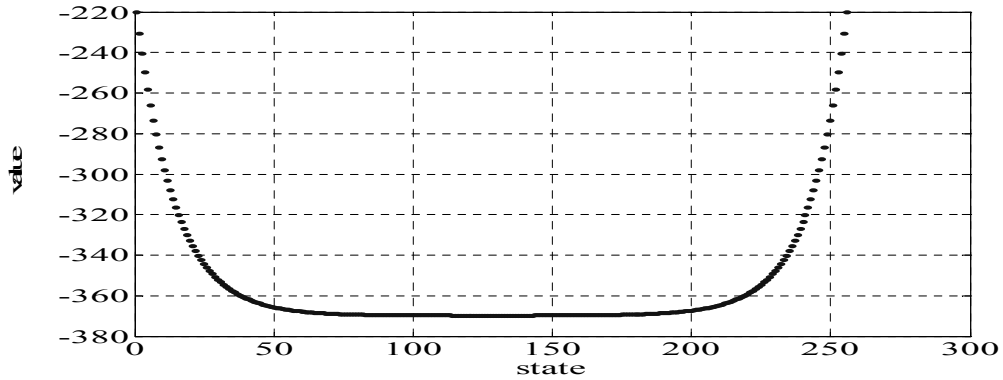


Figure 6.2: 1-D random walk value function for *N*=256 states.

6.1.2 The Mountain Car application

A more complicated test bed we use is the popular mountain car task [S96] [MA95] [MM02]. A car positioned at the bottom of a valley, is required to reach the top of the mountain at exactly zero speed (see Figure 6.). Since its engine is too weak, it cannot drive straight up the mountain, but has to back up to gain momentum. A reinforcement of +1 is received on reaching the goal on the right at zero speed. Reaching the goal at any other speed reduces the reward linearly until a penalty of -1 inflicted for the maximal speed. A penalty of -1 is also inflicted on reaching the left border. The action space consists of either accelerating to the right or to the left at a constant thrust. This is a continuous time, continuous 2-D space problem (horizontal position and velocity). We used the discretization scheme described in [MM02] to derive a discrete time discrete space MDP that approximates the problem. We used a 100×100 uniform grid in both horizontal position and velocity, resulting with a 10,000 state MDP. Technical details of the problem, the discretization scheme, and parameter values are provided in [MM02]. For further detail on the dynamics of this problem, see [MA95]. Figure 6.4 shows the optimal value and policy on a 100×100 grid.

We use the discrete mountain car problem to examine practical issues. We intentionally withhold the algorithm from using any geometrical information from the continuous problem, making the discrete problem seem unstructured and therefore making it hard for a developer to define good aggregates beforehand. In this way, we can test how AMG works on problems without foreseen natural aggregations. We then observe whether the setup phase manages to automatically construct grids that correlate the unused geometrical structure. Finally, we note that this problem is non-symmetric, enabling us to test the effects of violating the symmetry assumption.

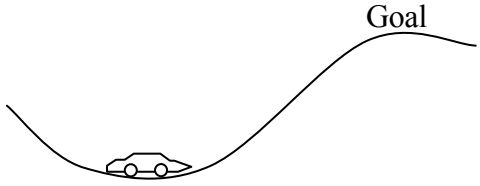


Figure 6.3: Mountain Car Task

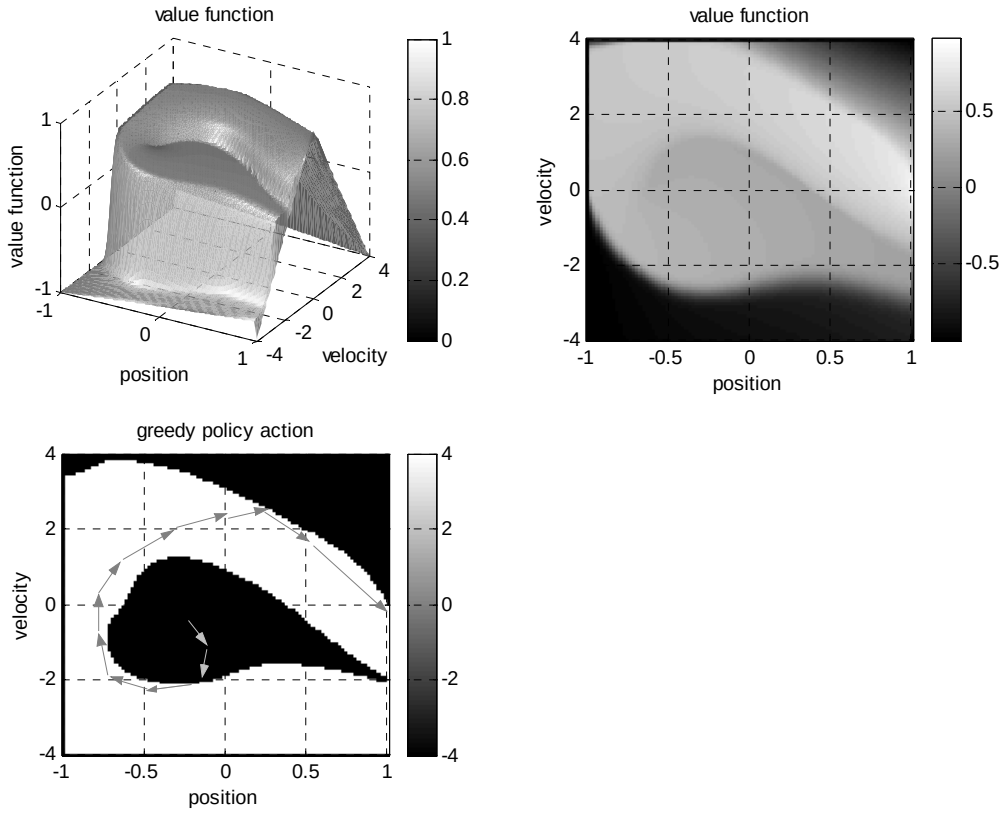


Figure 6.4: Value function and optimal policy for the mountain car task. top: views of the optimal value function on a 100×100 grid. bottom: optimal control - accelerate to the right (white), accelerate to the left (black). The gray arrows represent an optimal trajectory.

6.2 Notations and Conventions

Throughout the chapter we use the following abbreviations: Value Iteration smoothing (VI) (eq. (3.2.15), Gauss-Seidel smoothing (GS) (section 2.2.5), Ruge-Stüben interpolation (RS) (eq. (3.7.69)) and strict aggregation interpolation (aggr) (eq. (3.7.64)).

6.2.1 Known model case - technical details for sections 6.3-6.4

Throughout the following sections we present convergence curves of the L_2 norm of the residual error, $\left|\mathbf{g}-\left(\mathbf{I}-\gamma\mathbf{P}\right)\mathbf{v}_n\right|_2$. The curves are given as a function of the number of iterations

t, or as a function of the accumulated computational load. For Value Iteration (VI) and Gauss-Seidel (GS) a single iteration is one pass through all the states. For AMG based algorithms a single iteration consists of an entire fine to coarse and back to fine pass (V-cycle). In order to fairly compare methods of different complexities per iteration, we measure convergence in terms of math operations, either summation or multiplication, required to attain a certain level of accuracy. For ease of presentation we introduce a *computational unit*. In the context of Policy Evaluation, one computational unit equals the number of math operations in a single VI iteration. In the context of Policy Iteration, one computational unit equals the number of operations in a single policy improvement step. We used code to count math operations, as known bounds are too loose for our purposes.

We used Gauss-Seidel as the smoothing method for all AMG algorithms as it proved better than Jacobi or Richardson. We chose the number of pre- and post- iterations so as to maximize the convergence rate. Intermediate numbers of 2-3 iterations were found to be best; where performing enough iterations is required to remove local errors, while performing too many iterations is not cost-effective. However, during simulations we observed that the performance was rather insensitive to the exact choice of parameters.

For ease of readability we often display the same results side by side, using linear scale (left side) and logarithmic scale (right) for the horizontal axis.

6.2.2 Learning case - technical details for section 6.5

In the learning case, we used the 1-D random walk Markov chain with $N = 256$ states, the algorithm parameter $\lambda=0$ and a constant step size of $\alpha_t = 0.1$. Since a learning algorithm uses stochastic samples, it produces a random process for an output. To produce convergence curves we average over $M=5$ independent Monte-Carlo executions. We first generate M independent trajectories of the Markov chain, with the initial state chosen randomly according to the stationary distribution. We then apply each trajectory $m \in \{1,2,\ldots,M\}$ to each of the algorithms under investigation, initialized by $\theta_0 = \mathbf{0}$, and obtain its estimate of the parameter

vector at time *t*, denoted by $\theta_t^{(m)}$. We calculate the asymptotic parameter vector by solving (3.4.29) and denote it by θ_∞. We estimate the mean bias of $\theta_t^{(m)}$, denoted by $\theta_{bias,t}$, as

$$\theta_{bias,t} = \frac{1}{M}\sum_{m=1}^{M}\theta_t^{(m)} - \theta_\infty \approx \mathbb{E}\left(\theta_t - \theta_\infty\right). \tag{6.2.4}$$

We do not plot the bias $\theta_{bias,t}$ as a function of iterations since it is a vector. Instead we plot its L_2 norm $\left|\theta_{\text{bias,t}}\right|_2$ as a measure of distance from asymptotic solution. We note that while $\theta_{bias,t}$ is an unbiased estimator of $\mathbb{E}\left(\theta_t - \theta_\infty\right)$, $\left|\theta_{\text{bias,t}}\right|_2$ is a biased estimator of $\left|\mathbb{E}\left(\theta_t - \theta_\infty\right)\right|$. The bias is negligible if the variance of $\theta_{bias,t}$ is small compared to its mean.

6.3 Policy Evaluation - known model case

6.3.1 Results for the 1-D random walk problem

In this section, we observe the convergence behavior of different policy evaluation methods applied to the 1-D random walk problem. Figure 6.5 shows convergence curves of standard and AMG based methods when using 6 levels. Figure 6.6 and Figure 6.7 show the dependence of convergence rates on the number of grid levels and on the state space size respectively.

As expected, the asymptotic convergence of all methods is exponential as indicated by the straight lines in Figure 6.5. Figure 6.6 shows that the number of computational units required to reach a residual error norm of 10^{-10} when using 6 levels are: 38203 for VI, 19103 for GS, 1174 for AMG:aggr, and 23 for AMG:RS. Gauss-Seidel converges two times faster than VI. However the AMG methods converge considerably faster than GS; using strict aggregation results in 16 times faster convergence, and using Ruge-Stüben results in 830 times speedup relative to GS. For this specific test bed, the Ruge-Stüben interpolation proved much better than interpolation based on strict aggregation, requiring only 3 iterations versus 130 to converge. The reason lies in larger interpolation errors in the strict aggregation

method, which inject algebraically smooth errors in the correction step that slow down convergence. This is also indicated in Figure 6.6 on the left by the increase in iterations to the target residual as the number of levels increases, for AMG with aggregation. This increase is expected since increasing the number of levels replaces the exact solution at level ℓ with the approximate result of a V-cycle from level ℓ onward. Interpolation errors accumulate causing a decrease in convergence rate. On the other hand, the exact solution on the coarsest grid is costly. Figure 6.6 on the right shows that the computational load required to reach the target residual, decreases with the number of grid levels. As the number of level increases the coarsest grid equations contain fewer variables, decreasing its computational cost. The additional cost of inter-level operators and iterations at inter-levels is small for coarse levels. The total cost of inter-level operators and iterations has generally the same complexity as a single iteration at the finest level.

Figure 6. shows the effort required to reach a residual of 10^{-10} as a function of the number of states. We used $\log_2(N)-2$ levels where N is the number of states, such that the number of variables at the coarsest level is similar for all problems. Notice that the number of operations per computational unit increases linearly with the number of states. The effort measured either in iteration count or computational units, to reach the target residual increases linearly with the number of states in all methods. However, the rate of increase for the AMG methods is slower than for the standard methods. The increase factor in computational load from a 128 state to a 2048 state problem is: 271 for VI, 270 for GS, 66 for AMG:aggr, and 13 for AMG:Ruge-Stüben. These numbers were obtained by multiplying the number of computational units in Figure 6.5 by the number of states, since the computational unit is proportional to the number of states. The increase factor for the AMG methods is considerably smaller than for standard methods, which suggests that AMG may be better suited for problems with more than one dimension, for which the number of states increases exponentially with the number of features.

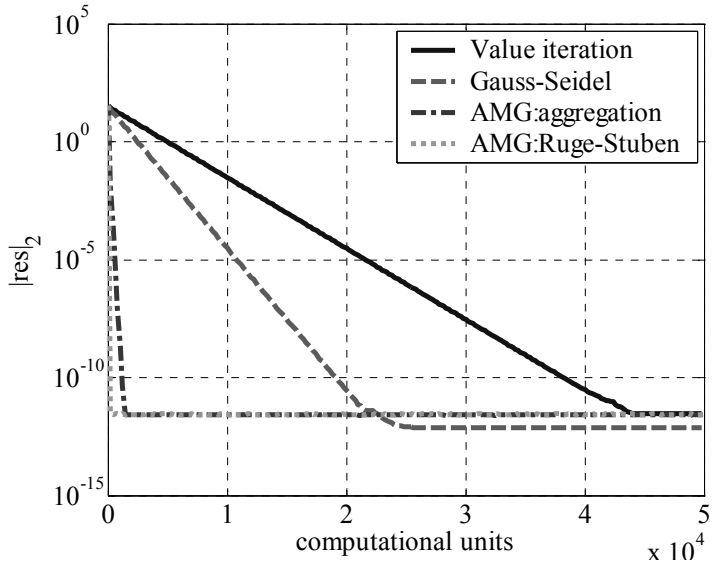


Figure 6.5: Convergence curves for the 1-D random walk problem. *N*=1000 states. AMG methods use 6 grid levels, 1 pre- and 1 post-iteration.

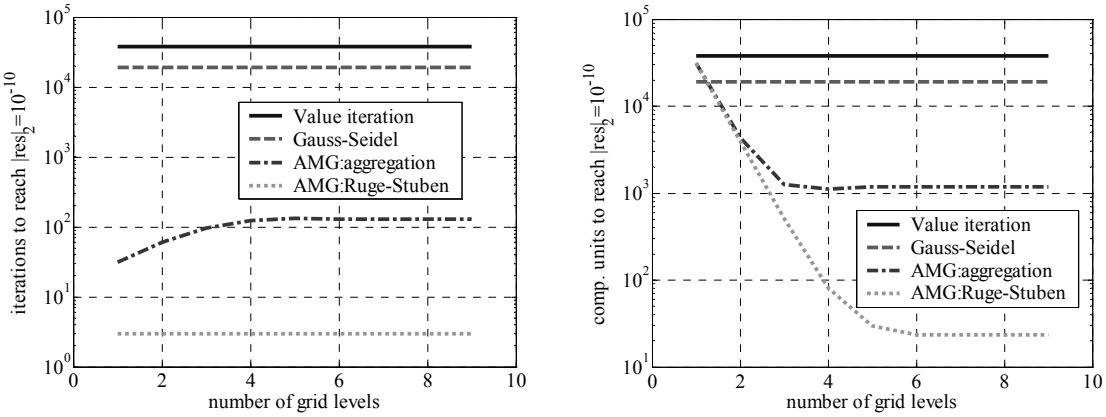


Figure 6.6: Computational effort in iterations(left)/computational units(right) to reach a residual of 10^{-10} as a function of the number of grid levels.
The number of iterations/computational units to reach $|res|_2=10^{-10}$ are: 38203 for VI , 19103 for GS, 130/1174 for AMG:aggr, and 3/23 for AMG:RS, when using 6 levels.

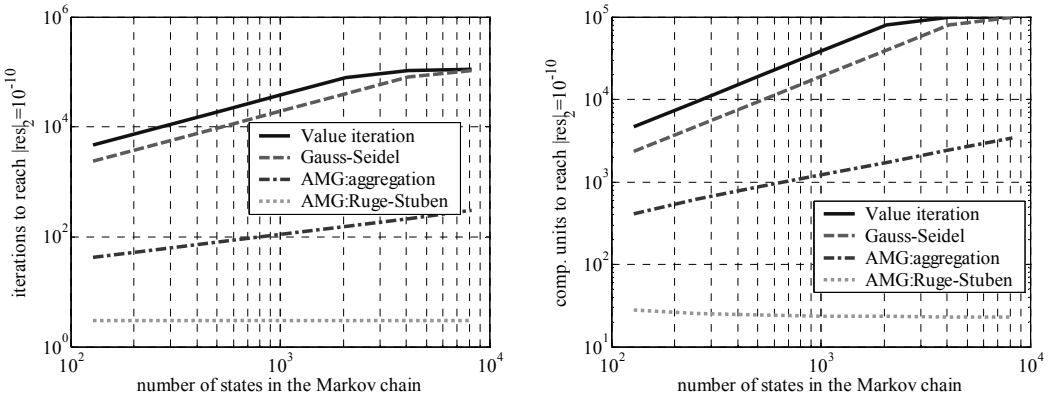


Figure 6.7: Computational effort in iterations(left)/computational units(right) to reach a residual of 10^{-10} as a function of the number of states.

6.3.2 Policy evaluation in the mountain car problem

In this section, we compare policy evaluation methods for the mountain car problem. Figure 6.8 and Figure 6.9 show convergence curves for this problem, using a 100×100 and 30×30 discretization grids respectively. As in the previous simple problem, AMG:aggr and AMG:RS converge faster than VI or GS. However, this time AMG:aggr converges faster than AMG:RS. The advantage of AMG:aggr is a result of its computationally cheap interpolation relative to the Ruge-Stüben interpolation, which makes it more cost effective. Notice that AMG:aggr and AMG:RS converge though the problem is not symmetric. In section 4.1.4 we presented the Kaczmarz and the least-squares (LS) problem formulations which guarantee convergence even in non-symmetric problems. Convergence curves for these methods are given in Figure 6.9. We note that we used a 30×30 grid configuration since running these methods on a 100×100 grid was too resource consuming. Though these methods guarantee convergence, they are slow to converge: AMG:LS is more than 8 times slower than AMG:aggr, and AMG:Kaczmarz does not reach the target residual within the simulation time. The reason for the relative slow convergence rates of AMG:LS and AMG:Kaczmarz is that the imposed symmetry is obtained at the expense of an increase in the condition number of the equation system to be solved (see section 4.1.4).

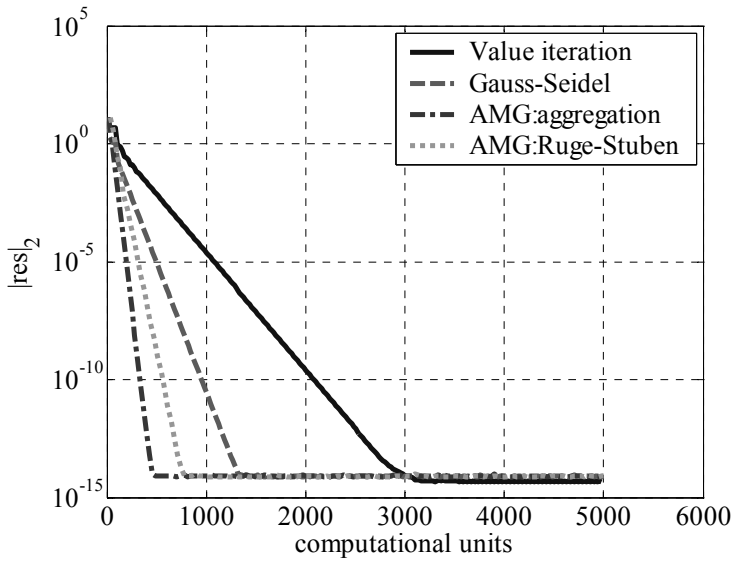


Figure 6.8: Convergence curves for the mountain car problem on a 100×100 grid.
AMG methods use 9 grid levels with 3 pre- and 2-post iterations. The number of comp. units to reach $|res|_2=10^{-10}$ are: 2081 for VI, 952 for GS, 340 for AMG:aggr, and 566 for AMG:RS.

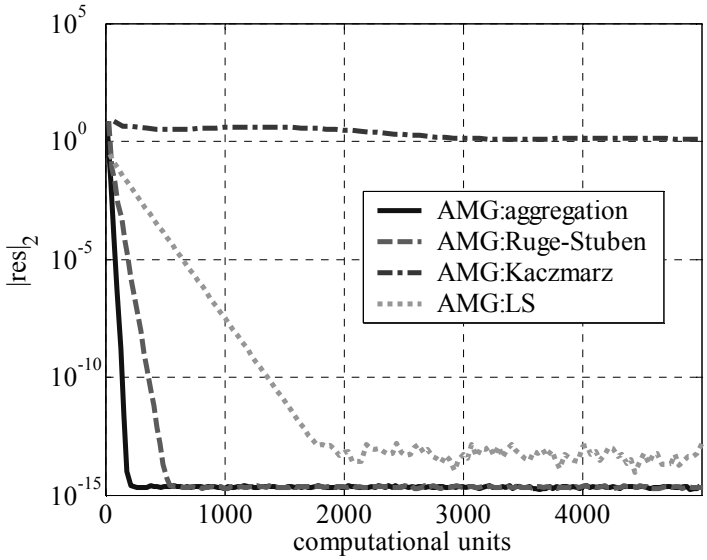


Figure 6.9: Convergence curves for the mountain car problem generated on a 30×30 grid.
AMG methods use 6 grid levels with 3pre and 2post iterations. The number of comp. units to reach $|res|_2=10^{-10}$ are: 153 for AMG:aggr, 369 for AMG:RS, and 1360 for AMG:LS. AMG:Kaczmarz did not reach the target residual within 5000 comp. units.

6.3.3 What do grids look like?

In the coarsening process, the points of level $\ell+1$ are a subset of the points of level ℓ. Figure 6.10 and Figure 6.12 show the selected grid points in each level. Large points belong to coarse and finer grids as well. Small points belong to fine grids only. Lines connecting points mark a high transitional probability between the states. Figure 6. and Figure 6.13 represent the basis functions generated at each level. In order to visualize these functions we define *influence regions*. In the correction step, coarse grid points are interpolated to the finer level, thus influencing their value. We assign each point in a fine level to its most influencing coarse level point. This assignment partitions the state space into *influence regions*, which we present in Figure 6. and Figure 6.13. Each influence region is given an arbitrary gray level for better visualization.

For the 1-D problem, the setup phase generated a uniform aggregation as shown in Figure 6.10, which may be regarded as natural for this problem. The basis function hierarchies in Figure 6., reveal that correction is done in blocks clustered by state adjacency, with global errors eliminated on the coarsest grid, and local errors on fine grids. In the mountain car problem, the setup phase generated a grid hierarchy shown in Figure 6.12. A high probability trajectory is shown in Figure 6.4 above. Comparing these figures reveals that aggregation is formed along high probability trajectories. A closer look in Figure 6.12 shows uniform aggregation along these trajectories. The variables of coarse levels are automatically chosen with equal spacing along such trajectories, with the basis functions shown in Figure 6.13 (left) aligned. This makes sense since states lying on the same trajectory are susceptible to suffer the same global error, and should therefore share a similar correction. We refer the interested reader to [MMS02] in which influence traces are defined in the discretization process. These traces are similar to the basis functions formed here.

Figure 6.13 shows the basis functions for two different policies: the optimal policy and a random policy in which right or left accelerations are chosen with equal probability. The differences demonstrate the dependency of the generated basis functions on the policy. We note that during our simulations we observed divergence when using basis functions of one policy to estimate the other. This stresses the importance of dynamically adapting the basis functions to match policy changes.

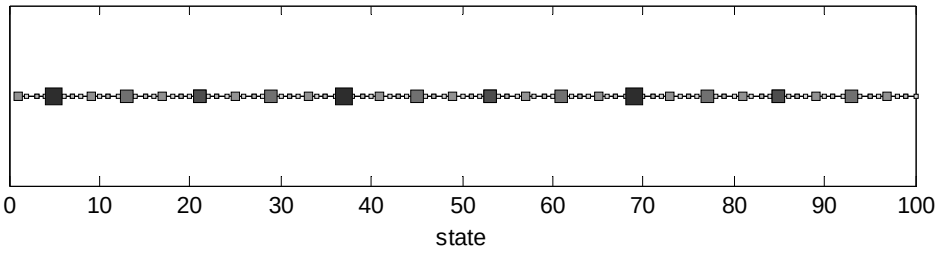


Figure 6.10: Grid points for the 1-D random walk problem.

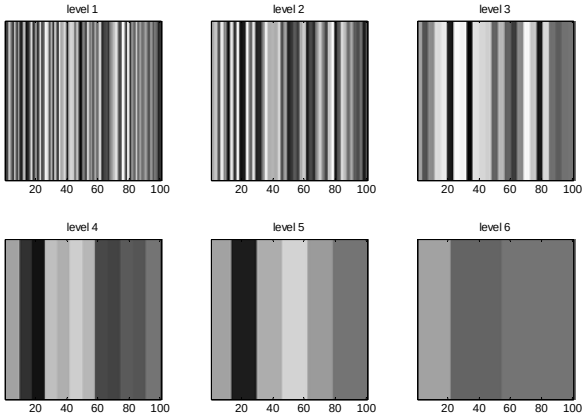


Figure 6.11: Basis functions for the 1-D random walk problem at different grid levels.

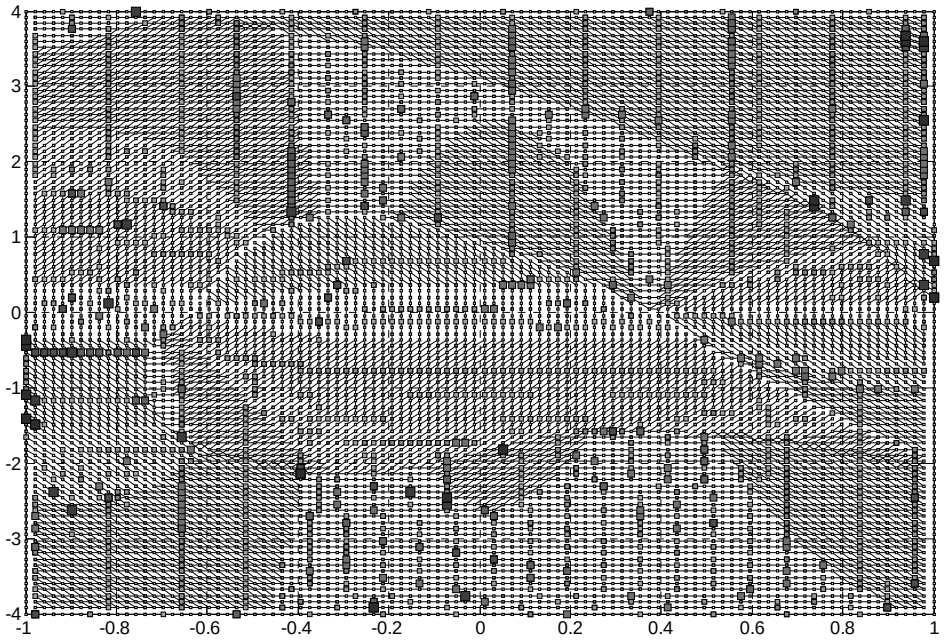


Figure 6.12: Grid points for the mountain car problem using the optimal policy.

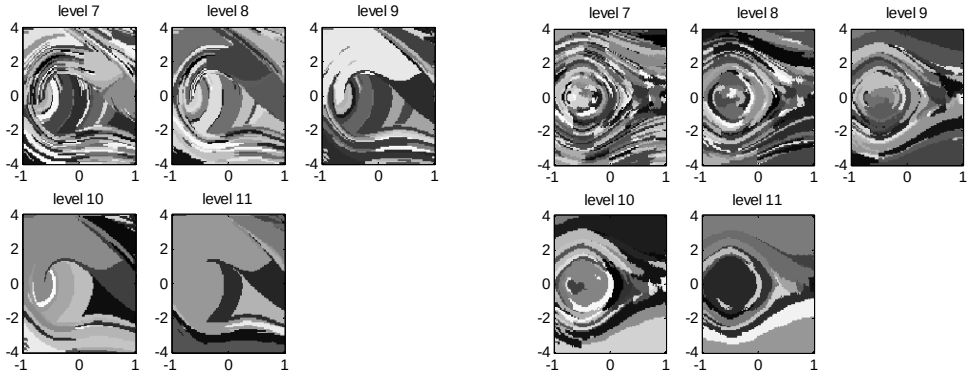


Figure 6.13: Basis functions for the mountain car problem for the optimal policy (left) and for a random choice policy (right).

6.4 Modified Policy Iteration

In this section, we seek the optimal policy of the mountain car problem. We compare convergence curves of different settings of Algorithm 4.3 and the classic VI and PI algorithms. The direct solution of (3.2.7) required in the evaluation step of PI is computationally very demanding. We therefore approximate its solution by using 100 VI iterations, and present the performance of PI as a function of iterations only, and not the computational load. The modified PI algorithm was implemented using 4 iterations in the policy evaluation step, taking its computational load into account. We use two measures to assess the convergence behavior: the percentage of states in which the greedy policy differs from the optimal one (Figure 6.14), and the norm of the residual error (Figure 6.15). Convergence curves of these measures are shown as functions of either computational load or the number of iterations.

In computational terms, AMG:aggr reaches the optimal policy about 9 times faster than VI and more than 5 times faster than modified PI. Though AMG:RS takes about the same number of iterations to reach optimal policy as AMG:aggr, it is two times slower in computational terms. The reason is that RS interpolation is more computationally expensive than interpolation based on aggregation. Observe that both AMG based methods and Policy Iteration take about 20 iterations to reach optimal policy. The fact that the number of policy im-

provement steps is the same suggests that policy evaluation using a single V-cycle iteration provided a solution close enough to the true value so that the policy improvement step is optimal. Similar results are obtained for the second convergence measure (Figure 6.15). AMG based methods reach the target residual an order of magnitude faster than the standard methods in computational terms, and as fast as PI in terms of the number of policy improvement steps.

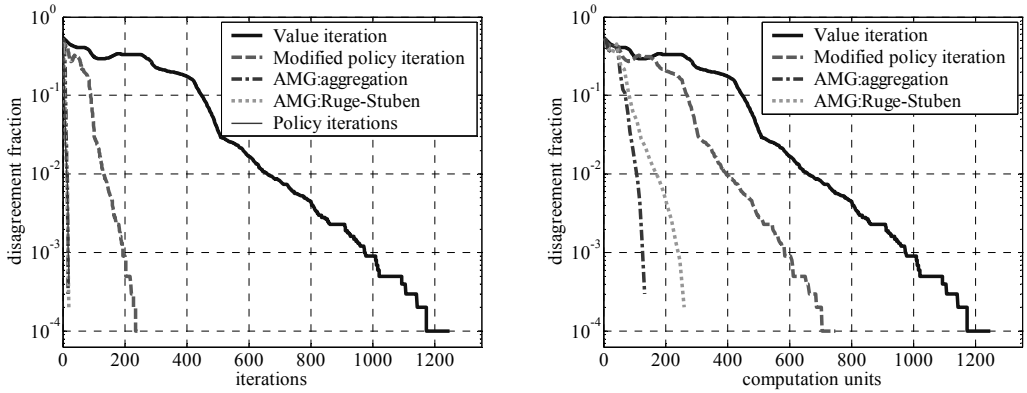


Figure 6.14: Curves of the fraction of states in which policy differs from the optimal, in the mountain car problem.
The number of iterations/comp. units to reach optimal policy is: 1247 for VI, 250/750 for Modified PI, 18/140 for AMG:aggr, 20/275 for AMG:RS, and 21/- for PI.

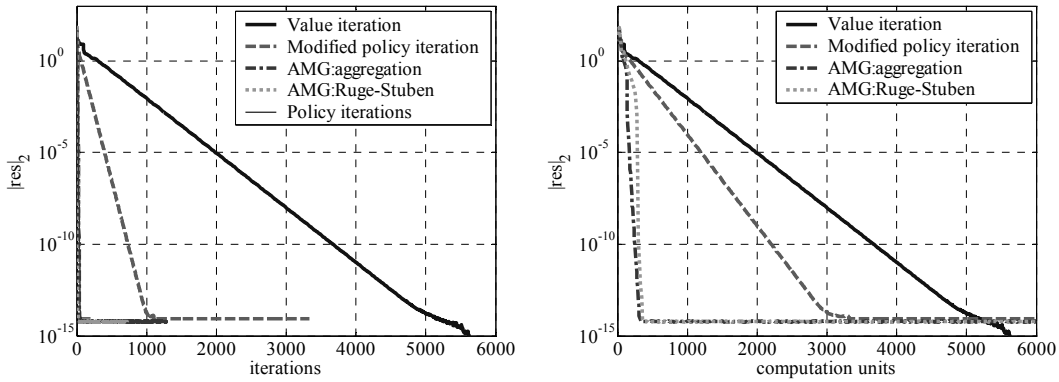


Figure 6.15: Curves of the residual error norm, in the mountain car problem.
The number of iterations/comp. units to reach a residual of 10^{-10} is: 3675 for VI, 735/2205 for Modified PI, 32/250 for AMG:aggr, 22/308 for AMG:RS, and 38/- for PI.

6.5 Multigrid Temporal Difference Algorithms

In this section we present results on Multigrid TD learning algorithms. Throughout this section our test bed is the 1-D random walk problem with *N*=256 states. We use $\lambda = 0$ and the learning step sequence $\alpha_t = 0.1$ for all algorithms. For Multigrid algorithms, we use the *setup* step (see section 3.7) to generate a hierarchy of feature vectors, $\phi_\ell(s)$ prior to the algorithm execution.

6.5.1 Grid level convergence

Here we examine the convergence behavior of TD algorithms applied separately on different grid levels. We apply TD(0) at level ℓ by using $\phi_\ell(s)$ as the feature vector, the reward from the finest grid and initializing $\theta_\ell = \mathbf{0}$. The convergence curves are shown in Figure 6.16. Convergence on coarser grids is faster than on finer grids, but the steady state error is larger. There is a tradeoff between accuracy achieved on fine grids and relatively high convergence rates on coarse grids. Our Multigrid TD algorithms apply TD at multiple grid levels taking advantage of this property: accuracy is obtained by fine grid iterations, while fast corrections of global nature are obtained at coarser levels.

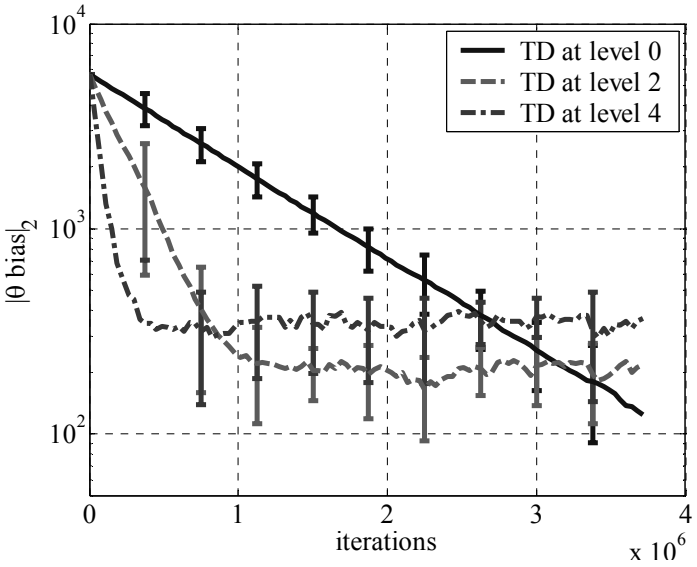


Figure 6.16: Curves of the bias in θ for TD(0) applied at different grid levels.

6.5.2 Results for Multigrid TD methods

In this section we bring results for several variations of LB-MGTD and for S-MGTD. We test two level schemes of LB-MGTD: a one-way bottom up scheme called grid-refinement and a V-cycle scheme (sections 3.6.3 and 3.6.2 respectively). We switch levels every 5,000 iterations in both schemes. Diagrams of the active level of these schemes are given in Figure 6.17. All algorithms in this section use a constant learning step of $\alpha_t = 0.1$, which was also the learning step for all levels in S-MGTD, i.e. $\alpha_{\ell,t} = \alpha_t$.

Figure 6.18 shows convergence curves with TD(λ) used as the coarsest grid solver. S-MGTD and LB-MGTD with a V-cycle scheme have similar convergence rate, which is considerably faster than for the other algorithms tested, specifically an order of magnitude faster than TD(λ). Among the LB-MGTD variations, V-cycle has the fastest convergence rate. Notice the ripples in the curve caused by changes in convergence rate following active level transitions. This effect does not occur in S-MGTD, since there are no explicit level changes. The grid-refinement scheme has a fast convergence rate at first, when error components of global nature are reduced. However, the convergence rate deteriorates after return-

ing to the fine grid, and becomes similar to the convergence rate of TD(λ). The reason lies in the slow convergence of smooth errors generated in the interpolation.

In section 5.2.4 we suggested to use fast TD solvers at the coarsest level. Figure 6.19 and Figure 6.20 show convergence curves with different coarsest level solvers for the grid-refinement and V-cycle schemes respectively. For the grid refinement scheme (Figure 6.19), all methods have the same asymptotic convergence rate, as expected, due to algebraically smooth errors caused in interpolation. However, LSTD(λ) and λ-LSPE applied on the coarsest level enabled a better initialization for TD(λ) on finer levels. The finest level reached a mean bias of 4097 for TD(λ), 1408 for λ-LSPE, and 387 for LSTD, after 25,000 iterations. Since TD(λ) has a slow converges rate the improved initialization enabled to reach an initial bias that standard TD(λ) doesn't reach within 750,000 iterations. Though LSTD achieved better initialization than λ-LSPE, its value was undefined during the first iterations, until the matrix $\mathbf{A}_t$ became non-singular. Its residual changed discontinuously from the initial residual of about 6000 to 320. Since the residual is actually a discrete time random process, we use the continuity term loosely, to say that the residual of the ODE which is a continuous time random process is continuous (see section 5.1). On the other hand, by initializing $\mathbf{B}_{t=-1}^{-1} = \delta \mathbf{I}$ with $\delta = 10$ (see (3.4.42)), λ-LSPE had a well defined solution at all times, and its residual changed gradually. We remind that the update rule of λ-LSPE averages its previous solution with a new least squares solution, weighted by the step size. This enables to control the tradeoff between smoothness (high bias) and fast adaptivity to changes (high variance). The choice of parameters was done to highlight the difference between λ-LSPE and LSTD(λ). We note that a similar tradeoff control exists in the efficient implementation of LSTD(λ) called RLS-TD(λ). The choice of parameters affects only the short term behavior, since LSTD and λ-LSPE converge to each other faster than to the convergence point (this is proven in [NB03]).

In the V-cycle scheme (Figure 6.20) the use of LSTD(λ) and λ-LSPE as coarse grid solvers helps to obtain a fast convergence rate during the first three V-cycle cycles (150,000 iterations). During the next cycles, the variance grows making our estimate of the convergence curve unreliable. Notice the spikes when using LSTD, caused by the high initial vari-

ance of LSTD(λ), versus the smoothed version achieved by λ-LSPE. Eventually the scheme which uses TD(λ) as the coarse grid solvers catches up and offers a solution with lower variance. The high variance in the coarsest level results from driving the coarse grid solvers with the sampled residual as a one step reward. The sampled residual, which we remind to be $r_{\ell+1}(s_t, s_{t+1}) = r_\ell(s_t, s_{t+1}) - \left(\phi_\ell(s_t) - \gamma\phi_\ell(s_{t+1})\right)^T \theta_\ell^-$ (see (5.2.22)), is a sum of random variables, and therefore its variance intends to increase at coarser levels. LSTD and λ-LSPE happen to be more sensitive to this additional noise than TD(λ). This negative affect can be reduced in λ-LSPE by changing the algorithm parameters, i.e. smaller step size and initializing $\mathbf{B}_{t=-1}^{-1} = \delta\mathbf{I}$ with larger δ.

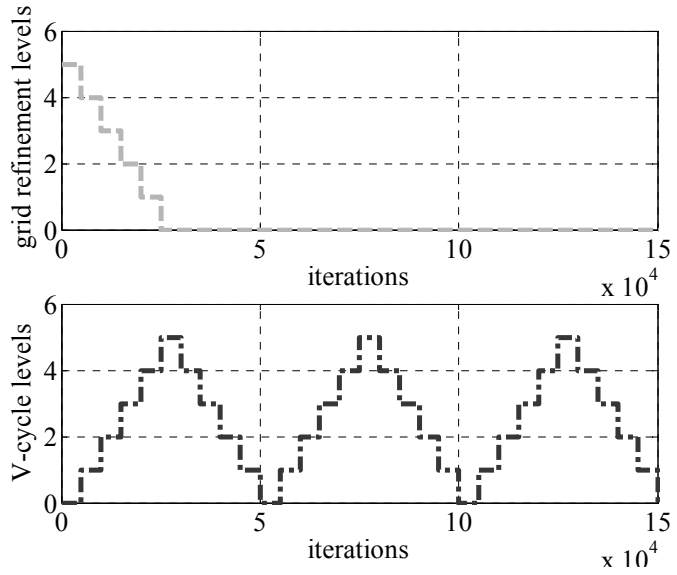


Figure 6.17: Active grid for the grid refinement (top) and V-cycle (bottom) schemes.

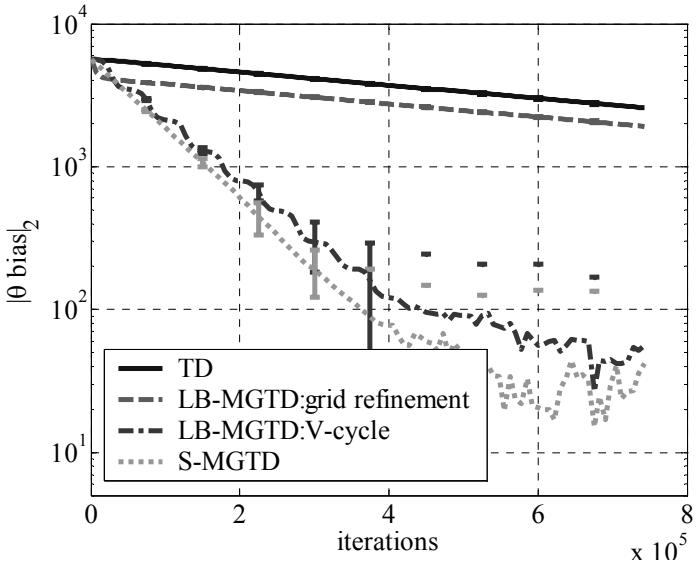


Figure 6.18: Curves of different Multigrid TD algorithms. TD(λ) is used as the coarsest level solver.
The number of iterations[×1000] to reduce the bias norm by half is: 654 for TD, 370 for LB-MGTD:GR, 76 for LB-MGTD:V-cycle, and 62 for S-MGTD.

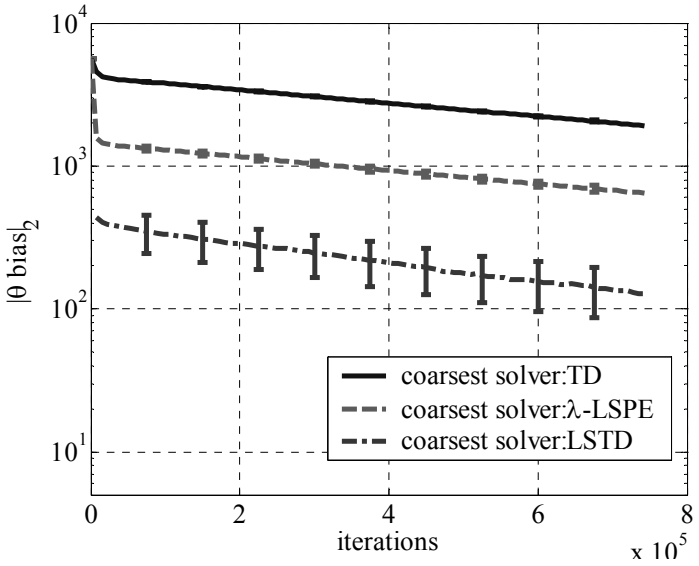


Figure 6.19: Curves of LB-MGTD with a grid refinement scheme for different coarsest level solvers.

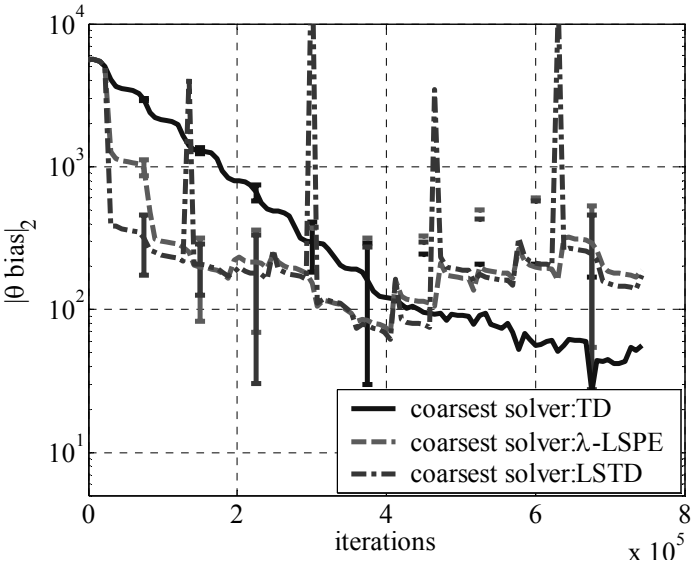


Figure 6.20: Curves of LB-MGTD with a V-cycle scheme for different coarsest level solvers.

6.6 Concluding Remarks

The results in section 6.3 show that incorporating AMG with policy evaluation brings advantages in terms of accelerated convergence and less sensitivity of the convergence rate to the problem size. This motivated the incorporation of AMG with policy iteration, which is shown to speedup convergence to the optimal policy in section 6.4. The results of section 6.3 also motivated the incorporation of AMG ideas in the learning case. We observed that the Multigrid-TD algorithm converges faster than the standard TD algorithm. The grid-refinement, which is a bottom up only scheme, has an asymptotic convergence rate similar to TD, since interpolation injects algebraically smooth errors. The V-cycle scheme however has a faster convergence rate due to elimination of errors at multiple scales. Among the coarse grid solvers used, we observe that while LSTD and λ-LSPE are faster to converge than TD, they are more susceptible to noise, increasing the variance of the solution. It is therefore preferable to use a fast coarsest grid solver during the initial few cycles, and then switch to TD(λ) in consecutive cycles. Other options are to change the parameters of λ-LSPE to incor-

porate more smoothing as the number of iterations increase, along with reducing the step size α_t. We note that the criterion used for switching levels is arbitrary, so that deriving better criteria should improve performance. This problem is less important in the S-MGTD algorithm, which shows significant improved convergence rate over the other algorithms examined.

Chapter 7

Conclusions

In this dissertation, the AMG approach was used to speed up policy evaluation for the known model case and in TD learning. Two new TD learning algorithms with a complexity per step similar to TD(λ), called LB-MGTD(λ) and S-MGTD(λ) were proposed. We showed how to incorporate LSTD(λ) and λ-LSPE as fast solvers of the coarsest level in LB-MGTD(λ). We prove separate convergence of each level for LB-MGTD(λ) providing rational for the scheme, and prove the convergence of S-MGTD(λ). Our convergence proof for S-MGTD(λ) is not limited to the symmetric case. This is surprising since the symmetry assumption is required by current theorems in order to guarantee convergence of AMG. This is a limiting requirement since in practical non-symmetrical problems using AMG offers considerable speedup only at a risk of divergence. In this sense S-MGTD(λ) is safer than LB-MGTD(λ) for non-symmetrical problems.

Experimental results on two test-bed problems demonstrate that AMG for policy evaluation can considerably speed up convergence. This speed up is used within a policy improvement step, demonstrating that the optimal policy can be reached with fewer mathematical operations. On a single problem the LB-MGTD and S-MGTD learning algorithms showed considerable speed up relative to TD(λ). During our experiments, we observed that increasing number of states in this problem makes the proposed algorithms even more superior to TD, suggesting that LB-MGTD and S-MGTD are less susceptible to scalability. This property is well known in Multigrid literature.

Experimental results on the incorporation of LSTD(λ) or λ-LSPE as coarsest level solvers in LB-MGTD(λ) showed considerable speedup when used in a grid refinement scheme or during the first cycles of the V-cycle scheme. However, these solvers showed increased susceptibility to noise increasing the variance in consecutive cycles. In practice it is therefore

favorable to use λ-LSPE as a coarsest level solver during the first few cycles, and then switch to TD(λ) at the coarsest grid. In our tests, we used a simple criterion to switch between levels. An improved criterion may estimate the residual convergence rate, and switch between levels when it deteriorates.

We have shown an equivalent formulation of the proposed S-MGTD as a TD(λ) variant with modified eligibility traces, in which the basis functions used in the eligibility traces are different than those used in the value function approximation. The derivation of S-MGTD from the Multigrid approach and the experimental results suggest that this may offer a considerable speed up. A proper choice for basis functions used for eligibility traces is a linear combination of interpolations of the original basis functions restrictions, where the interpolator is built to approximate algebraically smooth errors (see (5.3.66) and (5.3.77) in section 5.3.5).

In this dissertation we focused on the solve phase of AMG. In the known model case, the implementation of a setup phase that defines inter-level operators is straightforward, with many methods available in AMG literature (see references in [TOS01]). In the learning case, we focused on the development of learning versions of the solve phase, and assumed that inter-level operators are available beforehand. We offered a way to obtain these operators on-line by applying a standard setup procedure to an on-line estimate of the matrix **A** as done in the LSTD algorithm [NB03]. We note that the setup phase is considerably less susceptible to the accuracy in **A** than LSTD, holding the potential of using rough or even qualitive estimators to estimate only the significant elements of **A**. The derivation of such schemes is kept for future work.

The setup phase of AMG automatically generates hierarchies of aggregates, based on the notion of smooth error approximation. AMG theory and practical experience with setup procedures may be applied to address the open question of how to form aggregation hierarchies automatically.

Finally, we note that the algorithms we proposed are not limited to algebraic Multigrid, and may easily be extended to work with geometric Multigrid when ordered meshes are natural to the problem at hand.

Bibliography

[AC88] Akian M. and Chancelier J.P., Dynamic Programming Complexity and Application. *Proceedings of the 27th Conference on Decision and Control*, Austin, Texas, pp. 1551-1558, 1988.

[B02] Boyan J.A., Technical Update: Least Squares Temporal Difference Learning. *Machine Learning*, 49, pp. 233-246, 2002.

[B72] Brandt A., Multi-Level Adaptive Technique for Fast Numerical Solution to Boundary Value Problems. *In the 3rd International Conference on Numerical Methods in Fluid Mechanics*, 1972.

[B86] Brandt A., Algebraic Multigrid Theory: the Symmetric Case. *Appl. Math. Comp*. Vol. 19, pp. 23-56, 1986.

[B99] Bertsekas D.P. *Nonlinear Programming,* 2nd edition. Athena Scientific, 1999.

[BB99] Bradtke S.J. and Barto A.G., Linear Least-Squares Algorithms for Temporal Difference Learning. *Machine Learning*, 22:1-3, pp. 33-57, 1996.

[BBN03] Bertsekas D.P., Borkar V.S., and Nedić A., Improved Temporal Difference Methods with Linear Function Approximation. *Report LIDS-2573*, December 2003.

[BC89] Bertsekas D.P. and Castañon D.A., Adaptive Aggregation Methods for Infinite Horizon Dynamic Programming. *IEEE Transactions on Automatic Control*, Vol. 34, No. 6, 1989.

[BHMC00] Briggs W.L., Henson V.E., and McCormick S.F., *Multigrid Tutorial, Second Edition*. Philadelphia, SIAM, 2000

[BM03] Barto A.G. and Mahadevan S., Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems: Theory and Applications*, Vol. 13, pp. 41-77, 2003.

[BMP90] Benveniste A., Métivier M., and Prioret P., *Adaptive Algorithms and Stochastic Approximations*. Berlin: Springer-Verlag, 1990.

[BT95] Bertsekas D.P. and Tsitsiklis J.N., *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1995.

[CM82] Chatelin F. and Miranker W.L., Acceleration by Aggregation of Successive Approximation Methods. *Linear Algebra and its Applications*, Vol. 43, pp. 17-47, 1982.

[D00] Dietterich T.G. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research* 13, pp. 227-303, 2000.

[D98] Digney B., Learning Hierarchical Control Structure for Multiple Tasks and Changing Environments. *Proceedings of the Fifth Conference on the Simulation of Adaptive Behavior*, Cambridge, 1998.

[DH92] Dayan P. and Hinton G.E., Feudal Reinforcement Learning. *Advances in Neural Information Processing Systems (NIPS)* 5, pp. 271-278, 1992.

[F61] Fedorenko R.P., A Relaxation Scheme for the Solution of Elliptic Differential Equations. *UdSSR Comput. Math. Phys.* 1,5, pp. 1092-1096, 1961 (in Russian).

[FD02] Foster D. and Dayan F., Structure in the Space of Value Functions. *Machine Learning*, Vol. 49, pp. 325-346, 2002.

[FS02] Feinberg E.A. Shwartz A. *Handbook of Markov Decision Processes - Methods and Applications*. Kluwer Academic, Netherlands, 2002.

[H76] Hackbusch W., Ein Iteratives Verfaheren zur Schnellen Auflösung Elliptischer Randwertprobleme. *Technical Report* 67-12, Universität Köln, 1976.

[HA98] Heckendorn R.B. and Anderson C.W., A Multigrid Form of Value Iteration Applied to a Markov Decision Problem. *Technical Report* CS-98-113, Colorado state university, computer science department, 1998.

[KA99] Kretchmar R.M. and Anderson C.W., Using Temporal Neighborhoods to Adapt Function Approximators in Reinforcement Learning. *Proceedings of the International Work Conference on Artificial and Natural Neural Networks* (IWANN), Alicante, Spain, pp. 488-496, 1999.

[KXZ03] Kim H., Xu J., and Zikatanov L., A Multigrid Method Based on Graph Matching for Convection-Diffusion Equations. *Numerical Linear Algebra with Applications.* Vol. 10, pp. 181-195, 2003.

[L93] Long-Ji Lin., Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, Vol. 8 (3-4), pp. 293-321, 1992.

[MA95] Moore A.W. and Atkeson C., The Parti-Game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State Space. *Machine Learning*, Vol. 21, 1995.

[MC87] McCormick S.F., *Multigrid Methods, Frontiers in Applied Mathmatics.* SIAM, 2000. Chapter 4, Ruge J.W. and Stüben K., Algebraic Multigrid.

[MM02] Munos R. and Moore A., Variable Resolution Discretization in Optimal Control. *Machine Learning*, Vol. 49, pp. 291-323, 2002. Parameters for the Mountain Car problem can be found in http://www-2.cs.cmu.edu/~munos/variable/ .

[MMS02] Menache I., Mannor S., and Shimkin N. Q-Cut - Dynamic Discovery of Sub-Goals in Reinforcement Learning. *European Conference on Machine Learning*, pp. 295-306, 2002.

[MMS03] Menache I., Mannor S., and Shimkin N., Basis Function Adaptation in Temporal Difference Reinforcement Learning. To appear in the *Annals of Operations Research, special issue on the Cross Entropy method*. Revised: December 2003

[NB03] Nedić A. and Bertsekas D.P., Least Squares Policy Evaluation Algorithms with Linear Function Approximation. *Discrete Event Dynamic Systems: Theory and Applications*, Vol. 13, pp. 79-110, Kluwer Academic, Netherlands, 2003.

[P94] Puterman M.L., *Markov Decision Processes: discrete stochastic dynamic programming.* Wiley-Interscience, New York, 1994.

[S01] Stüben K., *An Introduction to Algebraic Multigrid*. Appendix in [TOS01], 2001.

[S88] Sutton R.S., Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, Vol. 3, pp. 9-44, 1988

[S96] Sutton R.S., Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. *Advances in Neural Information Processing Systems (NIPS)*, Vol. 8, pp. 1038-1044, MIT Press, 1996.

[SB98] Sutton R.S. and Barto A.G., *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[SD98] Singh S. and Dayan P., Analytical Mean Squared Error Curves for Temporal Difference Learning. *Machine Learning*, Vol. 32, pp. 5-40. 1998. Also in *NIPS* 9, pp. 1054-1060.

[SJJ95] Singh S.P., Jaakkola T., and Jordan M.I., Reinforcement Learning with Soft State Aggregation. *Advances in Neural Information Processing Systems*, Vol. 7, pp. 361-368, MIT Press, 1995.

[SPK84] Schweitzer P.J., Putterman M.L., and Kindle K.W., Iterative Aggregation-Disaggregation Procedures for Discounted Semi-Markov Reward Processes. *Operations Research*, Vol. 33, No. 3, pp. 589-605, 1985.

[T92] Tesauro G.J., Practical Issues in Temporal Difference Learning. *Machine Learning* Vol. 8, pp. 257-277, 1992.

[TOS01] Trottenberg U., Oosterlee C., and Schüller A., *Multigrid*. Academic Press, San Diego, 2001.

[TVR97] Tsitsiklis J.N. and Van Roy B., An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*. Vol. 42, No. 5, pp. 674-690, 1997.

[VH99] Henson V.E., An Algebraic Multigrid Tutorial. Presented at the *Ninth Copper Mountain Conference on Multigrid Methods*. UCRL-MI-133749. 1999.
http://www.llnl.gov/CASC/people/henson/presentations/amgtut.pdf

[W99] Wagner C., *Introduction to Algebraic Multigrid.* Course notes, University of Heidelberg, 1999.
http://www.iwr.uni-heidelberg.de/groups/techsim/chris/amg.pdf

[XHH02] Xin Xu, Han-gen He and Dewen Hu., Efficient Reinforcement Learning Using Recursive Least-Squares Methods. *Journal of Artificial Intelligence Research*, Vol. 16, pp. 259-292, 2002.