Foundations and Trends® in
Machine Learning
Vol. 1, No. 4 (2008) 403–565
© 2009 S. Mahadevan
DOI: 10.1561/2200000003

Learning Representation and Control in Markov Decision Processes: New Frontiers

By Sridhar Mahadevan

Contents

Foundations and Trends® in
Machine Learning
Vol. 1, No. 4 (2008) 403–565
© 2009 S. Mahadevan
DOI: 10.1561/2200000003

Learning Representation and Control in Markov Decision Processes: New Frontiers

Sridhar Mahadevan

Department of Computer Science, University of Massachusetts — Amherst, 140 Governor's Drive, Amherst, MA 01003, USA, mahadeva@cs.umass.edu

Abstract

This paper describes a novel machine learning framework for solving sequential decision problems called Markov decision processes (MDPs) by iteratively computing low-dimensional representations and approximately optimal policies. A unified mathematical framework for learning representation and optimal control in MDPs is presented based on a class of singular operators called Laplacians, whose matrix representations have nonpositive off-diagonal elements and zero row sums. Exact solutions of discounted and average-reward MDPs are expressed in terms of a generalized spectral inverse of the Laplacian called the *Drazin inverse*. A generic algorithm called *representation policy iteration* (RPI) is presented which interleaves computing low-dimensional representations and approximately optimal policies. Two approaches for dimensionality reduction of MDPs are described based on geometric and reward-sensitive regularization, whereby low-dimensional representations are formed by *diagonalization* or *dilation* of Laplacian operators. Model-based and model-free variants of the RPI algorithm are presented; they are also compared experimentally on discrete and continuous MDPs. Some directions for future work are finally outlined.

1

Introduction

In this section, we introduce the problem of representation discovery in sequential decision problems called Markov decision processes (MDPs), whereby the aim is to solve MDPs by automatically finding "low-dimensional" descriptions of "high-dimensional" functions on a state (action) space. The functions of interest include policy functions specifying the desired action to take, reward functions specifying the immediate payoff for taking a particular action, transition distributions describing the stochastic effects of doing actions, as well as value functions that represent the long-term sum of rewards of acting according to a given policy. Our aim is to illustrate the major ideas in an informal setting, leaving more precise definitions to later sections. The concept of a Laplacian operator is introduced, and its importance to MDPs is explained. The general problem of dimensionality reduction in MDPs is discussed. A roadmap to the remainder of the paper is also provided.

1.1 Motivation

A variety of problems of practical interest to researchers across a diverse range of areas, from artificial intelligence (AI) [117] to operations research (OR) [109, 110], can be abstractly characterized as "sequential decision-making." Namely, in all these problems, the task can be formulated in terms of a set of discrete or continuous set of states, in each of which a decision maker has to select one of a discrete set of actions, which incurs a reward or cost. The objective of the decision maker is to choose actions "optimally," that is, to compute a policy function that maps states to actions maximizing some long-term cumulative measure of rewards. Examples range from game-playing [132] and manufacturing [33] to robotics [81, 97], and scheduling [143]. MDPs [56, 110] have emerged as the standard mathematical framework to model sequential decision-making. A MDP is mathematically defined in terms of a set of states S; a set of actions A (which may often be conditionally defined in terms of choices available in the current state as A_s); a stochastic transition distribution $P^a_{ss'}$ describing the set of outcomes s' of performing action a in state s; and a payoff or "reward" function $R^a_{ss'}$. The optimization objective is to find a mapping or policy from states to actions that maximize some cumulative measure of rewards. Commonly used objective measures include maximizing the expected "discounted" sum of rewards (where rewards in the future are geometrically attenuated by powers of a fixed positive scalar value $\gamma < 1$), or maximizing the average reward or expected reward per decision. Crucially, the optimization goal takes into account the uncertainty associated with actions. Every policy defines a value function on the state space, where the value of a state is the sum of the immediate reward received and the expected value of the next state that results from choosing the action dictated by the policy. A MDP is typically solved through the well-known Bellman equation [110], which is a recursive equation relating the value of the current state to values of adjacent states.

Exact solution methods, such as linear programming [36], policy iteration [56], and value iteration [110], assume value functions are represented using a table (or more generally on a fixed set of *basis functions*). The complexity of these algorithms is typically polynomial (cubic) in the size of the discrete state space $|S|$ (or exponential in the size of any compact description of the state space). When the number of states is large or if the state space is continuous, exact representations become infeasible, and some parametric or nonparametric function approximation method needs to be used. For example, if the states S of a discrete MDP are enumerated from $s = 1, \ldots, s = |S|$, where $|S| = n$, then functions over this discrete state space can be viewed as vectors that lie in a Euclidean space $\mathbb{R}^{|S|}$. Most previous work on approximately solving large MDPs surveyed in books on *approximate dynamic programming* [109], *neuro-dynamic programming* [12], and *reinforcement learning* [129], assume that MDPs are solved approximately by a set of handcoded "features" or basis functions mapping a state s to a k-dimensional real vector $\phi(s) \in \mathbb{R}^k$, where $k \ll |S|$.

Popular choices of parametric bases include radial basis functions (RBFs), neural networks, CMACs, and polynomials. Concretely, a polynomial basis can be viewed as an $|S| \times k$ matrix, where the ith column represents the basis function $1, 2^i, 3^i, \ldots, |S|^i$. A radial basis function $\phi_k(s) = e^{-\frac{||s-s_k||^2}{2\sigma^2}}$, where σ is a scaling factor, and s_k is the "center" of the basis function. A value function V is approximated as a linear combination of basis functions, namely: $V \approx \Phi w$, where Φ is a matrix whose columns are the specified basis functions, and w is a weight vector. If the number of columns of Φ is $k \ll |S|$, then Φ can be viewed as providing a low-dimensional projection of the original value function $\in \mathbb{R}^{|S|}$ to a subspace $\in \mathbb{R}^k$.

It has long been recognized that traditional parametric function approximators, such as RBFs, may have difficulty accurately approximating value functions due to nonlinearities in a MDP's state space (see Figure 1.1). Dayan [35] and Drummond [40] have noted that states close in Euclidean distance may have values that are very far apart (e.g., two states on opposite sides of a wall in a spatial navigation task). A traditional parametric architecture, such as an RBF, makes the simplifying assumption that the underlying space has Euclidean geometry.

The same issues arise in continuous MDPs as well. Figure 1.2 shows a set of samples produced by doing a random walk in a 2D inverted pendulum task. Here, the state variables are θ, the pole angle, and $\dot{\theta}$, the angular velocity. Note that in this task, and in many other continuous control tasks, there are often physical constraints that limit the "degrees of freedom" to a lower-dimensional manifold, resulting in motion along highly constrained regions of the state space. Figure 1.2

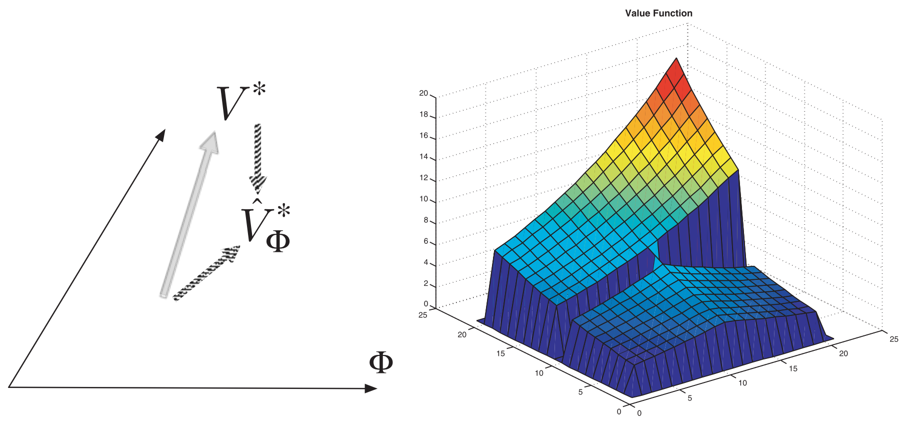


Fig. 1.1 *Left*: Dimensionality reduction of a MDP M involves finding a set of bases Φ such that any function on a MDP's state space, such as its optimal value function V^*, can be compressed effectively. *Right*: The optimal value function in a "two-room" discrete MDP with 400 states. The agent can take actions in the four compass directions. Each action succeeds with probability 0.9, otherwise leaves the agent in the same state. The agent is "rewarded" only for reaching a corner "goal" state. Access to each room from the other is available only through a central door, and this "bottleneck" results in a nonlinear optimal value function. This value function is ostensibly a high dimensional vector $\in \mathbb{R}^{400}$, but can be compressed onto a much lower-dimensional subspace.

also shows an approximation to the optimal value function constructed using a linear combination of "proto-value" basis functions [77], or eigenfunctions obtained by diagonalizing a random walk operator on a graph connecting nearby samples.

Both the discrete MDP shown in Figure 1.1 and the continuous MDP shown in Figure 1.2 have "inaccessible" regions of the state space, which can be exploited in focusing the function approximator to accessible regions. Parametric approximators, as typically constructed, do not distinguish between accessible and inaccessible regions. The approaches described below go beyond modeling just the reachable state space, in that they also build representations based on the transition matrix associated with a specific policy and a particular reward function [103, 106]. By constructing basis functions adapted to the nonuniform density and geometry of the state space, as well as the transition matrix and reward function, the approaches described in this paper are able to construct adaptive representations that can outperform parametric bases.

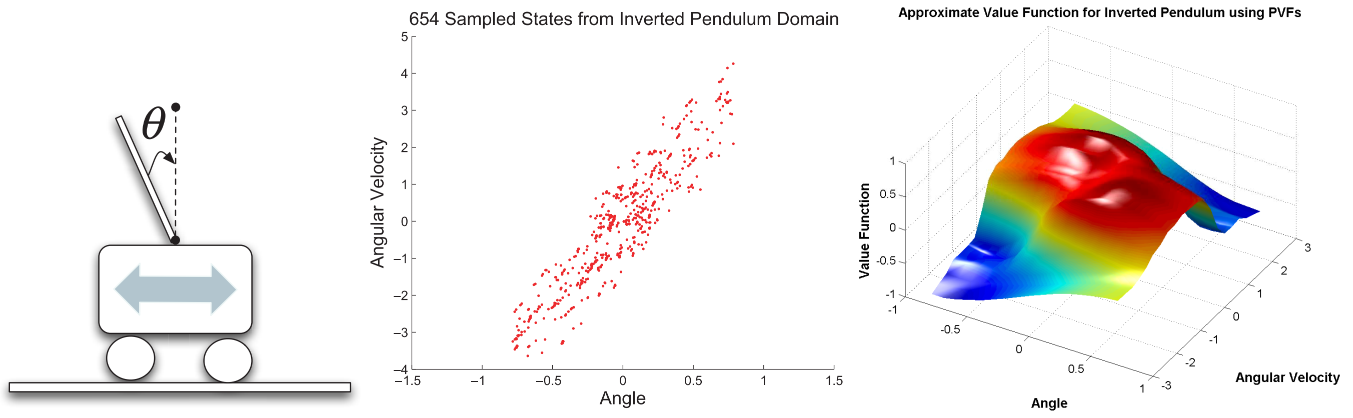


Fig. 1.2 *Left*: The inverted pendulum is a continuous MDP that involves keeping the pole upright by applying equal and opposite forces on the cart. *Middle*: Samples from a series of random walks in a 2D inverted pendulum task. Due to physical constraints, the samples are largely confined to a narrow manifold in the state space. *Right*: By explicitly modeling this manifold by a graph, one of the basis construction methods described in this paper derives customized basis functions called proto-value functions (PVFs) [83] by diagonalizing a random walk operator on a graph connecting nearby samples. An approximation of the optimal value function using proto-value functions is illustrated.

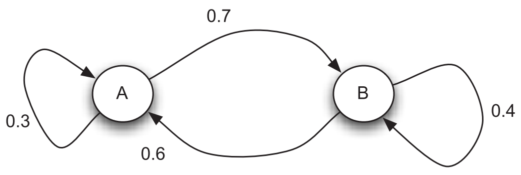
1.2 Laplacian Operators

A unique perspective adopted in this paper is based on exploring links between a family of singular matrices, termed *Laplacians*, and the solution of MDPs.[1] In continuous spaces, the (symmetric) Laplacian operator has been the object of study for almost two centuries: it has been called "the most beautiful object in all of mathematics and physics" [95] as it has played a central role in physics and in many areas of mathematics. On graphs, the discretized (symmetric and non-symmetric) Laplacian has been studied extensively in graph theory, where its spectra reveal structural properties of undirected and directed graphs [26, 27]. Stated in its most general form, the (nonsymmetric) Laplacian matrix is one whose off-diagonal elements are nonpositive and whose row sums are equal to 0 [2, 22, 24]. As we show in this paper, there are strong connections between Laplacian matrices and MDPs. In particular, for any MDP, either in the discounted or average-reward seting, its solution can be shown to involve computing a generalized Laplacian matrix.

Since their row sums are equal to 0, Laplacian matrices are singular, and they do not have a direct inverse (the nullspace is nontrivial since the constant vector of all 1s is an eigenvector associated with the 0 eigenvalue). However, a family of generalized inverses exist for low-rank and singular matrices. The well-known *Moore-Penrose* pseudo-inverse is widely used in least-squares approximation, which will be useful later in this paper. However, a less well-known family of *spectral* inverses — the Drazin inverse (and a special instance of it called the group inverse) is of foundational importance to the study of Markov chains. Indeed, Campbell and Meyer [20] state that:

> For an m-state [Markov] chain whose transition matrix is T, we will be primarily concerned with the matrix $A = I - T$. Virtually everything that one wants to know about a chain can be extracted from A and its Drazin inverse.

[1] In this paper, we use the term "operator" to mean a mapping on a finite or infinite-dimensional space, and the term "matrix" to denote its representation on a specific set of bases.



$$\mathbb{L} = I - P = \begin{bmatrix} 0.7 & -0.7 \\ -0.6 & 0.6 \end{bmatrix}$$

Fig. 1.3 Illustration of a Laplacian operator associated with a MDP. *Left*: A simple two-state Markov chain with transition matrix P. *Right*: Its associated Laplacian matrix. Exact (and approximate) solutions to MDPs can be expressed in terms of a generalized spectral inverse of such singular Laplacian matrices.

We denote transition matrices generally as P, and define the Laplacian associated with a transition matrix as $\mathbb{L} = I - P$ [2, 22, 24] (see Figure 1.3). It has long been known that the Drazin inverse of the singular Laplacian matrix $\mathbb{L}$ reveals a great deal of information about the structure of the Markov chain [92, 121]. In particular, the states in a Markov chain can be partitioned into its various recurrent classes or transient classes based on the Drazin inverse. Also, the sensitivity of the invariant distribution of an ergodic Markov chain to perturbations in the transition matrix can be quantified by the size of the entries in the Drazin inverse of the Laplacian.[2] The solution to average-reward and discounted MDPs can be shown to depend on the Drazin inverse of the Laplacian [21, 110].

As we will show in this paper, Laplacian matrices play a crucial role in the approximate solution of MDPs as well. We will explore a specific set of bases, called Drazin bases, to approximate solutions to MDPs (see Figure 1.4). In continuous as well as discrete MDPs, approximation requires interpolation of noisy samples of the true value function. A growing body of work in machine learning on nonlinear dimensionality reduction [73], manifold learning [8, 30, 116, 131], regression on graphs [99] and representation discovery [80] exploit the remarkable properties of the symmetric Laplacian operator on graphs and manifolds [115]. We will describe how regularization based on symmetric and nonsymmetric graph Laplacians can be shown to provide an automatic method of constructing basis functions for approximately solving

[2] Meyer [92] defines the *condition number* of a Markov chain with transition matrix P by the absolute value of the largest element in the Drazin inverse of $I - P$.

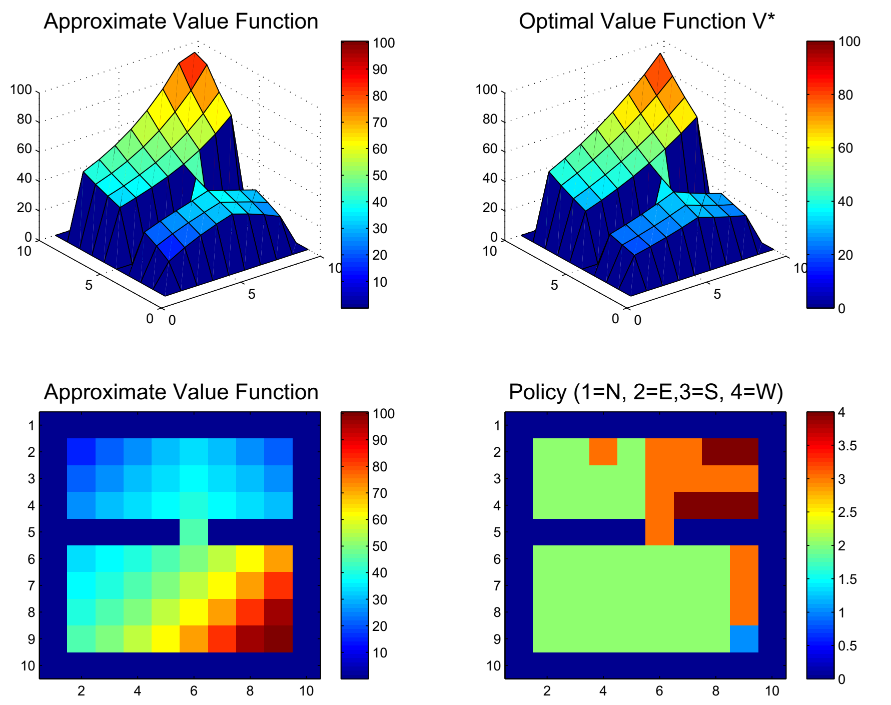


Fig. 1.4 *Top right*: The optimal value function in a two-room MDP with 100 states. *Top left*: Using just 4 Drazin bases, the original MDP is compressed onto a 4D problem, whose solution yields an optimal policy. *Bottom left*: The approximation plotted in 2D showing the state space layout. *Bottom right*: The learned policy using the approximation is optimal.

Table 1.1 Some Laplacian operators on undirected graphs. W is a symmetric weight matrix reflecting pairwise similarities. D is a diagonal matrix whose entries are row sums of W. All these operators are represented by matrices whose row sums are 0 and have non-positive off-diagonal entries.

Operator	Definition	Spectrum
Combinatorial Laplacian	$L = D - W$	$\lambda \in [0, 2\max_v d_v]$
Normalized Laplacian	$\mathcal{L} = I - D^{-1/2} W D^{-1/2}$	$\lambda \in [0, 2]$
Random Walk Laplacian	$L_r = I - D^{-1} W$	$\lambda \in [0, 2]$

MDPs [57, 77, 83, 102]. Table 1.1 describes a few examples of graph Laplacian matrices. The spectral properties of the graph Laplacian reveal a great deal of information about the structure of a graph. In particular, the eigevectors of the symmetric Laplacian yield a low-dimensional representation of a MDP, generating an orthogonal basis that reflects the nonlinear geometry of the state space. We turn to describe the problem of dimensionality reduction in MDPs next.

1.3 Dimensionality Reduction of MDPs

Constructing a low-dimensional representation of a MDP means finding a *basis* Φ with respect to which the original MDP can be represented "compactly" and solved "efficiently."

Definition 1.1. *Basis Construction Problem in MDPs*: Given a Markov decision process M, find an "optimal" basis matrix Φ that provides a "low-dimensional" representation of M, and enables solving M as "accurately" as possible with the "least" computational effort.

Notions like "optimal," "accurately," and "least" will for now be left somewhat vague, but will be defined more precisely later. Note that the solution to the basis construction problem involves managing a set of mutually incompatible trade-offs. For example, a discrete MDP can be solved exactly using the unit vector ("table lookup") representation: this choice of basis optimizes the "accuracy' dimension, and requires no effort in finding the basis, but incurs a sizable computational cost. Exact algorithms like policy iteration [56] have a computational complexity cubic in the size of the state space $|S|$, or exponential in the size of any compact encoding of a state. On the other extreme, it is easy to project a high-dimensional value function $V \in \mathbb{R}^{|S|}$ on a low-order basis space of dimension $\mathbb{R}^k$, where $k \ll |S|$ by trivially choosing a set of random vectors (e.g., each vector is normalized to have length 1 and whose entries are distributed uniformly between 0 and 1). In this case, the cost of solving the MDP may be dramatically reduced, and the effort in finding the basis matrix is again trivial, but the resulting solution may be far from optimal.

It is possible to design an extremely compact basis matrix Φ if the optimal value function V^* is known — namely, use V^* itself! However, knowing V^* presupposes solving the original MDP (presumably on some initial basis, say the unit vectors). This latter solution illustrates the somewhat paradoxical situation that the basis construction problem may require as much or more computational effort than that required to solve the original MDP. An example of an efficient basis is given in Figure 1.4. Here, the optimal policy is found by compressing a 100 state MDP into an effectively 4D space, whose solution gives an optimal policy. However, the cost of finding the Drazin bases is quite significant, since it involves finding the generalized inverse of the Laplacian. In many applications, a decision maker is required to solve many instances of the same problem. An example may be a robot that is tasked to retrieve a set of objects in a given environment, where each object is located in a different room. Thus, the cost of finding such low-dimensional representations may be amortized over the solution of a range of MDPs M_i, say all of which are defined on the same state (action) space, and differ only in the reward function. Finally, in the fully general setting of learning to solve MDPs, the decision maker may only have access to *samples* from the underlying MDP, say by simulation whereby training data are available in the form of trajectories (s_t, a_t, r_t, s_{t+1}). Here, s_t is the state at time t, a_t is the action selected, r_t is the payoff or reward received, and s_{t+1} is the resulting state from performing action a_t. This setting is commonly studied in a variety of areas, such as approximate dynamic programming [12, 109] and reinforcement learning [129]. The methods described later will illustrate these competing trade-offs and how to balance them. It is worthwhile to point out that these similar issues often arise in other domains, e.g., the use of wavelet methods to compress images [86].

1.3.1 Invariant Subspaces of a MDP

This paper describes a range of methods for constructing *adaptive* bases that are customized to the nonlinear *geometry* of a state space, or to a particular policy and reward function. The overarching theme underlying the various methods described in this paper is the notion of constructing representations by decomposing the effect of a linear operator T on the space of functions on a state (or state-action) space, principally by finding its *invariant* subspaces.[3] There are many reasons to find invariant subspaces of an operator T. The solution to a MDP can

[3] If $T : \mathbb{X} \to \mathbb{X}$ is an operator, a subspace $\mathbb{Y} \subseteq \mathbb{X}$ is called invariant if for each $x \in \mathbb{Y}$, $Tx \in \mathbb{Y}$.

be expressed abstractly in terms of finding the fixed point of an operator on the space of value functions. More formally, it can be shown that the value function V^π associated with a policy π is a fixed point of the Bellman operator T^π:

$$T^\pi(V^\pi)(x) = V^\pi(x). \tag{1.1}$$

Thus, the value function V^π forms a 1D invariant subspace of the Bellman operator. We will see, however, that there are compelling reasons for finding other larger invariant spaces. The invariant subspaces associated with a transition matrix P have the attractive property of eliminating prediction errors [11, 104]. Two main principles for constructing invariant subspaces of operators are explored: *diagonalization* and *dilation*.

1.3.2 Diagonalization and Dilation

In this paper, we explore two broad principles for solving the basis construction problem in MDPs by finding invariant subspaces, based on widely used principles in a variety of subfields in mathematics from group theory [123], harmonic analysis [52, 86], linear algebra [127], and statistics [59]. Diagonalization corresponds to finding eigenvectors of an operator: it reduces a possibly full matrix to a diagonal matrix. For example, in linear algebra [127], eigenvectors form invariant subspaces of T since $Tx = \lambda x = x\lambda$. Here, λ is the representation of T on the space spanned by x.

Diagonalization: One generic principle for basis construction involves remapping functions over the state space into a frequency-oriented coordinate system, generically termed *Fourier* analysis [125]. Examples include dimensionality reduction methods in statistics, such as principal components analysis (PCA) [59], low-rank approximations of matrices such as singular value decomposition (SVD) [49], and time-series and image-compression methods, such as the fast Fourier transform [136]. In the case of MDPs, the basis functions can be constructed by diagonalizing the state transition matrix. Often, these matrices are not diagonalizable or are simply not known. In this case, it is possible to construct bases by diagonalizing a "weaker" operator, namely a random walk operator on a graph induced from a MDP's state space, similar to recent work on *manifold learning* [28, 98, 116, 131]. The graph Laplacian [26] is often used, since it is symmetric and its eigenvectors are closely related to that of the natural random walk. We call the bases resulting from diagonalizing the graph Laplacian "proto-value functions" or PVFs [77]. Unlike applications of graph-based machine learning, such as spectral clustering [96] or semi-supervised learning [98], approximating value functions on graphs involves new challenges as samples of the desired function are not readily available. Instead, an iterative procedure is used to sample from a series of functions $\hat{V}_t$, each of which is progressively closer to the desired optimal value function V^*. Furthermore, samples are not available *a priori*, but must be collected by exploration of the MDP's state space. The concept of invariant eigenspaces generalizes to infinite-dimensional Hilbert spaces [37]; one example of which is Fourier analysis in Euclidean spaces. We will explore building finite-dimensional Fourier bases on graphs, and see how to generalize these ideas to eigenfunctions on continuous spaces.

Dilation: Another general method for constructing invariant subspaces uses the principle of *dilation*. For example, a dilation operator on the space of functions on real numbers is $Tf(x) = f(2x)$. Several dilation-based approaches will be compared, including methods based on *Krylov* spaces, a standard approach of solving systems of linear equations [41, 118]. Applied to MDPs, this approach results in the reward function being "dilated" by powers of some operator, such as the transition matrix [106, 103]. A novel basis construction method called *Drazin bases* is described in this paper, which uses the Drazin inverse of the Laplacian $\mathbb{L}^D$. These are a new family of bases building on a theoretical result showing that the discounted value function of a MDP can be written in terms of a series of powers of the Drazin inverse.

Another dilation-based procedure involves a multiscale construction where functions over space or time are progressively remapped into time–frequency or space–frequency *atoms* [34, 86]. This multiscale construction is most characteristic of a family of more recent methods called *wavelets* [34, 86]. We will explore multiscale basis construction on graphs and manifolds using a recent graph-based approach called *diffusion wavelets* [30]. There has been much work on multiscale wavelet methods in image compression and signal processing [87], which can also be viewed using the framework of invariance. We will construct multiscale wavelet bases on discrete graphs, building on the recently developed diffusion wavelet framework [76]. These approaches can be applied to a range of different operators, ranging from a model-based setting using the system dynamics transition matrix to "weaker" operators such as the natural random-walk on the (sampled) underlying state (action) space.

Combined with any of the procedures described above for constructing task-adaptive bases, it is possible to design a variety of architectures for simultaneously learning representation and control. One such framework is generically referred to as *representation policy iteration* [78], comprising of an outer loop where basis functions are constructed and an inner loop where the optimal policy within the linear span of the constructed bases is learned.

1.4 Roadmap to the Paper

The rest of the paper is organized as follows. Section 2 provides an overview of MDPs. Section 3 introduces a general family of Laplacian matrices, and shows how they are intimately connected to solving MDPs. Section 4 surveys various methods for approximately solving MDPs, including least-squares methods, linear programming methods, and reproducing kernel Hilbert space methods. Section 5 formulates the problem of constructing low-dimensional representations of MDPs more precisely, and describes a set of trade-offs that need to be balanced in coming up with effective solutions. Section 6 describes the first of the two main approaches to building basis functions by diagonalization. Section 7 describes methods for constructing representations by dilations of operators. Section 8 shows how these basis construction methods can be combined with methods for approximately solving MDPs to yield model-based techniques that simultaneously learn representation and control. Section 9 describes a generalization of the graph Laplacian operator to continuous sets called manifolds, as well as an interpolation method for approximating continuous eigenfunctions of the manifold Laplacian. Section 10 describes a model-free version of the RPI framework, and evaluates its performance in continuous MDPs. Finally, Section 11 concludes with a brief survey of related work, and a discussion of directions for future work.

2

Sequential Decision Problems

We begin by providing a detailed overview of a class of sequential decision problems called *Markov decision processes* (MDPs). Two common optimality frameworks used in MDPs are described: in average-reward MDPs, the goal of the decision maker is to optimize the expected reward per step; in the discounted framework, the goal is to optimize the expected cumulative discounted sum of rewards, where rewards in the future are decayed geometrically by a discount factor $\gamma < 1$. In Section 3, we will show how Laplacian matrices provide a way to unify both these frameworks by expressing the discounted value function in terms of the average-reward or gain of a policy. We describe methods for exactly solving MDPs, including policy iteration, value iteration, and linear programming. Finally, we discuss simulation-based *reinforcement learning* methods for approximately solving MDPs, when only samples of the transition model and reward function are available. We restrict the discussion in this section to methods that employ table lookup representations (or unit vector bases) to exactly represent transition models, rewards, and policies, leaving the discussion of more general approximation methods to Section 4.

2.1 Markov Decision Processes

The basic modeling paradigm underlying a MDP is that of a decision maker (or "agent") who interacts with an external environment by taking actions. The sequential decision problem is formulated as a set of "states," each of which models a particular situation in the decision problem. For example, a state may be a configuration of pieces in a game of backgammon or chess, the location of a robot, the number of jobs waiting in a queueing system and so on. A crucial assumption made in the MDP model is that the evolution of states is "Markovian," meaning that the distribution of future states is conditionally independent of the past, given perfect knowledge of the current state. Actions cause the environment to stochastically transition to a new state. The desirability of a state is determined by an immediate "reward" or "payoff". The goal of the agent is to choose actions such that it maximizes its long term payoffs. The transition from one state to the next is governed by a probability distribution that reflects the uncertainty in the outcome of decisions. A detailed review of MDPs can be found in Puterman's text [110].

2.1.1 Discounted MDPs

We begin with the discounted optimality framework. The average-reward MDP formulation is described in Section 2.1.2. As we will see later in Section 3, these two frameworks are intimately related through a generalized Laplacian operator.

Definition 2.1. A discrete Markov decision process (MDP) $M = (S, A, P^a_{ss'}, R^a_{ss'})$ is defined by a finite set of discrete states S, a finite set of actions A, a transition model $P^a_{ss'}$ specifying the distribution over future states s' when an action a is performed in state s, and a corresponding reward model $R^a_{ss'}$ specifying a scalar cost or reward. In continuous MDPs, the set of states $\subseteq \mathbb{R}^d$.

Abstractly, a value function is a mapping $S \to \mathbb{R}$ or equivalently (in discrete MDPs) a vector $\in \mathbb{R}^{|S|}$. Given a policy $\pi : S \to A$ mapping states to actions, its corresponding discounted value function V^π specifies the expected long-term discounted sum of rewards received by the agent in any given state s when actions are chosen using the policy.

Definition 2.2. The long-term discounted value associated with a state under a fixed policy is defined as:

$$V^\pi(s) = E_\pi(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s = s_t),$$

where E_π indicates the conditional expectation that the process begins in the initial state s. Actions are chosen at each step using policy π, and r_t is the reward received at time step t.

We can associate with any policy π a value function V^π, which is the fixed point of an operator T^π.

Definition 2.3. Given any deterministic policy π, the operator T^π is defined as

$$T^\pi(V)(s) = R_{s\pi(s)} + \gamma \sum_{s' \in S} P_{ss'}^{\pi(s)} V(s'),$$

where $R_{s\pi(s)} = \sum_{s'} P_{ss'}^{\pi(s)} R_{ss'}^{\pi(s)}$.

It can be shown that $V^\pi(s)$ is a fixed point of T^π, that is

$$T^\pi(V^\pi)(s) = V^\pi(s). \tag{2.1}$$

Definition 2.4. The optimal value function V^* of a MDP is defined as

$$V^*(s) \equiv V^*(s) \geq V^\pi(s) \quad \forall\ \pi, \quad s \in S,$$

where the optimal policy π^* is defined as

$$\pi^*(s) \in \underset{a}{\operatorname{argmax}} \left(R_{sa} + \gamma \sum_{s'} P_{ss'}^a V^*(s') \right).$$

Here, $R_{sa} = \sum_{s' \in s} P_{ss'}^a R_{ss'}^a$.

The optimal policy π^* is not in general unique. However, any optimal policy π^* defines the same unique optimal value function. The optimal value function can be shown to be a fixed point of another operator T^*, defined as follows.

Definition 2.5. The fixed point operator T^* for a MDP is defined as

$$T^*(V)(s) = \max_a \left(R_{sa} + \gamma \sum_{s' \in S} P^a_{ss'} V(s') \right).$$

It can be shown that the optimal value function is a fixed point of the operator T^*, that is

$$T^*(V^*)(s) = V^*(s), \quad \forall\ s \in S. \tag{2.2}$$

Action-value functions are mappings from $S \times A \to \mathbb{R}$, and represent a convenient reformulation of the value function.

Definition 2.6. The *optimal action value* $Q^*(x,a)$ is defined as the long-term value of the nonstationary policy performing a first, and then acting optimally, according to V^*:

$$Q^*(s,a) \equiv E\left(r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,|\, s_t = s, a_t = a \right),$$

where $V^*(s) = \max_a Q^*(s,a)$.

---

where r_{t+1} is the actual reward received at the next time step, and s_{t+1} is the state resulting from executing action a in state s_t. The corresponding Bellman equation for $Q^*(x,a)$ is given as

$$Q^*(s,a) = R_{sa} + \gamma \sum_{s'} P^s_{ss'} \max_{a'} Q^*(s', a'). \tag{2.3}$$

We will describe how action-value functions can be approximated using "simulation-based" reinforcement learning methods in Section 2.3.

2.1.2 Average-Reward Markov Decision Processes

Another popular formulation of MDPs seeks optimal policies that maximize the *expected* reward per step, or the *gain*. We begin by introducing this formulation in the setting of a Markov reward process (MRP) $M_r = (P, R)$, where P is a transition matrix (such as defined by a fixed policy in a MDP), and R is a reward function.

The *chain structure* of a Markov reward process turns out to be crucial in the setting of average-reward MDPs. A transition matrix P can induce several different types of structures, based on whether states continue to be visited indefinitely. The complexity of solving a general average-reward MDP turns out to depend critically on the chain structure.

Definition 2.7. A MDP M is called *ergodic* if every state in M is recurrent under any policy. A recurrent state is one that is visited indefinitely often.

States that are not recurrent are called *transient*. If a state i has a nonzero probability of transitioning to state j and vice-versa, the two states *communicate* with each other. A set of recurrent states forms an irreducible communicating class if no state outside the set can be reached from a state within the set. *Unichain* MDPs have one recurrent set of states, and a possibly nonempty set of transient states. *Multichain* or general MDPs have multiple recurrent sets. In ergodic MDPs, there are no transient states.

Definition 2.8. The long-run transition or *limiting matrix* P^* is defined as

$$P^* = \lim_{n \to \infty} \frac{1}{n} \sum_{k=0}^{n-1} P^k, \tag{2.4}$$

which can be shown to exist when S is finite or countable.

If an MRP is ergodic, then each row of the long-term transition matrix P^* is identical, and equal to the invariant distribution ρ (defined below more formally). If $P_{ss} > 0$, the state s is called *aperiodic*. If all states in a MDP are aperiodic, the MDP is called aperiodic. Notice that a MDP may be ergodic, but not aperiodic (e.g., a two-state MDP where state 1 transitions to 2 and vice versa, but each state does not transition to itself).

Definition 2.9. The invariant distribution ρ of an ergodic MRP $M = (P, R)$ is the left eigenvector associated with the largest eigenvalue $\lambda = 1$, which is also the spectral radius of P.[1]

By convention, ρ is written as a *row* vector $\rho = [\rho_1, \rho_2, \ldots, \rho_n]$, so that

$$\rho P = \rho. \tag{2.5}$$

The limiting matrix P^* for an ergodic and aperiodic MRP can be written as

$$P^* = \lim_{k \to \infty} P^k = \mathbf{1}\rho, \tag{2.6}$$

where $\mathbf{1}$ is a column vector of all 1's. The multiplication $\mathbf{1}\rho$ is the *outer product* of a column vector and row vector, and generates a rank one matrix whose every row is ρ.

Since P is ergodic (or irreducible), by the Perron-Frobenius theorem [10], the largest eigenvalue is indeed the spectral radius $\lambda = 1$, and all other eigenvalues are < 1. Furthermore, the eigenvector associated with λ has all positive real elements.

Definition 2.10. The *gain* or average-reward of a MRP at state s is defined as

$$g(s) = P^* R(s). \tag{2.7}$$

Intuitively, the gain is the average-reward received over the long run starting from state s. The *bias* or average-adjusted sum of rewards

[1] The spectral radius of a matrix A is the real number $\max\{|\lambda| : Ax = \lambda x\}$, where $|\lambda|$ is the modulus of the (possibly complex-valued) eigenvalue λ.

received for an MRP is defined as

$$h(s) = E\left(\sum_{t=1}^{\infty}(R(s_t) - g(s_t))\right), \tag{2.8}$$

where $s_1 = s$.

In ergodic MRPs, since P^* has identical rows equal to ρ, the gain g is not state-dependent, which significantly simplifies the solution of ergodic average-reward MDPs. This property also holds true for *unichain* MRPs, since transient states will not affect the long-term average reward. However, the different transient states can affect the total reward received, and it is desirable to find a policy that maximizes both the gain and the bias. In ergodic (or irreducible) MRPs, the bias can be written as

$$h = \sum_{t=0}^{\infty}(P^t - P^*)R. \tag{2.9}$$

The policy evaluation problem for average-reward ergodic (or unichain) MDPs can be defined as follows.

Definition 2.11. Let M be an ergodic (or unichain) average-reward MDP. The *bias* or average-adjusted value of any policy π can be found by solving the equations:

$$h^\pi = R^\pi - \rho^\pi + P^\pi h^\pi, \tag{2.10}$$

where ρ^π is the gain associated with policy π. $R^\pi(s)$ is the expected one step reward received for taking action $\pi(s)$.

Notice that this is a set of $|S|$ equations in $|S| + 1$ unknowns. One solution is to set the bias of an arbitrary state $h(s) = 0$, as the bias values are really to be viewed as *relative* values. Another approach is to estimate the average-reward independently by averaging the sampled rewards. The action-value formulation of average-reward MDPs can be defined analogously. In Section 3, we will show that solutions to average-reward and discounted MDP are related to each other.

2.1.3 Examples of MDPs

We briefly illustrate the range of applications of MDPs in this section, with some examples from continuous and discrete spaces. A much wider set of examples can be found in standard treatments [12, 109, 129]. Our goal is two-fold: First, these examples illustrate the need for finding low-dimensional representations of MDPs since many of them involve continuous or large state spaces; some of these tasks will be used in the experimental study later in Section 10. Second, these problems show that finding good basis functions for MDPs can be aided by incorporating knowledge of the task, in particular the overall geometry of the state space, or a high-level recursive task decomposition.

2.1.4 Grid World Domains

In discrete MDPs, the state space is over a discrete set of states. In the simplest cases, the states are viewed as atomic entities, such as in the example illustrated in Figure 1.1. Such problems are widely used as simple "toy" testbeds in many papers on MDPs in AI and machine learning. Figure 2.1 illustrates a larger "multiagent" grid world domain

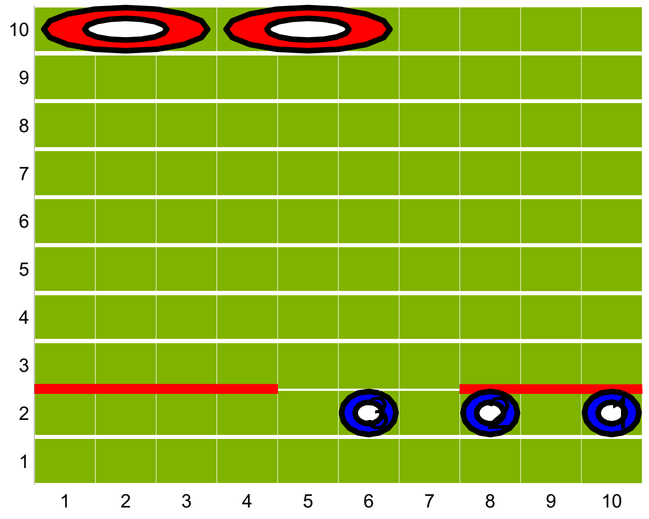


Fig. 2.1 A multiagent "blocker" domain with a state space of $>10^6$ states.

with a state space of over 10^6 states, proposed in [119]. This task is a cooperative multiagent problem where a group of agents try to reach the top row of a grid, but are prevented in doing so by “blocker” agents who move horizontally on the top row. If any agent reaches the top row, the entire team is rewarded by $+1$; otherwise, each agent receives a negative reward of -1 on each step. The agents always start randomly placed on the bottom row of the grid, and the blockers are randomly placed on the top row. The blockers remain restricted to the top row, executing a fixed strategy. The overall state space is the Cartesian product of the location of each agent.

2.1.5 Factored MDPs

In many interesting real-world applications of MDPs, the state space and action space can be modeled as resulting from the combinatorial product of a set of discrete (or continuous) state variables. Figure 2.2 illustrates a simulated “humanoid” robotics domain [114], modeled as a large factored “concurrent” MDP. Concurrent decision-making is a familiar everyday phenomenon of *multi-tasking*: when we drive, we usually engage in other activities as well, such as talking on a cellphone or conversing with other occupants in the vehicle. Concurrency is a fundamental issue in the control of high degree-of-freedom humanoid

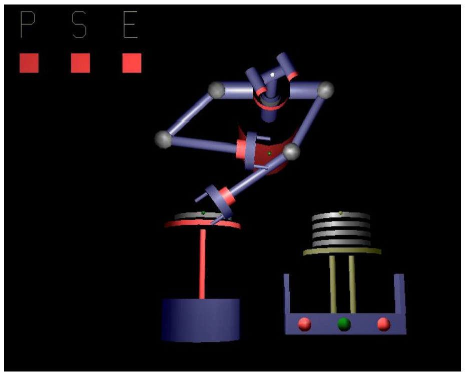


Fig. 2.2 A factored MDP model of a two-armed humanoid robot. The task is to efficiently stack dishes by *coarticulating* multiple concurrent temporally extended actions: for example, as the left arm of the robot is placing a dish on the stack, the right arm can reach over to obtain the next dish.

robots. Solving concurrent MDPs is challenging not only due to the large state space, but also due to the large action space generated by many possible subsets of action variables. A factored concurrent MDP [114] is a model of sequential decision-making, where the interaction between an agent and its environment is modeled by a set of factored state variables $s_1, \ldots, s_n$, and a set of discrete action variables $a_1, \ldots, a_k$. In the humanoid robot concurrent MDP example, there are 7 state variables and 3 action variables, as shown in Figure 2.3. The set of possible states is generated by Cartesian products of the state variables, and the set of possible concurrent actions is generated by products of action variables (not all concurrent actions may be legal in every state).

A common assumption in factored MDPs [55, 124, 51] is to approximate value functions as a linear combination of *fixed* basis functions $F_\Phi = \{\phi_1, \ldots, \phi_k\}$: each basis function represents a localized "feature" $\phi_i : S_i \times A_i \to \mathbb{R}$, where typically S_i is defined over a subset of the state variables, and A_i is defined over a subset of action variables. We will discuss approximation methods for solving (factored) MDPs in more detail in Section 4.

Left arm ($\mathbf{a}_l$)	Right arm ($\mathbf{a}_r$)	Eyes ($\mathbf{a}_e$)
pick	*pick*	*fixate-on-washer*
washer-to-front	*washer-to-front*	*fixate-on-front*
front-to-rack	*front-to-rack*	*fixate-on-rack*
rack-to-front	*rack-to-front*	*no-op*
front-to-washer	*front-to-washer*	
stack	*stack*	
no-op	*no-op*	

$\mathbf{s}_{washer}$	$\mathbf{l}_{pos}$, $\mathbf{r}_{pos}$, $\mathbf{e}_{pos}$	$\mathbf{l}_{stat}$, $\mathbf{r}_{stat}$	$\mathbf{s}_{rack}$
$0, 1, \ldots, n$	*washer*	*has-plate*	*stacked*
	front	*empty*	*not-stacked*
	rack		

Fig. 2.3 Factored state and action variables for the humanoid robot task illustrated in Figure 2.2.

2.1.6 Manufacturing Application

MDPs have long been applied to problems in industrial optimization, ranging from optimizing production on unreliable machines [85] to scheduling elevators [129]. Figure 2.4 shows an example of a scheduling problem involving autonomous guided vehicles (AGVs) in a factory domain (see e.g., [47]). M1 to M3 are workstations in this environment. Parts of type i have to be carried to the drop-off station at workstation i (D_i), and the assembled parts brought back from pick-up stations of workstations (P_i's) to the warehouse. The AGV travel is unidirectional as the arrows show. The AGV receives a reward of 20 when it picks up a part at the warehouse, delivers a part to a drop-off station, picks up an assembled part from a pick-up station, or delivers an assembled part to the warehouse. It also gets a reward of -5 when it attempts to execute Put1–Put3, Pick1–Pick3, Load1–Load3, Unload, and Idle actions illegally. There is a reward of -1 for all other actions. The state of the environment consists of the number of parts in the pick-up and

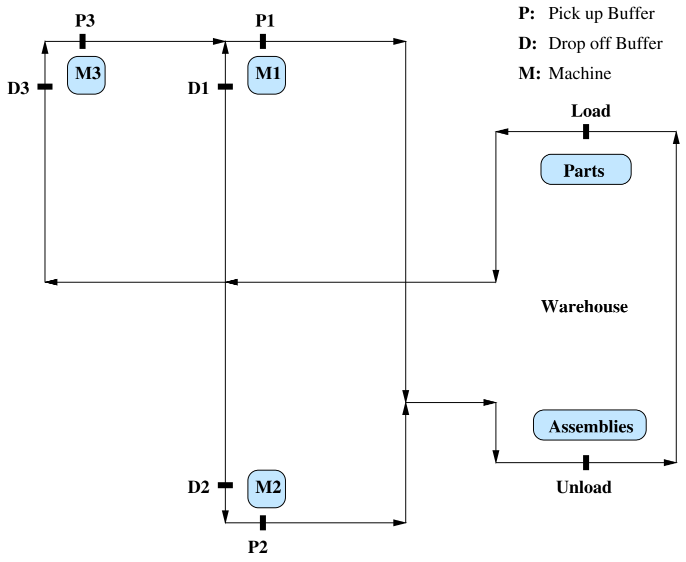


Fig. 2.4 An AGV scheduling task can be modeled as an average-reward MDP. An AGV agent (not shown) carries raw materials and finished parts between machines (M1–M3) and warehouse.

drop-off stations of each machine and whether the warehouse contains parts of each of the three types. In addition, the agent keeps track of its own location and status as a part of its state space. Thus, in the flat case, the state space consists of 33 locations, 6 buffers of size 2, 7 possible states of the AGV (carrying part1–part3, carrying assembly1–assembly3, empty), and 2 values for each part in the warehouse, i.e., $|S| = 33 \times 3^6 \times 7 \times 2^3 = 1{,}347{,}192$ states. Since there are 14 primitive actions (Left, Forward, Right, Put1–Put3, Pick1–Pick3, Load1–Load3, Unload, Idle) in this problem, the total number of parameters that must be learned (the size of the action-value $Q(s,a)$ table) in the flat case is $1{,}347{,}192 \times 14 = 18{,}860{,}688$.

2.1.7 Continuous MDPs

In many applications of MDPs, the states or state variables are continuous, taking values over some prescribed range of real numbers. One example is the Acrobot task [129], a two-link under-actuated robot that is an idealized model of a gymnast swinging on a highbar. The only action available is a torque on the second joint, discretized to one of three values (positive, negative, and none). The reward is -1 for all transitions leading up to the goal state. The detailed equations of motion are given in [129]. The state space for the Acrobot is 4D. Each state is a 4-tuple represented by $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$. θ_1 and θ_2 represent the angle of the first and second links to the vertical, respectively, and are naturally in the range $(0, 2\pi)$. $\dot{\theta}_1$ and $\dot{\theta}_2$ represent the angular velocities of the two links. Notice that angles near 0 are actually very close to angles near 2π due to the rotational symmetry in the state space. Figure 2.5 plots the Acrobot state space projected onto the subspace spanned by the two joint angles θ_1 and θ_2. This subspace is actually a torus.

2.2 Exact Solution Methods

The examples of MDPs shown previously illustrate the computational challenges in solving large MDPs. Before proceeding to tackle large MDPs, it will be essential to introduce the basic solution methods that later approximation methods are based on.

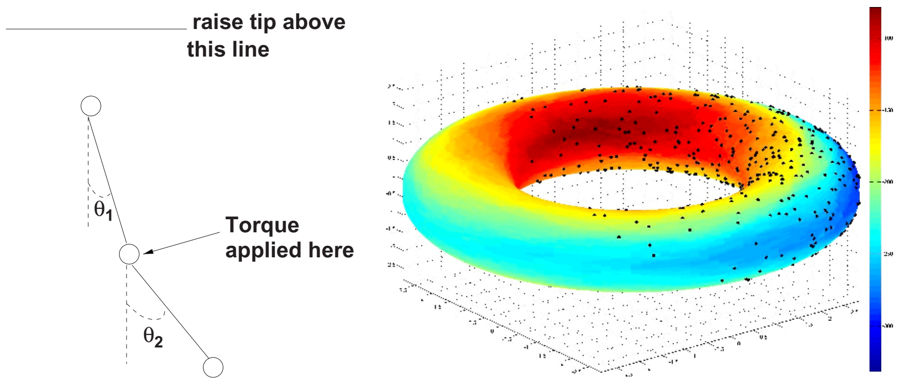


Fig. 2.5 The state space of the Acrobot (shown on the left) exhibits rotational symmetries. The figure on the right plots its projection onto the subspace of $\mathbb{R}^2$ spanned by the two joint angles θ_1 and θ_2, which can be visualized as a torus. The angular velocities $\dot{\theta}_1$ and $\dot{\theta}_2$ were set to 0 for this plot. The points shown on the torus are subsampled states from a random walk. The colors indicate the value function, with red (darker) regions representing states with higher values.

2.2.1 Value Iteration

One way to solve a MDP is to construct the optimal value function V^*, or action value function $Q^*(s,a)$. Since these functions are the fixed points of operators, a natural strategy is to find the fixed point by successive approximation. The value iteration algorithm computes the next approximation V^{t+1} by iteratively "backing up" the current approximation:

$$V^{t+1}(s) = T^*(V^t)(s) = \max_a \left(R_{sa} + \gamma \sum_a P^a_{ss'} V^t(s') \right). \qquad (2.11)$$

The value iteration algorithm can be terminated when the difference between successive approximations is less than some tolerance parameter, that is $\|V^{t+1} - V^t\| \leq \epsilon$, where $\| \cdot \|$ denotes the "norm" or "length" of a function. It can be shown that the operator T^* is a *contraction* in a Hilbert space, and hence the algorithm will asymptotically converge [138].[2] A similar procedure can be used to compute the optimal action value function $Q^*(s,a)$. A drawback of value iteration

[2] Intuitively, this property implies that at each iteration, the "distance" between V^t and V^* shrinks, and hence asymptotically, the algorithm must converge.

is that its convergence can be slow for γ near 1. Also, at each step, it is necessary to "backup" values over the entire state space. This problem has been addressed by a number of approaches; algorithms such as *real-time dynamic programming* (RTDP) and "simulation-based" reinforcement learning methods back up values over only a set of sample transitions [129].

2.2.2 Policy Iteration

The *policy iteration* algorithm was introduced by Howard [56]. In this procedure, at each iteration, the decision maker "evaluates" a specific policy π, finding its associated value function V^π. Then, the associated "greedy" policy π' associated with V^π is computed, which deviates from π for one step by finding the action that maximizes the one-step reward and then follows π subsequently. Howard proved that if π is not the optimal policy, the greedy policy associated with V^π must improve on π. This procedure will become the foundation

Algorithm 1 The Policy Iteration Algorithm for Discounted MDPs

```
Set $\pi$ to some random initial policy.
1: repeat
2:   Solve the "Bellman" linear system of equations to determine $V^\pi$
         $(I - \gamma P^\pi)V^\pi = R^\pi$
3:   Find the "greedy" policy $\pi'$ associated with $V^\pi$:
         $\pi'(s) \in \operatorname*{argmax}_a \left( \sum_{s'} P^a_{ss'} \left( R^a_{ss'} + \gamma V(s') \right) \right)$
4:   if $\pi' \neq \pi$ then
5:     Set $\pi \leftarrow \pi'$ and return to Step 2.
6:   else
7:     Set done $\leftarrow$ true.
8:   end if
9: until done.
   Return $\pi$ as the optimal discounted policy for MDP $M$.
```

for the *Representation Policy Iteration* (RPI) framework presented in Sections 8 and 10, where both basis functions as well as optimal policies will be simultaneously learned [78]. As described here, the policy iteration algorithm assumes that value functions (and other functions such as rewards, transition models, and policies) can be stored exactly, using a "table lookup" representation. In Section 4, we will describe a variant of this procedure, called least-squares policy iteration (LSPI) [69], which is based on least-squares approximation of the action value function Q^π associated with policy π.

2.2.3 Linear Programming

A third approach to exactly solving MDPs is based on *linear programming* [110]. The variables for the linear program are the values $V(i)$. The exact formulation is given below:

Definition 2.12. The linear program required to solve a MDP M is given as

$$\begin{aligned} \text{Variables}: &\ V(1),\ldots,V(n), \\ \text{Minimize}: &\ \sum_s \alpha_s V(s), \\ \text{Subject to}: &\ V(s) \geq \sum_{s'} P^a_{ss'} \left(R^a_{ss'} + \gamma V(s') \right), \quad \forall\ s \in S, \quad a \in A, \end{aligned}$$

where α is a state relevance weight vector whose weights are all positive.

There is one constraint for each state s and action a, thus, leading to an intractable set of constraints in problems with an exponential state space. Later, we will describe a variant of this linear programming (LP) formulation in Section 4, which uses a set of basis functions to compute an approximation of the exact value function that lies in the space spanned by the bases.

2.3 Simulation-Based Methods

We now turn to briefly describe a class of approximation methods for solving MDPs, which retain the restriction of representing functions exactly, but require only sample transitions (s_t, a_t, r_t, s_t') instead of true knowledge of the MDP. Such methods can be generically referred to as *simulation-based* methods, and are the topic of much study in various areas, including *approximate dynamic programming* [12, 109] and *reinforcement learning* [129]. Two classes of methods will be described: Monte–Carlo methods, which approximate the exact return V^π by summing the actual returns, and *temporal-difference learning* methods, which can be viewed as a biased version of Monte–Carlo methods.

2.3.1 Monte–Carlo Methods

Monte–Carlo methods have long been studied in a variety of fields, and there is a well-developed statistical theory underlying them [113]. Monte–Carlo methods for solving MDPs are based on the simple idea that the value of a particular state $V^\pi(s)$ associated with a particular policy π can be empirically determined by "simulating" the policy π on a given MDP M, and averaging the sum of rewards received. Monte–Carlo methods are simple to implement. As a statistical procedure, they have the attractive property of being *unbiased* estimators of the true value. Unfortunately, their variance can be high. A more detailed discussion of Monte–Carlo methods for MDPs is given in [12, 129].

Algorithm 2 Monte–Carlo method for evaluating policy π in state s.

```
for $i = 1$ to $N$ do
    Set the step counter $t = 0$, initial state $s_t = s$, and $\hat{V}_i = 0$.
    Set the action $a = \pi(s_t)$, and "execute" action $a$.
    Let the sample reward received be $r_t$. Set $t \leftarrow t + 1$.
    Set $\hat{V}_i = \hat{V}_i + r_t$.
    if terminated then
        Set $\hat{V}_i = \frac{1}{t}\hat{V}_i$.
    end if
end for
Return $\hat{V}^\pi(s) = \frac{1}{N}\sum_{i=1}^{N} \hat{V}_i$.
```

2.3.2 Temporal-Difference Learning

Temporal-difference (TD) learning methods [130] exploit the property that the value of a state $V^\pi(s)$ can be estimated as the sum of the immediate reward received, r_t, and a *biased* estimate of the value at the next state. The simplest TD(0) algorithm can be written as

$$\hat{V}^\pi_{t+1}(s) \leftarrow (1-\alpha_t)\hat{V}^\pi_t(s) + \alpha_t \left(r_t + \hat{V}^\pi_t(s) \right). \tag{2.12}$$

Here, $\alpha_t \in (0,1)$ is a time-varying "learning rate" that averages the estimate over multiple samples. As in the Monte–Carlo procedure, the TD(0) algorithm also only requires a simulation of a MDP, and asynchronously evaluates the value of states in a MDP over sampled trajectories. A more sophisticated least-squares variant of TD algorithm uses a set of nonunit vector bases to approximate value functions [16, 18]. Readers familiar with the theory of stochastic approximation [65] may recognize the TD update rule given above as similar to the *Robbins-Munro* method of finding the roots of a function. Indeed, a rigorous convergence analysis of TD learning has been made drawing on the theory of stochastic approximation [12, 138]. A TD-based method for learning action values called Q-learning was introduced by Waktins [140]. Q-learning estimates the optimal action value function using the following learning rule:

$$Q_t(s,a) \leftarrow (1-\alpha_t)Q_t(s,a) + \alpha_t \big(r_t + \gamma \max_{a'} Q_t(s',a') \big). \tag{2.13}$$

Q-learning is sometimes referred to as an "off-policy" learning algorithm since it estimates the optimal action value function $Q^*(s,a)$, while simulating the MDP using any policy, such as a random walk. In practice, a greedy policy is used that picks actions at each state with the highest $Q(x,a)$, occasionally choosing random actions to ensure sufficient exploration.

3

Laplacian Operators and MDPs

In this section, we introduce a broad class of Laplacian operators, which play an important role in later sections. In particular, we explore two specific types of Laplacian operators. The first type is intimately linked to the study of Markov decision processes (MDPs) introduced in Section 2. In particular, we show that exact solutions of average-reward and discounted MDP models can be expressed in terms of a generalized inverse of a Laplacian operator. We introduce a generalized spectral inverse called the Drazin inverse. We show that discounted value functions can be written as a Laurent series expansion that involves powers of the *Drazin* inverse of the Laplacian. This expansion will provide the theoretical foundation for a new approach to approximating MDPs using the Drazin basis, which we will explore in later sections. We also introduce a more restricted class of positive-definite Laplacian matrices that have been the topic of considerable interest in machine learning to approximate functions on graphs and continuous sets embedded in Euclidean spaces called *manifolds*. These so-called graph Laplacian operators will be extended later to manifolds, and lead to an effective method of approximating continuous MDPs.

3.1 Laplacian Operators

A unique feature of this paper is the explicit linking of a class of singular operators called Laplacians with exact and approximate solutions of MDPs. A common property shared by all the Laplacian operators in this paper is that their matrix representations have row sums that are 0, and their off-diagonal entries are nonpositive.

Definition 3.1. The *generalized Laplacian* operator [2, 22, 24] is defined as a (possibly nonsymmetric) matrix $\mathbb{L} \in \mathbb{R}^{n\times n}$, where

$$\mathbb{L}(i,j) \leq 0, \quad i \neq j,$$
$$\sum_{j=1}^{n} \mathbb{L}(i,j) = 0, \quad i = 1,\ldots,n.$$

Note importantly that no assumption is made here that $\mathbb{L}$ is symmetric.[1] It is possible to associate a generalized Laplacian matrix $\mathbb{L}_\Gamma$ with any *directed graph* $\Gamma = (V,E,W)$, where the edge weights $W(i,j)$ associated with the directed arc $(i,j) \in E$ are strictly positive. More precisely, the Laplacian matrix $\mathbb{L}_\Gamma$ is defined as follows:

$$\mathbb{L}_\Gamma(i,j) = \begin{cases} -W(i,j) & \text{if } i \neq j \quad \text{and} \quad (i,j) \in E, \\ 0 & \text{if } i \neq j \quad \text{and} \quad (i,j) \notin E, \\ -\sum_{k\neq i} \mathbb{L}_\Gamma(i,k) & \text{if } i = j. \end{cases}$$

There are interesting connections between generalized Laplacian matrices and stochastic matrices. In particular, the following theorem is a straightforward consequence of the above definition.

Theorem 3.1. For any stochastic matrix P and $\alpha > 0$, the matrix $\alpha(I - P)$ is a generalized Laplacian.

There are fundamental connection between Laplacian matrices and MDPs, as we discuss next. In what follows, for brevity, we will refer to "generalized Laplacians" as simply "Laplacian matrices."

[1] Laplacian matrices are sometimes also referred to as *discrete Schrodinger operators*, particularly when the matrix is symmetric [134]. Laplacian matrices can also be viewed as singular M-matrices [10].

3.2 Laplacian Matrices in MDPs

In this section, we describe how the solution of average-reward and discounted MDPs can be defined in terms of a Laplacian matrix. We explain how the Laplacian plays a key role in the exact solution of MDPs. Special instances of this general form will be studied in later sections in the approximate solution of MDPs.[2]

3.2.1 Discounted MDPs and the Laplacian

As we have seen earlier, solving a MDP requires computing the value functions associated with policies. Let P^π represent an $|S| \times |S|$ transition matrix of a (deterministic) policy $\pi : S \to A$ mapping each state $s \in S$ to a desired action $a = \pi(s)$. We can associate a Laplacian $\mathbb{L}^\pi$ with any policy π.

Definition 3.2. Given any policy π in a MDP M with associated transition matrix P^π, the Laplacian $\mathbb{L}^\pi$ is defined as

$$\mathbb{L}^\pi = I - P^\pi. \tag{3.1}$$

Let R^π be a (column) vector of size $|S|$ of rewards. The value function associated with policy π can be computed using the Neumann series:

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi = \left(I + \gamma P^\pi + \gamma^2 (P^\pi)^2 + \cdots\right) R^\pi. \tag{3.2}$$

To show the connection of the Laplacian $\mathbb{L}^\pi$ to V^π, we reformulate Equation (3.2) in terms of an *interest rate* ρ [110].

Definition 3.3. The *interest rate* $\rho \equiv (1 - \gamma)\gamma^{-1}$ or equivalently, $\gamma = \frac{1}{1+\rho}$. The interpretation of ρ as an interest rate follows from the property that, if a reward of 1 is "invested" at the first time step, then $1 + \rho$ is the amount received at the next time step. The interest rate

[2] It is somewhat striking that the literature on Markov chains and MDPs investigating the properties of Laplacian matrices [21, 92, 110, 121] nonetheless does not refer to these matrices as Laplacians!

formulation of the discounted value function associated with a policy π can be written as

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi = (1+\rho)(\rho I + \mathbb{L}^\pi)^{-1} R^\pi, \tag{3.3}$$

where $\mathbb{L}^\pi = I - P^\pi$ is the Laplacian matrix associated with policy π.

For $\rho > 0$, the matrix $(\rho I + \mathbb{L}^\pi)^{-1}$ is called the resolvent of $-\mathbb{L}^\pi$ at ρ. For $\gamma < 1$, the spectral radius of $\gamma P^\pi < 1$, and hence the inverse $(1 - \gamma P^\pi)^{-1}$ exists. Consequently, the resolvent $(\rho I + \mathbb{L}^\pi)^{-1}$ also exists.

3.2.2 Average-Reward MDPs and the Laplacian

There are close connections between the solution of average-reward MDPs [110] and the Laplacian as well; the gain and bias of a Markov reward process can be defined in terms of the Laplacian.

Theorem 3.2. Given a MDP $M = (S, A, P, R)$, let the Markov reward process defined by any policy π be $M^\pi = (P^\pi, R^\pi)$. Let the gain of the MRP be defined as g^π and h^π. The gain g^π is in the nullspace of the Laplacian $\mathbb{L}^\pi$. More precisely, we have

$$\mathbb{L}^\pi g^\pi = (I - P^\pi) g^\pi = 0, \tag{3.4}$$

$$g^\pi + \mathbb{L}^\pi h^\pi = R^\pi. \tag{3.5}$$

Equation (3.4) can be readily derived from the properties of the long-term limiting matrix $(P^\pi)^*$:

$$\begin{aligned} \mathbb{L}^\pi g^\pi &= (I - P^\pi)(P^\pi)^* R^\pi \\ &= ((P^\pi)^* - P^\pi (P^\pi)^*) R^\pi = ((P^\pi)^* - (P^\pi)^*) R^\pi = 0. \end{aligned}$$

The derivation of Equation (3.5) is more subtle; we will need to introduce a generalized inverse of the Laplacian. We will also show how the study of average-reward and discounted MDPs can be unified using the generalized inverse of the Laplacian.

3.3 Generalized Inverses of the Laplacian

Not all matrices are invertible. In particular, if a matrix $A \in \mathbb{C}^n \times \mathbb{C}^n$ has an eigenvalue $\lambda = 0$, or alternatively, if its column space is not of full rank, then A has no real inverse. However, in a large class of applications in statistics and machine learning, such low-rank matrices are commonplace. Clearly, the Laplacian associated with the transition matrix P, namely $\mathbb{L} = I - P$, is nonvertible as it is not of full rank: its row sums add to 0, and hence the constant eigenvector $\mathbf{1}$ is associated with the 0 eigenvalue. In such cases, one can define *generalized inverses* that satisfy many of the properties of a real inverse [20].

Definition 3.4. A generalized inverse X of a matrix $A \in \mathbb{C}^n \times \mathbb{C}^n$ is a matrix that satisfies one or more of the following properties:

(1) $AXA = A$.
(2) $XAX = X$.
(3) $(AX)^* = AX$.
(4) $(XA)^* = XA$.
(5) $AX = XA$.
(6) $A^{k+1}X = A^k$.

where $()^*$ denotes the conjugate transpose.

A generalized inverse matrix X of A that satisfies a subset $C \subset \{1, 2, 3, 4, 5, 6\}$ of these properties is labeled a A^C inverse.

All of these properties are satisfied, of course, by the true inverse A^{-1} of a nonsingular matrix. However, if A is singular, there is no generalized inverse X of A that satisfies all these properties. It turns out, however, that there are unique generalized inverses that satisfy some interesting subsets. For example, the well-known *Moore-Penrose* pseudo-inverse $A^\dagger$ of A is a $A^{\{1,2,3,4\}}$ inverse since it satisfies the first four properties. We will later use this generalized inverse for least-squares approximation of value functions.

Another generalized inverse exists called the *Drazin inverse* (or its special case, the *group inverse*) plays a crucial role in the study of Markov chains and MDPs [110, 121]. We will introduce this inverse in

Section 3.3.3. As we will show below, the Drazin or group inverse of the Laplacian also intimately links the study of average-reward MDPs with discounted MDPs. We will later investigate the Drazin basis as a way of approximating MDPs. We will also introduce another important matrix, the fundamental matrix of a Markov chain, which is related to the Drazin inverse.

3.3.1 Fundamental Matrix

For simplicity, we begin our discussion with the simpler case of ergodic chains — recall that these have a single recurrent class. The fundamental matrix is an important matrix associated with the study of Markov chains [121].

Theorem 3.3. The eigenvalues of the matrix $P - P^*$ of an ergodic Markov chain lie within the unit circle. Consequently, the *fundamental matrix* [110, 121] associated with P

$$\mathbb{L} + P^* = I - P + P^* \tag{3.6}$$

is invertible.

Proof. Since P is ergodic,[3] from the Perron Frobenius theorem [10], it follows that $\lambda = 1$ is its largest eigenvalue and also its spectral radius. Furthermore, all other eigenvalues λ_i have their modulus $|\lambda_i| < 1, 1 \leq i \leq n - 1$ (assuming P is an $n \times n$ matrix).[4] Each of these eigenvalues λ_i must also be an eigenvalue of $P - P^*$, because if $Px_i = \lambda_i x_i$, where $\lambda_i \neq 0$, then it follows that

$$P^* x_i = \frac{1}{\lambda_i} P^* P x_i = \frac{1}{\lambda_i} P^* x_i.$$

But, this can only hold if $P^* x_i = 0$ since $\lambda_i \neq 0$ and $\lambda_i \neq 1$. Hence, it follows that

$$(P - P^*)x_i = Px_i = \lambda_i x_i.$$

[3] A general proof for nonergodic chains can be given using the Drazin inverse, which we introduce in Section 3.3.3. We give a simpler proof here following [21].

[4] In general, eigenvalues of stochastic matrices are complex-valued. The modulus of a complex number $z = a + ib$ is defined as $|z| = \sqrt{a^2 + b^2}$.

Thus, x_i is an eigenvector of $P - P^*$ with eigenvalue λ_i if it is also an eigenvector of P. If, on the other hand, $\lambda_i = 0$, then $P^* x_i = P^* P x_i = \lambda_i P^* x_i = 0$, and hence $\lambda_i = 0$ is an eigenvalue of $P - P^*$ as well. Finally, the eigenvalues of the fundamental matrix $\mathbb{L} - P^* = I - P + P^*$ are $1, 1 - \lambda_i, \ldots, 1 - \lambda_{n-1}$, none of which are 0. $\square$

3.3.2 Group Inverse of the Laplacian

We now introduce a generalized inverse of the Laplacian $\mathbb{L} = I - P$ called the *group inverse* [20, 21]. The group inverse is a special case of a more general inverse called the *Drazin inverse*, which is introduced below.

Definition 3.5. The *group inverse* $A^{\#}$ of a square matrix $A \in \mathbb{C}^n \times \mathbb{C}^n$ is a $A^{1,2,5}$ inverse.

We now introduce the group inverse of the Laplacian, and show why it is of fundamental importance in MDPs.

Definition 3.6. The *group inverse* of the Laplacian $\mathbb{L} = I - P$ of a Markov chain with transition matrix P is defined as

$$\mathbb{L}^{\#} = (I - P + P^*)^{-1} - P^*. \tag{3.7}$$

It can be shown that the properties 1, 2, and 5 in Definition 3.4 are satisfied. As an example, let us show that $\mathbb{L}^{\#}\mathbb{L} = \mathbb{L}\mathbb{L}^{\#}$.

$$\begin{aligned}
\mathbb{L}^{\#}\mathbb{L} &= \left[(I - P + P^*)^{-1} - P^*\right](I - P) \\
&= (I - P + P^*)^{-1}(I - P) - P^*(I - P) \\
&= (I - P + P^*)^{-1}(I - P) \\
&= (I - P + P^*)^{-1}(I - P) + (I - P + P^*)^{-1}P^* - P^* \\
&= (I - P + P^*)^{-1}(I - P + P^*) - P^* \\
&= I - P^* \\
&= \mathbb{L}\mathbb{L}^{\#}.
\end{aligned}$$

The meaning of the term "group inverse" arises from the property that it is actually the inverse element in a group $\Gamma_{\mathbb{L}}$ of all Laplacian matrices $\mathbb{L}$, defined for a specific invariant distribution ρ, where $\rho P = \rho$, and $P^* = \mathbf{1}\rho$.

$$\Gamma_{\mathbb{L}} = \{\mathbb{L} : \rho\mathbb{L} = 0, \mathbb{L}\mathbf{1} = 0, (\mathbb{L} + P^*)^{-1} \text{ exists}\}. \tag{3.8}$$

We now link the group inverse of the Laplacian to the average-reward Bellman equation (Equation (2.10)). We can expand the fundamental matrix in a Taylor series expansion as follows

$$(I - P + P^*)^{-1} = \sum_{t=0}^{\infty} \left(P^t - P^*\right). \tag{3.9}$$

Using this Taylor series expansion of the fundamental matrix, we can derive the average-reward Bellman equation as

$$\begin{aligned}
(I - P)(I - P + P^*)^{-1} &= I - P^* \\
P^* + (I - P)(I - P + P^*)^{-1} &= I \\
P^*R + (I - P)(I - P + P^*)^{-1}R &= R \\
g + (I - P)\sum_{t=0}^{\infty}\left(P^t - P^*\right)R &= R \\
g + (I - P)h &= R \\
g + \mathbb{L}h &= R, \\
&\qquad \text{after multiplying both sides by R}
\end{aligned}$$

which is exactly the form of the average-reward equation given in Equation (3.5).

A simple interpretation of the group inverse of the Laplacian can be given based on Equation (3.9):

$$\begin{aligned}
(I - P)^{\#} &= (I - P + P^*)^{-1} - P^* \\
&= \sum_{t=0}^{\infty}(P - P^*)^t - P^*
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{t=0}^{\infty}(P^t - P^*) - P^* \\
&= \lim_{n\to\infty}\sum_{k=0}^{n-1}(P^k - P^*) - P^* \\
&= \lim_{n\to\infty}\sum_{k=0}^{n-1}(P^k - nP^*).
\end{aligned}
$$

In other words, the element (i, j) in the group inverse matrix is the difference between the expected number of visits to state j starting in state i following the transition matrix P versus the expected number of visits to j following the long-term limiting matrix P^*. Figure 3.1 shows this difference on a simple two state Markov chain.

3.3.3 Drazin Inverse of the Laplacian

Finally, we introduce the Drazin inverse of the Laplacian. In this section, we consider arbitrary Markov chains, regardless of their chain structure. First, we define Drazin inverses generally, using the generalized inverse axioms, and then give an explicit matrix definition for stochastic matrices.

Definition 3.7. The *index* of a square matrix $A \in \mathbb{C}^n \times \mathbb{C}^n$ is the smallest nonnegative integer k such that

$$\mathcal{R}(A^k) = \mathcal{R}(A^{k+1}). \tag{3.10}$$

Here, $\mathcal{R}(A)$ is the range (or column space) of matrix A.

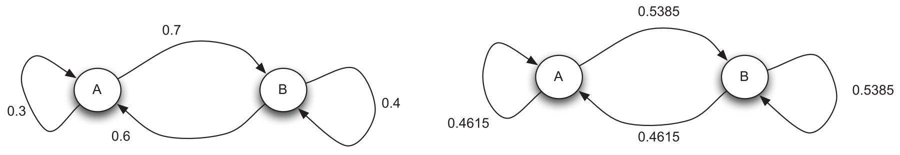


Fig. 3.1 This figure illustrates the concept of group inverse of the Laplacian, on a simple two state Markov chain. *Left*: The original Markov chain with transition matrix P. *Right*: A chain with transition matrix P^*, the limiting matrix. The group inverse measures the difference in the expected number of visits to a state following P versus P^*.

For example, a nonsingular (square) matrix A has index 0, because $\mathcal{R}(A^0) = \mathcal{R}(I) = \mathcal{R}(A) = \mathbb{C}^n$. The Laplacian $\mathbb{L} = I - P$ of a Markov chain has index 1.

Definition 3.8. The *Drazin* inverse of a matrix $A \in \mathbb{C}^n \times \mathbb{C}^n$ is a $A^{\{2,5,6\}}$ inverse, where property 6 holds when k is the index of A.

We now give an equivalent definition of the Drazin inverse in terms of the decomposition of a matrix.

Definition 3.9. If a general square matrix A is decomposed as follows

$$A = W \begin{pmatrix} C & 0 \\ 0 & N \end{pmatrix} W^{-1}, \tag{3.11}$$

where W and C are nonsingular matrices, and N is nilpotent,[5] then its Drazin inverse of A is given by:

$$A^D = W \begin{pmatrix} C^{-1} & 0 \\ 0 & 0 \end{pmatrix} W^{-1} \tag{3.12}$$

Let us specialize this definition for general (nonergodic) Markov chains, and investigate its application to MDPs [110].

Definition 3.10. Let a transition matrix P of a Markov chain be defined on a finite state space S, and suppose P induces m recurrent classes on S. Then, P can be decomposed into the following form:

$$P = W \begin{pmatrix} I & 0 \\ 0 & Q \end{pmatrix} W^{-1}, \tag{3.13}$$

where W is nonsingular, I is an $m \times m$ identity matrix, and Q is an $|S| - m \times |S| - m$ times matrix. The inverse $(I - Q)$ exists since 1 is not an eigenvalue of Q. Also $\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n-1} Q^k = 0$, since Q is nilpotent.

We can now define the Drazin inverse $\mathbb{L}^D$ of the Laplacian $\mathbb{L} = I - P$ in its general form.

[5] N is a nilpotent matrix if for some nonnegative integer k, $N^k = 0$.

Definition 3.11. Given an arbitrary transition matrix P on a finite state space S, the Drazin inverse $\mathbb{L}^D$ of $\mathbb{L}$ is given by

$$\mathbb{L}^D = (I - P)^D = W \begin{pmatrix} 0 & 0 \\ 0 & (I - Q)^{-1} \end{pmatrix} W^{-1}. \tag{3.14}$$

Also, the long-term limiting matrix P^* of a general Markov chain P can be written as

$$P^* = W \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} W^{-1}. \tag{3.15}$$

Note that we can now derive the identity given in Definition 3.6 for arbitrary Markov chains given the matrix form of the Drazin inverse, since

$$\begin{aligned} (I - P)^D &= W \begin{pmatrix} 0 & 0 \\ 0 & (I - Q)^{-1} \end{pmatrix} W^{-1} \\ &= W \begin{pmatrix} I & 0 \\ 0 & (I - Q)^{-1} \end{pmatrix} W^{-1} - W \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} W^{-1} \\ &= (I - P + P^*)^{-1} - P^*. \end{aligned}$$

3.3.4 Computation of Drazin Inverse and Limiting Matrix

Let us briefly discuss the issue of computing the Drazin inverse $\mathbb{L}^D$ and the long-term limiting matrix P^*. A number of algorithms have been developed [20]. One approach is to form the direct decomposition of a matrix A into its nonsingular component and nilpotent component.

There are many other methods. For example, any 1-inverse of $\mathbb{L}$ can be used to find its Drazin inverse, such as the Moore-Penrose pseudoinverse $\mathbb{L}^\dagger$. More sophisticated methods are available based on the orthogonal deflation procedure used in singular value decomposition [20]. In Section 7.5, we will describe a multiscale iterative method for fast computation of the Drazin inverse.

3.3.5 Unifying Average-Reward and Discounted MDPs

In Equation (3.3), we formulated the discounted value function in terms of the resolvent of the Laplacian. We now introduce an important

Algorithm 3 A general algorithm for Drazin inverse

```
1: Input: A matrix $A$ and its index $k$ (if $A = \mathbb{L} = I - P$, then $k = 1$).
2: Form the row-reduced echelon form of $A$ (e.g., use MATLAB
   command rref).
       $[A_r \;\; C_A] = \texttt{rref}(A^k),$          (3.16)
   where $A_r$ is the row-reduced echelon form, and $C_A$ is the set of
   column indices of $A$ such that $A(:, C_A) = [v_1, v_2, \ldots, v_r]$ forms a basis
   for $\mathcal{R}(A^k)$.
3: Form the matrix $I - A_r$. Its nonzero columns, denoted as
   $[v_{r+1}, \ldots, v_n]$, form a basis for $\mathcal{N}(A^k)$.
4: Construct a nonsingular matrix $W = [v_1, \ldots, v_n]$, and form the
   product
       $W^{-1}AW = \begin{pmatrix} C & 0 \\ 0 & N \end{pmatrix},$          (3.17)
   where $C$ is the nonsingular portion and $N$ is the nil-potent portion.
4: The Drazin inverse of $A$ is given as
       $A^D = W \begin{pmatrix} C^{-1} & 0 \\ 0 & 0 \end{pmatrix} W^{-1}.$          (3.18)
5: If $A$ is the Laplacian $\mathbb{L} = I - P$, the long-term limiting matrix $P^*$
   is given as
       $P^* = W \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} W^{-1}.$          (3.19)
```

connection between average-reward and discounted MDPs using the *Laurent* series expansion of the resolvent of the Laplacian. This series will be in terms of powers of the Drazin inverse of the Laplacian. We will later use this expansion as an interesting type of basis for approximating MDPs.

Theorem 3.4. If $\mathbb{L} = I - P$, and $0 < \rho < \sigma(I - P)$ (where σ denotes the spectral radius), the resolvent $(\rho I + \mathbb{L})^{-1}$ can be expressed as the following Laurent series:

$$(\rho I + \mathbb{L})^{-1} = \rho^{-1} P^* + \sum_{n=0}^{\infty} (-\rho)^n (\mathbb{L}^D)^{n+1}. \tag{3.20}$$

Proof. We base our proof on that given in [110]. Given the decomposition of P in Equation (3.13), we can express the resolvent $(\rho I + \mathbb{L})^{-1}$ as

$$\begin{aligned}(\rho I + \mathbb{L})^{-1} &= W \begin{pmatrix} \rho^{-1} I & 0 \\ 0 & (\rho I + (I - Q))^{-1} \end{pmatrix} W^{-1} \\ &= \rho^{-1} W \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} W^{-1} + W \begin{pmatrix} 0 & 0 \\ 0 & (\rho I + (I - Q))^{-1} \end{pmatrix} W^{-1}.\end{aligned}$$

Notice that the first term is $\rho^{-1} P^*$. Using the identity

$$(\rho I + (I - Q))^{-1} = (I + \rho (I - Q)^{-1})^{-1} (I - Q)^{-1},$$

we can expand the first expression in the second term above as

$$(I + \rho (I - Q)^{-1})^{-1} = \sum_{n=0}^{\infty} (-\rho)^n ((I - Q)^{-1})^n,$$

which is valid as long as $\rho < \sigma(I - Q)$ (the spectral radius of $I - Q$). Using this Taylor series expansion in the matrix decomposition above leads to the final result. □

Now, combining this theorem with the expression for the discounted value function V^π of a policy defined earlier in Equation (3.3), we finally obtain an expression for the discounted value function in terms of the Drazin inverse of the Laplacian.

Theorem 3.5. Given a discounted MDP M and policy π, the value function V^π can be expressed in terms of the *gain* and *bias* of the associated Markov reward process $M_\pi = (P^\pi, R^\pi)$, and the Drazin inverse of the Laplacian, as follows

$$V^\pi = (1 + \rho) \left(\rho^{-1} g^\pi + h^\pi + \sum_{n=0}^{\infty} (-\rho)^n ((\mathbb{L}^\pi)^D)^{n+1} R^\pi \right). \tag{3.21}$$

Proof. This result follows directly from the previous theorem and Equation (3.3). Note that $g^\pi = (P^\pi)^* R^\pi$ as defined earlier. Also, we showed earlier that $h^\pi = (I - P^\pi)(I - P^\pi + (P^\pi)^*)^{-1} R^\pi$. □

If we represent the coefficients in the Laurent series as $y_{-1}, y_0, \ldots,$ they can be shown to be solutions to the following set of equations (for $n = 1, 2, \ldots$). In terms of the expansion above, y_{-1} is the gain of the policy, y_0 is its bias, and so on.

$$\begin{aligned} \mathbb{L}^\pi y_{-1} &= 0 \\ y_{-1} + \mathbb{L}^\pi y_0 &= R^\pi \\ &\vdots \\ y_{n-1} + \mathbb{L}^\pi y_n &= 0 \end{aligned}$$

3.3.6 Examples

We now provide some simple examples to illustrate the Drazin expansion. Figure 3.3 shows a simple 7 state grid world MDP. If each compass action succeeds in moving the agent in the desired direction with probability 0.7, and leaves the agent in the same state with probability 0.3, the transition matrix of an optimal policy for reaching the goal marked G is given as

$$P = \begin{pmatrix} 0.30 & 0.70 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.30 & 0 & 0.70 & 0 & 0 & 0 \\ 0 & 0 & 0.30 & 0.70 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.30 & 0 & 0 & 0.70 \\ 0 & 0 & 0 & 0 & 0.30 & 0.70 & 0 \\ 0 & 0 & 0 & 0.70 & 0 & 0.30 & 0 \\ 0 & 0 & 0 & 0.3 & 0 & 0 & 0.7 \end{pmatrix}. \tag{3.22}$$

The invariant distribution ρ is given as the row vector

$$\rho = \begin{pmatrix} 0 & 0 & 0 & 0.3 & 0 & 0 & 0.7 \end{pmatrix} \tag{3.23}$$

The Drazin inverse of the Laplacian $\mathbb{L} = I - P$ is given as

$$\mathbb{L}^D = \begin{pmatrix} 1.2245 & 1.0204 & -0.2041 & 0.2463 & -0.2041 & -0.4082 & 0.0985 \\ -0.2041 & 1.0204 & -0.2041 & 0.2463 & -0.2041 & -0.4082 & 0.0985 \\ -0.2041 & -0.4082 & 1.2245 & 0.2463 & -0.2041 & -0.4082 & 0.0985 \\ -0.5041 & -0.7082 & -0.5041 & -0.0537 & -0.5041 & -0.7082 & -0.2015 \\ -0.2041 & -0.4082 & -0.2041 & 0.2463 & 1.2245 & 1.0204 & 0.0985 \\ -0.2041 & -0.4082 & -0.2041 & 0.2463 & -0.2041 & 1.0204 & 0.0985 \\ -0.9041 & -1.1082 & -0.9041 & -1.6606 & -0.9041 & -1.1082 & -0.0842 \end{pmatrix} \tag{3.24}$$

Notice the highly regular structure of the entries in the above matrix. This regularity is not a coincidence, but in fact the key reason why the Drazin inverse of the Laplacian provides an effective way to compress MDPs. Many interesting properties of the structure of Markov chains are captured by $\mathbb{L}^D$ (or equivalently $\mathbb{L}^{\#}$, the group inverse). In particular, the following properties can be shown (see [20] for a more detailed analysis):

- For a general Markov chain, states s_i and s_k belong to the same ergodic set if and only if the i^{th} and k^{th} rows of $I - \mathbb{L}\mathbb{L}^{\#}$ are equal.
- If states s_i and s_k are transient states, then $\mathbb{L}^{\#}_{ik}$ is the expected number of times the chain is in state s_k after starting in s_i. Also, s_i and s_k are in the same transient set if and only if $\mathbb{L}^{\#}_{ik} > 0$ and $\mathbb{L}^{\#}_{ki} > 0$.
- State s_i is a transient state if and only if the i^{th} column of $I - \mathbb{L}\mathbb{L}^{\#}$ is 0.

Figure 3.2 gives an example of how the framework of Drazin inverses of the Laplacian can be used to approximate value functions. Even though the value function is highly nonlinear due to the presence of walls and nonlinearities, it can be effectively compressed by projecting it onto a basis generated from the Drazin inverse of the Laplacian associated with a policy. We will describe a more complete evaluation of these and other bases in Section 8.

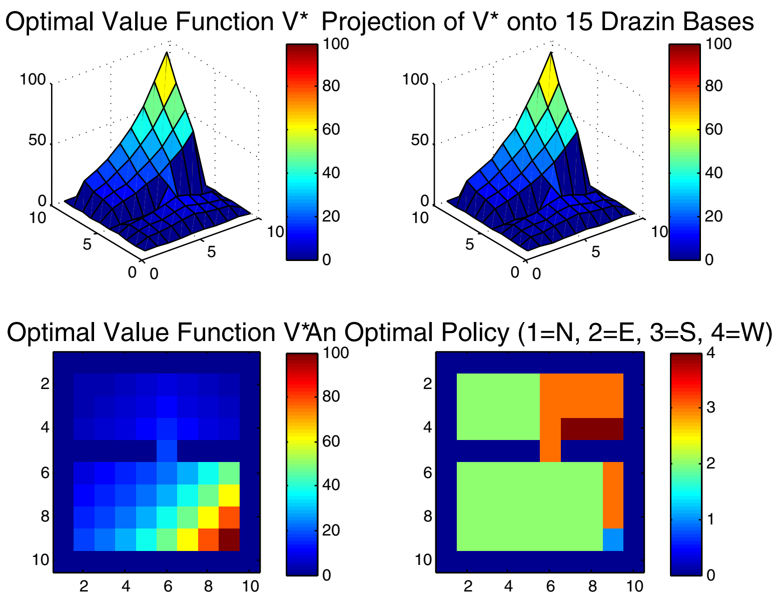


Fig. 3.2 This figure illustrates how the Drazin inverse of the Laplacian can be used to approximately solve MDPs. The value function shown is highly nonlinear due to walls. The approximation shown was computed by projecting the (known) optimal value function on a set of basis vectors generated from the Drazin inverse of the Laplacian associated with the optimal policy π. Although the problem has 100 states, the optimal value function is effectively compressed onto a subspace of dimension 15.

3.4 Positive-Semidefinite Laplacian Matrices

In this section, we introduce a more restricted family of *positive-semidefinite* (PSD) Laplacian matrices that have been the topic of significant recent work in machine learning.[6] A growing body of work in machine learning on nonlinear dimensionality reduction [73], manifold learning [8, 30, 116, 131], and representation discovery [80] exploit the remarkable properties of the PSD Laplacian operator on manifolds [115], many shared by its discrete counterpart, the graph Laplacian [2, 26, 44]. Although these cannot be directly used to model transition matrices in MDPs as the more generalized Laplacians described

[6] A positive-semidefinite matrix A is one where $v^T Av \geq 0$ for all $v \neq 0$. PSD matrices have real nonnegative eigenvalues and real-valued eigenvectors.

previously, they are still of significant interest in finding approximate solutions to MDPs, as will be described below.

3.4.1 Diffusion Models and Random Walks on Graphs

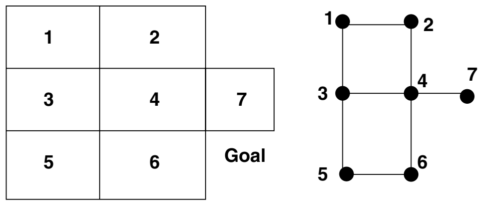
Consider a weighted graph $G = (V, E, W)$, where V is a finite set of vertices, and W is a weighted adjacency matrix with $W(i,j) > 0$ if $(i,j) \in E$, that is, it is possible to reach state i from j (or vice-versa) in a single step. A simple example of a diffusion model on G is the random walk matrix $P_r = D^{-1}W$. Figure 3.3 illustrates a random walk diffusion model. Note the random walk matrix $P_r = D^{-1}W$ is not symmetric. However, it can be easily shown that P_r defines a *reversible* Markov chain, which induces a vector (Hilbert) space with respect to the inner product defined by the invariant distribution ρ:

$$\langle f, g\rangle_\rho = \sum_{i \in V} f(i) g(i) \rho(i).$$

In addition, the matrix P_r can be shown to be self-adjoint with respect to the above inner product, that is

$$\langle P_r f, g\rangle_\rho = \langle f, P_r g\rangle_\rho.$$

Consequently, the matrix P_r can be shown to have real eigenvalues and orthonormal eigenvectors, with respect to the above inner product.



$$P_r = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0.33 & 0 & 0 & 0.33 & 0.33 & 0 & 0 \\ & & & \cdots & & & \end{bmatrix}$$

Fig. 3.3 *Top*: A simple diffusion model given by an undirected unweighted graph connecting each state to neighbors that are reachable using a single (reversible) action. *Bottom*: First three rows of the random walk matrix $P_r = D^{-1}W$. P_r is not symmetric, but self-adjoint with respect to the invariant distribution, and consequently has real eigenvalues and eigenvectors.

The random walk matrix $P_r = D^{-1}W$ is called a *diffusion model* because given any function f on the underlying graph G, the powers of $P_r^t f$ determine how quickly the random walk will "mix" and converge to the long term distribution [26]. It can be shown that the stationary distribution of a random walk on an undirected graph is given by $\rho(v) = \frac{d_v}{\mathrm{vol}(G)}$, where d_v is the degree of vertex v and the "volume" $\mathrm{vol}(G) = \sum_{v \in G} d_v$.

3.4.2 Graph Laplacians

Even though the random walk matrix P_r can be diagonalized, for computational reasons, it turns out to be highly beneficial to find a symmetric PSD matrix with a closely related spectral structure. These are the popular graph Laplacian matrices, which we now describe in more detail. The main idea underlying some of the graph–theoretic methods for basis construction is to view approximating the value function of a MDP as that of regularizing functions on state space graphs. We develop this point of view in detail in this section, showing how the Laplacian provides a highly effective regularization framework [98].

For simplicity, assume the underlying state space is represented as an undirected graph $G = (V, E, W)$, where V is the set of vertices, and E is the set of edges where $(u, v) \in E$ denotes an undirected edge from vertex u to vertex v. The *combinatorial Laplacian* L is defined as the operator $L = D - W$, where D is a diagonal matrix called the *valency* matrix whose entries are row sums of the weight matrix W. The first three rows of the combinatorial Laplacian matrix for the grid world MDP in Figure 3.3 is illustrated below, where each edge is assumed to have a unit weight:

$$L = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 3 & -1 & -1 & 0 & 0 \\ & & & \cdots & & & \end{bmatrix}.$$

Comparing the above matrix with the random walk matrix in Figure 3.3, it may seem like the two matrices have little in common. Surprisingly, there is indeed an intimate connection between the random walk matrix and the Laplacian. The Laplacian has many attractive spectral properties. It is both symmetric as well as PSD, and hence its eigenvalues are not only all real, but also nonnegative. To understand this property, it helps to view the Laplacian as an *operator* on the space of functions $\mathcal{F} : V \to \mathbb{R}$ on a graph. In particular, the Laplacian acts as a *difference* operator.

$$Lf(i) = \sum_{j \sim i} (f(i) - f(j)), \quad (i,j) \in E.$$

On a 2D grid, the Laplacian can be shown to essentially be a discretization of the continuous Laplace operator in Euclidean space:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2},$$

where the partial derivatives are replaced by finite differences.

3.4.3 Reproducing Kernels and the Graph Laplacian

The crucial idea of using the graph Laplacian as a regularizer is that rather than smoothing using properties of the ambient Euclidean space, smoothing takes the underlying manifold (or graph) into account. We develop this notion more formally in this section, by showing that the graph Laplacian induces a Reproducing Kernel Hilbert Space (RKHS) [54].

Definition 3.12. The Laplacian $L = D - W$ defines a *semi-norm* over all functions on an undirected graph G

$$\langle f,g \rangle = f^T Lg. \tag{3.25}$$

Furthermore, the length of a function is defined as $\|f\| = \sqrt{\langle f,f \rangle}$.

Since L is symmetric, the properties of a norm can be directly verified, except that for constant functions, $\langle f,f \rangle = 0$, thus, violating one of the properties of a norm. However, as we will show below, the Laplacian induces a regular norm over a more restricted space. Informally, the smaller $\|f\|$ is, the "smoother" f is on the graph. This intuition can be formalized as follows.

Definition 3.13. The *Dirichlet sum* is defined as

$$f^T L f = \sum_{(u,v)\in E} w_{uv}(f(u) - f(v))^2 \ . \tag{3.26}$$

Thus, for highly "smooth" functions, $f_i \approx f_j, (i,j) \in E$ and w_{ij} is small. More formally, a fundamental property of the graph Laplacian is that projections of functions on the eigenspace of the Laplacian produce the smoothest global approximation respecting the underlying graph topology.

Note that the Laplacian $L = D - W$ is singular since its smallest eigenvalue $\lambda_1 = 0$. In fact, it can be shown that for graphs with r components, the 0 eigenvalue has multiplicity r, and the corresponding eigenspace defined by the r associated eigenvectors spans the kernel of L (where the kernel of L is the space spanned by all x such that $Lx = 0$). To define the reproducing kernel associated with L, we define a restricted Hilbert space of all functions that span the complement of the kernel, that is, all functions that are orthogonal to the eigenvectors associated with the 0 eigenvalue.

Definition 3.14. Let the eigenvalues of L on a graph G with r connected components be ordered as $\lambda_1 = 0, \ldots, \lambda_r = 0$, $\lambda_{r+1} > 0, \ldots, \lambda_n > 0$. Let the associated eigenvectors as $u_i, 1 \leq i \leq n$. The Hilbert space associated with a graph G is defined as

$$\mathbb{H}(G) = \{g : g^T u_i = 0, 1 \leq i \leq r\}. \tag{3.27}$$

Clearly, $\|f\| = \sqrt{\langle f, f\rangle} = \sqrt{f^T L f}$ now defines a norm because $\|f\| > 0$. We now show that $\mathbb{H}(G)$ actually defines an RKHS as well, and its reproducing kernel is $K = L^+$, namely the pseudo-inverse of L. First, note that we can write the pseudo-inverse of L as follows.

Definition 3.15. The *pseudo-inverse* of Laplacian $L = D - W$ is defined as

$$L^+ \equiv \sum_{i=r+1}^{n} \frac{1}{\lambda_i} u_i u_i^T, \tag{3.28}$$

where the graph G is assumed to be undirected and unweighted, and having r connected components.

The definition easily generalizes to weighted (and directed) graphs. The main result we need to show is as follows.

Theorem 3.6. The reproducing kernel associated with the RKHS $\mathbb{H}(G)$ of an undirected unweighted graph G is given by

$$K = L^{+}. \tag{3.29}$$

Proof. Clearly, K is symmetric from the definition of L^{+}. Indeed, note also that

$$LL^{+} = I - \sum_{i=1}^{r} u_i u_i^T. \tag{3.30}$$

This result follows directly from the expansion of L^{+} and because U, the matrix of eigenvectors of L is orthogonal (hence, $UU^T = I$). More concretely,

$$\begin{aligned} LL^{+} &= \left(\sum_{i=1}^{n} \lambda_i u_i u_i^T \right) \left(\sum_{i=r+1}^{n} \frac{1}{\lambda_i} u_i u_i^T \right) \\ &= \left(\sum_{i=r+1}^{n} \lambda_i u_i u_i^T \right) \left(\sum_{i=r+1}^{n} \frac{1}{\lambda_i} u_i u_i^T \right) \\ &= \left(\sum_{i=r+1}^{n} u_i u_i^T \right) \\ &= I - \sum_{i=1}^{r} u_i u_i^T. \end{aligned}$$

Thus, given any function $g \in \mathbb{H}(G)$, we have $LL^{+}g = g$. Thus, it follows that

$$g(i) = e_i L^{+} L g = K(:,i) L g = \langle K(:,i), g \rangle,$$

where $K(:,i)$ is the ith column of K, and e_i is the ith unit basis vector. □

Given that the Laplacian L defines an RKHS, it is possible to define the problem of minimum-norm interpolation of functions on a graph G.

Definition 3.16. Given an undirected graph G, whose Laplacian $L = D - W$, given samples of a function f on l vertices of G, the minimum-norm interpolant in $\mathbb{H}(G)$ to f is given as follows

$$\min_{g \in \mathbb{H}(G)} \{\|g\| : g_i = f_i, i = 1, \ldots, l\}. \tag{3.31}$$

Where it is assumed without loss of generality that the function values are known on the first l vertices. A variety of different solutions to this minimum-norm problem can be developed. For example, one approach is to use a regularized least-square approach, which requires solving a linear system of equations of $|G| = n$ equations [98]. Another approach is to use the representer theorem [139], which states that the solution g can be expressed as the linear sum of kernel evaluations on the available samples.

$$g(i) = \sum_{j=1}^{l} K(i,j) c_j, \tag{3.32}$$

where the coefficients c are computed as the solution to the equation

$$c = \hat{K}^{+} f, \tag{3.33}$$

where $\hat{K} = K(1:l, 1:l)$. We will discuss the application of these ideas to basis construction in MDPs in Section 6.

3.4.4 Random Walks and the Laplacian

To make the connection between the random walk operator P_r and the Laplacian, the *normalized Laplacian* [26] needs to be introduced:

$$\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}.$$

To see the connection between the normalized Laplacian and the random walk matrix $P_r = D^{-1}W$, note the following identities:

$$\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}, \tag{3.34}$$

$$I - \mathcal{L} = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}, \tag{3.35}$$

$$D^{-\frac{1}{2}} (I - \mathcal{L}) D^{\frac{1}{2}} = D^{-1} W. \tag{3.36}$$

Hence, the random walk operator $D^{-1}W$ is similar to $I - \mathcal{L}$, so both have the same eigenvalues, and the eigenvectors of the random walk operator are the eigenvectors of $I - \mathcal{L}$ point-wise multiplied by $D^{-\frac{1}{2}}$. In particular, if λ_i is an eigenvalue of the random walk transition matrix P_r, then $1 - \lambda_i$ is the corresponding eigenvalue of $\mathcal{L}$.

The normalized Laplacian $\mathcal{L}$ also acts as a *difference* operator on a function f on a graph, that is

$$\mathcal{L}f(u) = \frac{1}{\sqrt{d_u}} \sum_{v \sim u} \left(\frac{f(u)}{\sqrt{d_u}} - \frac{f(v)}{\sqrt{d_v}} \right) w_{uv}. \tag{3.37}$$

The difference between the combinatorial and normalized Laplacian is that the latter models the degree of a vertex as a local measure.

4

Approximating Markov Decision Processes

In this section, we introduce a variety of methods for approximately solving MDPs, including least-squares, linear programming, and Hilbert space methods. All these approaches depend crucially on a choice of basis functions for constructing a low-dimensional representation of a MDP, and assume these are provided by the human designer. The focus of this section is not on basis construction, but rather on approximation. The problem of approximation is naturally formulated using the theory of vector spaces. We assume the reader is familiar with finite-dimensional vector spaces, as covered in standard undergraduate textbooks [127]. However, it will be convenient to generalize the usual matrix-based coordinate-dependent framework to a richer framework based on a type of vector space called a Hilbert space [37].

4.1 Linear Value Function Approximation

It is obviously difficult to represent value functions exactly on large discrete state spaces, or in continuous spaces. Furthermore, even in discrete spaces where it is possible to store value functions exactly, the use of a well-designed basis can greatly accelerate learning as it propagates corrections to the value function across large regions of the state space. Consequently, there has been much study of architectures for approximating value functions [12]. We introduce a general theoretical framework for value function approximation based on Hilbert space, a type of vector space, i.e., much studied in approximation theory [37] and machine learning [120].

4.1.1 A Hilbert Space Formulation

We address the problem of approximating the value function V^{π} associated with a fixed policy π, before considering the more difficult problem of approximation of the optimal policy V^{*}. An elegant and general way to formalize value function approximation builds on the framework of approximation in a type of vector space called a Hilbert space [37]. We follow the Hilbert space formulation of value function approximation given in [138], which the interested reader can refer to for additional details. Some readers may find the mathematics in this section to be somewhat abstract. This section can be omitted without loss of continuity, and readers can directly proceed to Section 4.2 where two specific projection methods are described using finite-dimensional matrix theory.

A Hilbert space $\mathcal{H}$ is a vector space equipped with an abstract notion of "length" defined by an "inner product" $\langle f, g\rangle$ between any two vectors $f, g \in \mathcal{H}$. As an example, in the familiar Euclidean space $\mathbb{R}^n$, which is a Hilbert space, the inner product of two vectors $\langle f, g\rangle = f^T g$, viewing f and g as column vectors. In a general MDP, the concept of distance needs to be modified from that in Euclidean distance by a nonuniform "weighting" that depends on the frequency with which different states are visited by a particular policy.

We now define an inner product, i.e., induced by the invariant distribution ρ^{π} of a policy. We assume that the MDP is ergodic, and as seen in Section 3, the limiting matrix $(P^{\pi})^{*} = \mathbf{1}\rho^{\pi}$.

Definition 4.1. The space of all value functions on a discrete MDP forms a Hilbert space under the inner product induced by the invariant distribution ρ^π of a specific policy π, where

$$\langle V_1, V_2 \rangle_{\rho^\pi} = \sum_{s \in S} V_1^\pi(s) V_2^\pi(s) \rho^\pi(s). \tag{4.1}$$

The "length" or norm in this inner product space is defined as $\|V\|_{\rho^\pi} = \sqrt{\langle V, V \rangle_{\rho^\pi}}$.[1]

The concept of projection is crucial in defining approximation in a Hilbert space. Intuitively, the idea is to find the element of a (closed) subspace that is "closest" to a given vector. In value function approximation, we are interested in finding the "best" approximation to the vector V^π that lies in the subspace spanned by a set of basis functions.

Definition 4.2. The projection operator $\Pi_{\mathcal{K}} : \mathcal{H} \to \mathcal{K}$ finds the closest element $\hat{u} \in \mathcal{K}$ to $u \in \mathcal{H}$, that is

$$\Pi_{\mathcal{K}}(u) = \operatorname*{argmin}_{g \in \mathcal{K}} \|g - u\|_{\mathcal{H}}. \tag{4.2}$$

Any projection operator Π is idempotent, so that $\Pi^2 = \Pi$ and nonexpansive, so that $\|\Pi(u)\| \leq \|u\|$.

If the basis functions ϕ_i are *orthonormal*, meaning that $\langle \phi_i, \phi_j \rangle_{\mathcal{H}} = \delta_{ij}$, where $\delta_{ij} = 1$ if and only if $i = j$, then the projection operator can be defined by the following *abstract* Fourier series.

Definition 4.3. If $\phi_1, \ldots, \phi_k$ is a basis for the subspace $\mathcal{K}$, the projection operator $\Pi_{\mathcal{K}}$ operator can be written as

$$\Pi_{\mathcal{K}}(u) = \sum_{i=1}^{k} \langle u, \phi_i \rangle_{\mathcal{H}} \phi_i. \tag{4.3}$$

---

[1] Technically, ρ^π must be a strictly positive distribution for $\langle ., . \rangle_{\rho^\pi}$ to form a valid inner product, which implies that the Markov chain can have no transient states s where $\rho^\pi(s) = 0$. A minor change allows extending the definition to arbitrary MDPs [138].

Applying this definition to approximation of the value function V^π, we have

$$V_\Phi^\pi = \Pi_\Phi V^\pi = \sum_{i=1}^{k} \langle V^\pi, \phi_i \rangle_{\rho^\pi} \ \phi_i, \tag{4.4}$$

where the projection operator onto the space spanned by the bases ϕ_i is indicated as Π_Φ for convenience. In the next section, we will investigate a "fixed point" algorithm for finding the best approximation to V^π. This algorithm works by finding the fixed point of the *composition* of the T^π operator and the projection operator Π_Φ. A natural question that arises is whether the error in such algorithms for value function approximation can be quantified. A convenient way to arrive at such results is through the concept of a contraction mapping.

Definition 4.4. An operator T on a Hilbert space $\mathcal{H}$ is called a *contraction mapping* if and only if

$$\|Tu - Tv\|_{\mathcal{H}} \leq \beta \|u - v\|_{\mathcal{H}}, \tag{4.5}$$

where $0 < \beta < 1$.

It can be shown that the operator T^π is a contraction mapping, where

$$\|T^\pi V_1 - T^\pi V_2\|_{\rho^\pi} \leq \gamma \|V_1 - V_2\|_{\rho^\pi}. \tag{4.6}$$

A standard method for computing least-square projections for value function approximation is to find the projection of the backed up value function $T^\pi(V^\pi)$ onto the space spanned by the bases Φ. This procedure is motivated by the observation that although the current approximation $\hat{V}$ may lie within the subspace spanned by the bases, the backed up value function may lie outside the subspace, and hence need to be projected back in. Thus, we are interested in investigating the properties of the *composite* operator $\Pi_\Phi T^\pi$. Since projections are nonexpansive, the composite operator is also a contraction mapping.

Theorem 4.1. The composite projection and backup operator $\Pi_\Phi T^\pi$ is a contraction mapping with a factor $\kappa \leq \gamma$.

Proof. We know that T^π is a contraction mapping with a factor γ. It follows easily from the nonexpansion property of Π_Φ that the combined operator is also a contraction mapping, since

$$\|\Pi_\Phi T^\pi V_1 - \Pi_\Phi T^\pi V_2\|_{\rho^\pi} \leq \|T^\pi V_1 - T^\pi V_2\|_{\rho^\pi} \leq \gamma \|V_1 - V_2\|_{\rho^\pi}. \qquad \square$$

Since the composite operator is a contraction mapping, it must also have a fixed point.

Definition 4.5. The fixed point of the combined operator $\Pi_\Phi T^\pi$ is defined as

$$\hat{V}_\Phi^\pi = \Pi_\Phi T^\pi \hat{V}_\Phi^\pi. \tag{4.7}$$

Exploiting the contraction property of the composite operator $\Pi_\Phi T^\pi$ the following general error bound establishes bounds on the "distance" between the true value function V^π and the fixed point approximation $\hat{V}_\Phi^\pi$.

Theorem 4.2. Let $\hat{V}_\Phi^\pi$ be the fixed point of the combined projection operator Π_Φ and the backup operator T^π. Then, it holds that

$$\|V^\pi - \hat{V}_\Phi^\pi\|_{\rho^\pi} \leq \frac{1}{\sqrt{1-\kappa^2}} \|V^\pi - \Pi_\Phi V^\pi\|_{\rho^\pi}. \tag{4.8}$$

κ is the contraction rate of the composite operator.

Proof.

$$\begin{aligned}
\|V^\pi - \hat{V}_\Phi^\pi\|_{\rho^\pi}^2 &= \|V^\pi - \Pi_\Phi V^\pi + \Pi_\Phi V^\pi - \hat{V}_\Phi^\pi\|_{\rho^\pi}^2 \\
&= \|V^\pi - \Pi_\Phi V^\pi\|_{\rho^\pi}^2 + \|\Pi_\Phi V^\pi - \hat{V}_\Phi^\pi\|_{\rho^\pi}^2 \\
&= \|V^\pi - \Pi_\Phi V^\pi\|_{\rho^\pi}^2 + \|\Pi_\Phi T^\pi(V^\pi) - \Pi_\Phi T^\pi(\hat{V}_\Phi^\pi)\|_{\rho^\pi}^2 \\
&\leq \|V^\pi - \Pi_\Phi V^\pi\|_{\rho^\pi}^2 + \kappa^2 \|V^\pi - \hat{V}_\Phi^\pi\|_{\rho^\pi}^2.
\end{aligned}$$

The result follows directly from the last inequality. The second step follows from the use of the generalized Pythogorean theorem in Hilbert spaces (the length of the sum of two orthogonal vectors is the sum of the lengths of each vector). Note that the error vector $V^\pi - \Pi_\Phi V^\pi$ is orthogonal to all value functions in the space spanned by the bases Φ, because of the property of orthogonal projectors. The inequality follows from Theorem 4.1. □

4.2 Least-Squares Approximation of a Fixed Policy

In this section, we review two standard approaches to approximating value functions using a linear combination of basis functions, which are known as the *Bellman residual* approach [122, 94], and the *fixed point* approach [69, 94] Assume a set of *basis functions* $F_\Phi = \{\phi_1, \ldots, \phi_k\}$ is given, where each basis function represents a "feature" $\phi_i : S \to \mathbb{R}$. The basis function matrix Φ is an $|S| \times k$ matrix, where each column represents a basis function, and each row specifies the value of all the basis functions in a particular state. We assume the Bellman backup operator T^π is known, and then proceed to discuss how to extend these approaches when T^π is unknown and must be estimated from samples.

Definition 4.6. The *Bellman Residual minimization* problem is to find a set of weights w_{BR} such that

$$w_{\mathrm{BR}} = \underset{w}{\operatorname{argmin}} \, \|T^\pi(\hat{V}) - \hat{V}\|_{\rho^\pi}, \tag{4.9}$$

where $\hat{V}$ is an initial approximation to V^π.

We can express the minimization as a "least-squares" problem as follows

$$\begin{aligned}
&\min_w \|T^\pi(\hat{V}) - \hat{V}\|_{\rho^\pi} \\
&\quad = \min_w \|R^\pi + \gamma P^\pi \hat{V} - \hat{V}\|_{\rho^\pi} \\
&\quad = \min_w \|R^\pi + \gamma P^\pi \Phi w - \Phi w\|_{\rho^\pi} \\
&\quad\quad (I - \gamma P^\pi)\Phi w \approx_{\rho^\pi} R^\pi
\end{aligned}$$

The last step formulates the equivalence as a weighted least-squares problem of the form $Ax \approx_\rho b$. It is well known that the solution to a weighted least-squares problem can be expressed generically as

$$x = (A^T C A)^{-1} A^T C b,$$

where C is a diagonal matrix whose entries specify the non-uniform weights measuring the "length" in the space. For Bellman residual minimization, the corresponding elements are:

$$A_{\mathrm{BR}} = (I - \gamma P^\pi)\Phi, \quad C = D_{\rho^\pi}, \quad b = R^\pi.$$

Thus, the Bellman residual minimization problem can be solved as the following least-squares solution:

$$w_{\mathrm{BR}} = (A_{\mathrm{BR}}^T D_{\rho^\pi} A_{\mathrm{BR}})^{-1}(A_{\mathrm{BR}}^T D_{\rho^\pi} R^\pi) \tag{4.10}$$

Another popular approach is to find the fixed point of the combined projection operator Π_Φ and the Bellman “backup operator T^π.

Definition 4.7. The *Bellman Fixed Point* method for linear value function approximation is to find the weights w_{FP} such that

$$w_{\mathrm{FP}} = \underset{w}{\operatorname{argmin}} \|\Pi_\Phi T^\pi(\hat{V}) - \hat{V}\|_{\rho^\pi}. \tag{4.11}$$

A similar least-squares solution to the fixed point method can be derived as follows

$$\begin{aligned}
&\min_w \|\Pi_\Phi T^\pi(\hat{V}) - \hat{V}\|_{\rho^\pi} \\
&\qquad = \min_w \|\Pi_\Phi T^\pi \Phi w - \Phi w\|_{\rho^\pi} \\
&\Pi_\Phi T^\pi \Phi w \approx_{\rho^\pi} \Phi w \\
&\Phi w = \Phi(\Phi^T D_{\rho^\pi}\Phi)^{-1}\Phi^T D_{\rho^\pi}(R + \gamma P^\pi \Phi w) \\
&w_{\mathrm{FP}} = (\Phi^T D_{\rho^\pi}\Phi - \gamma \Phi^T D_{\rho^\pi} P^\pi \Phi)^{-1}\Phi^T D_{\rho^\pi} R^\pi.
\end{aligned}$$

Figure 4.1 gives a geometrical perspective of the problem of value function approximation. The Bellman residual approach can be seen as

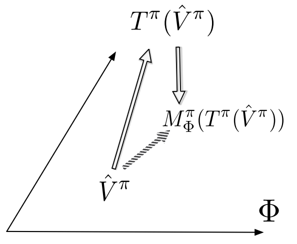


Fig. 4.1 A geometrical view of value function approximation.

minimizing the length of the vector $T^{\pi}(\hat{V}^{\pi}) - \hat{V}^{\pi}$, whereas the fixed point approach minimizes the projection of the residual in the space spanned by the basis Φ.

4.2.1 Least-Squares Temporal Difference Learning

In Section 2.3, we described the temporal difference TD(0) learning algorithm. Here, we generalize this algorithm and describe the LSTD method [16, 18]. Least-Square Temporal Difference (LSTD) computes the approximation to the value function V^{π} on a specific set of bases Φ. It can be viewed as an iterative approximation of the fixed-point projection method described above (see Equation (4.11)). Essentially, the algorithm maintains a matrix (denoted as A) and a column vector b, which are both updated from sampled trajectories. When `flag` is true, namely whenever the approximated value function is needed, the algorithm returns $w = A^{-1}b$ as the answer. The algorithm shown is the LSTD(λ) method for the undiscounted $\gamma = 1$ setting [16]. Here, $0 \leq \lambda \leq 1$ is a weighting parameter that determines the *eligibility* or recency of past states.

LSTD(λ) can be viewed as building a compressed model on the basis Φ. To gain some insight into the meaning of the A matrix and b column vector, let us assume that the basis matrix Φ is the identity, that is, the unit vector basis is used. Then, line 6 of the LSTD algorithm, for $\lambda = 0$, computes the sum

$$A = A + \phi(x_t)(\phi(x_t) - \phi(x_{t+1}))^T,$$

where $\phi(x_t)$ is a column vector of all 0 except for a single 1 at the location corresponding to state x_t. Thus, the outer product is a matrix with all 0's except at the row corresponding to state x_t, which has a 1 in the diagonal entry and a -1 at the column corresponding to state x_{t+1}. The sum of all these rank one matrices is

$$A_f = (N - T),$$

where N is a diagonal matrix counting the number of times that each state was visited, and T is a matrix that counts the number of transitions from state x_t to state x_{t+1}. Similarly, line 7 in the LSTD(0)

Algorithm 4 The LSTD(λ) Algorithm

```
// $A$: A matrix of size $k \times k$ initialized to the 0 matrix.
// $b$: A vector of size $k \times 1$ initialized to 0 vector.
// $t$: time counter initialized to 0.
// flag: A binary variable set to true when the coefficients are desired.
1: for $i = 1$ to $N$ do
2:     Choose a start state $x_t \in S$.
3:     Set $z_t = \phi(x_t)$
4:     while $x_t \neq$ END do
5:         Execute action $\pi(x_t)$, resulting in state $x_{t+1}$ and reward $R_t$
6:         Set $A = A + z_t(\phi(x_t) - \phi(x_{t+1}))^T$
7:         Set $b = b + z_t R_t$
8:         $z_{t+1} = \lambda z_t + \phi(x_{t+1})$
9:         $t = t + 1$
10:    end while
11:    if flag then
12:        Return $w = A^\dagger b$
13:    end if
14: end for
```

algorithm consists of the vector sum

$$b = b + \phi(x_t) R_t,$$

which is simply a vector of sum of rewards received on transitions out of each state x_t. The matrix pseudo-inversion step in line 12 of the LSTD algorithm can be written as

$$V^\pi = (N - T)^\dagger s,$$

where s is a column vector containing the sum of rewards received in each state. When Φ is not the identity matrix, then A can be shown to be computing a compressed model.

4.3 Approximation in Learning Control

The problem of value function approximation in control learning is significantly more difficult in that only samples of the MDP are available, from which T^π must be constructed. A standard algorithm for control learning is *approximate policy iteration* [12], which interleaves an *approximate policy evaluation* step of finding an approximation of the value function $\hat{V}^{\pi_k}$ associated with a given policy π_k at stage k, with a *policy improvement* step of finding the greedy policy associated with $\hat{V}^{\pi_k}$. Here, there are two sources of error introduced by approximating the exact value function, and approximating the policy. A specific type of approximate policy iteration method — the Least-Square Policy Iteration (LSPI) algorithm [69] — is described in Section 4.3.1, which uses a least-squares approach to approximate the action-value function to alleviate this problem.

4.3.1 Approximation of Action-Value Functions

Here, we focus on action-value function approximation, and in particular, describe the LSPI method [69]. The complete algorithm is described in Figure 4.2. In action-value learning, the goal is to approximate the true action-value function $Q^\pi(s,a)$ for a policy π using a set of basis functions $\phi(s,a)$ that can be viewed as compressing the space of action-value functions. The true action-value function $Q^\pi(s,a)$ is a vector in a high-dimensional space $\mathbb{R}^{|S|\times|A|}$, and using the basis functions amounts to reducing the dimension to $\mathbb{R}^k$, where $k \ll |S| \times |A|$. The approximated action value is thus

$$\hat{Q}^\pi(s,a;w) = \sum_{j=1}^{k} \phi_j(s,a) w_j,$$

where the w_j are weights or parameters that can be determined using a LSM. Let Q^π be a real (column) vector $\in \mathbb{R}^{|S|\times|A|}$. $\phi(s,a)$ is a real vector of size k where each entry corresponds to the basis function $\phi_j(s,a)$ evaluated at the state action pair (s,a). The approximate action-value function can be written as $\hat{Q}^\pi = \Phi w^\pi$, where w^π is a real column vector of length k and Φ is a real matrix with $|S| \times |A|$ rows and k columns. Each row of Φ specifies all the basis functions for a particular state action pair (s,a), and each column represents the value of a particular basis function over all state action pairs. As described in Section 4.1, the Bellman fixed point approximation tries to find a set of weights w^π

```
LSPI ($T, N, \epsilon, \Phi$):

// $T$: Number of episodes for sample collection
// $N$: Maximum length of each trial
// $\epsilon$ : Convergence condition for policy iteration
// $\Phi$: Basis function matrix

                    Sample Collection Phase

    • Collect a data set of samples $\mathcal{D} = \{(s_i, a_i, s_{i+1}, r_i), \ldots\}$ by following
      a random policy for a set of $T$ trials, each of maximum $N$ steps.

                    Control Learning Phase

    • Set $i = 0$. Initialize $w^i \in \mathbb{R}^k$ to a random vector.
    • Repeat the following steps:
        (1) Define $\pi^i(s) \in \text{argmax}_a \hat{Q}(s, a)$, where $\hat{Q}(s, a) = \Phi w^i$.
        (2) Set $i \leftarrow i + 1$. Using the stored transitions
            $(s_t, a_t, s'_t, a'_t, r_t) \in \mathcal{D}$, compute the matrix $A$ and vec-
            tor $b$ as follows:
                $\tilde{A}^{t+1} = \tilde{A}^t + \phi(s_t, a_t)\Big(\phi(s_t, a_t) - \gamma\phi(s'_t, \pi^i(s_t))\Big)^T.$
                $\tilde{b}^{t+1} = \tilde{b}^t + \phi(s_t, a_t) r_t.$
        (3) Solve the linear system of equations $\tilde{A} w^i = \tilde{b}$ using any
            standard method.
        (4) until $\|w^i - w^{i+1}\|^2 \leq \epsilon$.
    • Set $\pi(s) \in \text{argmax}_{a \in A} \hat{Q}^i(s, a)$, where $\hat{Q}^i = \Phi w^i$ is the $\epsilon$-optimal
      approximation to the optimal value function within the linear span
      of basis functions $\Phi$.
```

Fig. 4.2 This figure describes a least-squares variant of approximate policy iteration.

under which the projection of the backed up approximate Q-function $T_\pi \hat{Q}^\pi$ onto the space spanned by the columns of Φ is a fixed point, namely

$$\hat{Q}^\pi = \Pi_\Phi T^\pi(\hat{Q}^\pi),$$

where T_π is the Bellman backup operator. It was shown in the previous section that the resulting solution can be written in a *weighted* least-squares form as $Aw^\pi = b$, where the A matrix is given by

$$A = \left(\Phi^T D_{\rho^\pi}(\Phi - \gamma P^\pi \Phi)\right),$$

and the b column vector is given by

$$b = \Phi^T D_{\rho^\pi} R,$$

where D_{ρ^π} is a diagonal matrix whose entries reflect varying "costs" for making approximation errors on (s,a) pairs as a result of the nonuniform distribution $\rho^\pi(s,a)$ of visitation frequencies. A and b can be estimated from a database of transitions collected from some source, e.g., a random walk. The A matrix and b vector can be estimated as the sum of many rank-one matrix summations from a database of stored samples.

$$\begin{aligned}\tilde{A}^{t+1} &= \tilde{A}^t + \phi(s_t,a_t)\left(\phi(s_t,a_t) - \gamma\phi(s'_t,\pi(s'_t))\right)^T,\\ \tilde{b}^{t+1} &= \tilde{b}^t + \phi(s_t,a_t)r_t,\end{aligned}$$

where (s_t,a_t,r_t,s'_t) is the tth sample of experience from a trajectory generated by the agent (using some random or guided policy). Once the matrix A and vector b have been constructed, the system of equations $Aw^\pi = b$ can be solved for the weight vector w^π either by taking the inverse of A (if it is of full rank) or by taking its pseudo-inverse (if A is rank-deficient). This defines a specific policy since $\hat{Q}^\pi = \Phi w^\pi$. The process is then repeated, until convergence (which can be defined as when the normed difference between two successive weight vectors falls below a pre-defined threshold ϵ). Note that in succeeding iterations, the A matrix will be different since the policy π has changed.

4.4 Approximation Using Convex Optimization

A different class of approaches from least-squares for solving MDPs are those based on *convex optimization* [17]. Two methods that fall into this class are reviewed now: approximate linear programming [36] and reproducing kernel based methods that use quadratic programming [120].

4.4.1 Approximate Linear Programming

A straightforward way to generalize exact linear programming using basis functions is given below. This approach is referred to as *approximate linear programming* in the literature [36].

Definition 4.8. The approximate linear program required to solve a MDP M given a set of bases Φ is:

$$\begin{aligned}
\text{Variables} &: w_1, \ldots, w_k \\
\text{Minimize} &: \sum_s \alpha_s \sum_i w_i \phi_i(s) \\
\text{Subject to} &: \sum_i w_i \phi_i(s) \geq \sum_{s'} P^a_{ss'} \left(R^a_{ss'} + \gamma \sum_i w_i \phi_i(s) \right), \\
&\qquad \forall\ s \in S, \quad a \in A_s,
\end{aligned}$$

where α is a state relevance weight vector whose weights are all positive.

Note that the number of variables in the LP has now been reduced from $|S| = n$ to $k \ll |S|$. Unfortunately, the number of constraints is still equal to $|S||A|$. For special types of *factored* MDPs, it is possible to simplify the number of constraints by exploiting conditional independence properties in the transition matrix [51].

4.4.2 Reproducing Kernel Hilbert Space Methods

A Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$ is a vector space of functions on a (state) space equipped with a symmetric *kernel* function $K(.,.)$ that serves as the "representer" of evaluation of any value function:

$$V(x) = \langle V, K(x,.) \rangle_{\mathcal{H}},$$

which implies that the kernel satisfies the "reproducing property":

$$K(x,y) = \langle K(x,.), K(y,.) \rangle_{\mathcal{H}}.$$

One way to interpret the kernel function is to view it as "implicitly" encoding a large (possibly infinite) set of features $\phi(s)$, such that

$$K(x,y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}.$$

A simple "polynomial" kernel is defined as the inner product between a set of polynomial features:

$$\phi(x) = (1, x, x^2, \ldots, x^n),$$

which generates the kernel

$$K(x,y) = (1 + \langle x,y \rangle)^n.$$

To formulate value function approximation using RKHS methods, we assume that the exact value function, say V^π, can be approximated by the inner product

$$\hat{V}^\pi(s) \approx \langle w, \phi(s) \rangle_{\mathcal{H}}, \tag{4.12}$$

where $\phi(s) = K(s,.)$ is specified by the reproducing kernel associated with $\mathcal{H}$. We can treat the problem of approximating V^π as an instance of *support vector regression* (SVR) [120], where the goal is to approximate an unknown function f from samples $(x, f(x))$. The intuitive idea is to approximate f to within ϵ, where only errors outside ϵ are penalized. Also, occasional deviations above ϵ can be tolerated by a slack variable. One SVR formulation of value function approximation as a quadratic optimization problem is given below [13].

Definition 4.9. The *support vector regression* formulation of value function approximation can be formulated as minimizing the Bellman residual error $\text{BE} = \hat{V}^\pi_w - T^\pi(\hat{V}^\pi_w)$, where $\hat{V}^\pi_w = \Phi w$. In quadratic programming, regression problems can be formulated as minimizing the "norm" of the associated weight vector w in the RKHS defined by a kernel function $K(x,y) = \langle \phi(x), \phi(y) \rangle$. Its "primal form" formulation is given as

$$\begin{aligned} \min_{w,\xi} \frac{1}{2}\|w\|^2_{\mathcal{H}} &+ c\sum_{s \in \mathcal{S}}(\xi_s + \xi_s^*) \\ \text{such that} \quad \text{BE}(s) &\leq \epsilon + \xi_s \\ -\text{BE}(s) &\leq \epsilon + \xi_s^* \\ \xi_s, \xi_s^* &\geq 0, \quad \forall\ s \in \mathcal{S}, \end{aligned}$$

where $\mathcal{S}$ is a set of samples, ξ, ξ^* are "slack variables," and ϵ is a fixed "resolution" parameter.

Such quadratic programs are usually solved by using the Lagrangian dual. The key benefit of finding the dual solution is that it is expressed purely in terms of inner products of $K(x,y) = \langle \phi(x), \phi(y) \rangle$. This enables using the "kernel trick" [120].

4.5 Summary

Thus, we can see that there are a range of methods for approximately solving MDPs, all of which assume that the basis set ϕ is given explicitly. A detailed comparison of these methods is beyond the scope of this paper, but is the subject of ongoing research. We next turn to methods for automatically constructing the basis Φ.

5

Dimensionality Reduction Principles in MDPs

Before proceeding to describe specific solutions to the problem of constructing low-dimensional representations of Markov decision processes (MDPs), we outline some general characteristics of the problem in this section. There are several variants of the basis construction problem, each of which may lead to a different solution. Furthermore, there are also a set of trade-offs that need to be considered: the issue of optimizing a single versus multiple policies, as well as approximating a single decision problem versus multiple are some of the choices that need to be considered. In Sections 6 and 7, we will explore particular solutions in greater detail.

5.1 Low-Dimensional MDP Induced by a Basis

We have referred several times to the problem of basis construction as building a "low-dimensional" representation of a MDP. Here, we formalize this somewhat "intuitive" notion. In particular, it is possible to show that a least-squares approximation of the value function on a basis matrix Φ is equivalent to solving an exact "low-dimensional" MDP M_Φ [104]. This result implies that a basis Φ is not only useful in approximating value functions, but also induces a "simpler" MDP. The methods described later in Sections 6 and 7 yield abstractions that are useful in compressing any function on the state space.

Definition 5.1. Given a basis matrix Φ and policy π, the induced *approximate reward* function R^π_Φ and *approximate model* P^π_Φ are defined as:

$$\begin{aligned} P^\pi_\Phi &= (\Phi^T\Phi)^{-1}\Phi^T P^\pi \Phi, \\ R^\pi_\Phi &= (\Phi^T\Phi)^{-1}\Phi^T R^\pi. \end{aligned}$$

In essence, R^π_Φ is simply the least-squares projection of the original reward function R^π onto the column space of Φ. Similarly, P^π_Φ is the least-squares solution of the overconstrained system $\Phi P^\pi_\Phi \approx P\Phi$, where the left-hand side is the actual prediction of the features of the next state, according to the approximate model, and the right-hand side is its expected value. The following result can be easily shown.

Theorem 5.1. (Parr et al. [104]) Given a basis matrix Φ, the exact solution to the approximate policy evaluation problem defined by P^π_Φ and R^π_Φ is the same as that given by the fixed point solution of the exact policy evaluation problem defined by P^π and R^π onto the basis Φ.

Proof. We begin by reminding ourselves from Section 4 (see Equation (4.11)) that the solution to the fixed point projection problem is given as (ignoring the weighting matrix D_{ρ^π}):

$$w_{\mathrm{FP}} = \left(I - \gamma(\Phi^T\Phi)^{-1}\Phi^T P\Phi\right)^{-1}(\Phi^T\Phi)^{-1}\Phi^T R^\pi.$$

The exact solution to the "approximate" policy evaluation problem is simply

$$\begin{aligned} w &= (I - \gamma P^\pi_\Phi)^{-1} R^\pi_\Phi \\ &= (I - \gamma(\Phi^T\Phi)^{-1}\Phi^T P\Phi)^{-1}(\Phi^T\Phi)^{-1}\Phi^T R^\pi \\ &= w_{\mathrm{FP}}. \end{aligned}$$ $\square$

Thus, we can equivalently talk about approximating the exact value function by projection onto a set of basis functions Φ or equivalently, view this problem as the exact solution of a compressed MDP. This distinction is important to keep in mind in later sections. The following lemma is a straightforward consequence of the above theorem.

Lemma 5.2. Given an approximate model P^{π}_{Φ} and reward function R^{π}_{Φ} induced by a basis Φ, the exact policy evaluation problem defined by P^{π} and R^{π} is reduced from its original complexity of $O(n^3)$ by the basis Φ to $O(k^3)$.

Proof. The original policy evaluation problem requires inverting an $|S| \times |S|$ transition matrix, where $|S| = n$. The approximate policy evaluation problem reduces this to the inversion of a $k \times k$ matrix. We are ignoring, however, the complexity of mapping the original value function from the $|S|$-dimensional space to the reduced k-dimensional basis space and back. We will describe a multiscale basis construction method in Section 7 that enables a faster solution to the policy evaluation problem for some types of MDPs using a quicker method for matrix inversion. □

Although using a basis reduces the complexity of solving a MDP, what we have not yet factored into this analysis is the cost for constructing the basis, as well as the "loss" in solution quality resulting from it. These factors depend on the exact basis construction method, which we explore in subsequent sections.

5.2 Formulating the Basis Construction Problem

The problem of approximating a discrete MDP $M = (S, A, R, P)$ can be defined as constructing a basis matrix Φ of size $|S| \times k$ (or equivalently, $|S||A| \times k$), where $k \ll |S|$ (or $k \ll |S||A|$). A good basis enables the MDP to be solved "approximately" in time significantly less than it would take to solve using the default "table lookup" unit vector basis. For continuous MDPs, the basis functions are represented on a set of "sampled" states. In coming up with a more precise formulation, there are a set of trade-offs to be considered, which are described next.

5.2.1 Cost of Computation

A desirable goal is to have the cost of constructing a basis be less than the cost of solving the original MDP (in some default basis). However, in some cases, this goal may not be achievable. Such bases are still useful if the aim is to solve multiple closely related MDPs, which can amortize the cost of computing the basis. For example, in Section 7, we will describe a multiscale basis for compressing powers of a transition matrix, which provides a fast way to solve the policy evaluation problem. In this case, each successive dyadic power of the matrix is represented on a new basis. These bases depend on the transition matrix, but not on the reward function, so they can be re-used with multiple reward functions. However, reward-sensitive bases such as the Drazin or Krylov bases described in Section 7 are tailored to a specific policy (transition matrix) and reward function. These bases are more difficult to transfer, making their cost of construction a more sensitive issue.

5.2.2 Representational Complexity of a Basis

Another important consideration is the size of a basis, e.g., the number of bits required to specify a basis function or the number of nonzero coefficients of its representation on some other basis (e.g., unit vectors). Laplacian eigenvector bases (described in Section 6) are not sparse, since their support is over the whole state space. These bases are expensive to store on a general state space graph. However, we show that in special circumstances, where the state space graph decomposes as a product of simpler graphs, it is possible to represent eigenvector bases very efficiently. In Section 10, we show an application to a large multiagent MDP with 10^6 states using Laplacian eigenvector bases. We also discuss multiscale wavelet bases in Section 6, which by their very construction are significantly sparser than eigenvector bases. Finally, in Section 9, we discuss basis construction in continuous MDPs, and describe both sampling and interpolation methods that enable Laplacian eigenvector bases to be constructed in continuous MDPs. Results in Section 10 show that in several interesting continuous MDPs, excellent performance can be achieved by representing each basis function on only a few hundred sampled states.

5.2.3 Reward-Sensitive versus Reward-Independent

One primary distinction between different methods of constructing representations is whether the bases are tuned to a particular reward function, or whether they are invariant across rewards. These two choices present some interesting trade-offs: having the bases tuned to a particular reward function makes it possible to construct more effective bases that are customized to the subspace in which a particular value function lives; if the goal is indeed to compute V^π or V^*, it makes sense to customize the bases to the reward function. However, on the other hand, reward-sensitive bases are perhaps of little or no value if the goal of the decision maker is to solve multiple MDPs on the same state (action) space. Consider a robot which is tasked to navigate around a fixed environment and retrieve objects. If we define the reward function based on retrieving a particular object in a specific location, then the constructed bases will be adapted to this very specific objective. If the goal is modified, say retrieving a different object in the very same location or the same object in another location, a new set of bases have to be constructed. We will explore both reward-specific and reward-invariant bases in Sections 6 and 7.

5.2.4 Single Policy versus Multiple

Along with customizing the bases to a particular reward function, it is also possible and indeed beneficial to also customize the basis to a specific policy π in solving for its value function V^π. Each round of the policy iteration algorithm (defined earlier in Section 2.2.2) will consequently result in a specific set of basis functions tuned to a specific policy. It is possible to make the policy-specific bases additionally customized to a particular reward, or indeed, to have them invariant to the reward function. At every new round of policy iteration, a new set of bases will have to be generated. Consequently, in this formulation, the bases constructed have a fairly short life span, and multiple sets of bases have to be constructed for solving just a single MDP. On the other hand, by tuning the bases to a specific policy and reward function, it is possible to develop some theoretical guarantees on how well they can approximate V^π.

5.2.5 Incremental versus Batch Methods

One final distinction between different basis construction methods is where the representations are formed incrementally, one basis function at a time, or in a "batch" mode where multiple (or all) basis functions are constructed. The methods surveyed in the next section construct basis functions from samples drawn from a MDP. In this context, an additional distinction that can be drawn is whether each basis function is constructed incrementally from each sample. These distinctions will become clearer in later sections when we discuss concrete basis construction methods.

5.3 Basis Construction Through Adaptive State Aggregation

We now describe a simple basis construction algorithm that will illustrate some of the issues discussed above, as well as provide motivation for a more general formulation. *State aggregation* partitions the original state space S into a set of k subsets $S_1, \ldots, S_k$, where $\cup_{i=1}^{k} S_i = S$ and furthermore, $S_i \cap S_j = \emptyset, i \neq j$. We can view state aggregation as generating a special type of basis matrix Φ, where each column is an indicator function for each cluster. Each row is associated with a state and has a single nonzero entry, specifying the cluster to which the state belongs. There has been a variety of approaches to state aggregation, and a review of these methods can be found in [75]. One particular method is of interest in terms of basis construction, namely the adaptive aggregation method of using the Bellman error to cluster states [11]. Here, at each iteration, states are grouped according to the Bellman residual $T^{\pi}(V) - V$ associated with them. An abstract Markov chain is then formed using these clusters as states, and value iteration at the base level is interleaved with value iteration at the abstract level, where rewards at the abstract level are defined using the Bellman residual. Thus, the aggregation algorithm interleaves regular value iteration

$$V^{k+1} = T(V^k) = R^{\pi} + \gamma P^{\pi} V^k,$$

with a low-rank correction step using the basis matrix:

$$V^k = V^k + \Phi y.$$

Here, Φ is the basis function matrix that defines the state aggregation. y is computed by solving a reduced policy evaluation problem at the "abstract" level:

$$y = (I - \gamma P_{\Phi}^{\pi})^{-1} R_{\Phi}^{\pi}$$

where P_{Φ}^{π} and R_{Φ}^{π} are the induced transition matrix and reward function at the abstract level

$$\begin{aligned} P_{\Phi}^{\pi} &= \Phi^{\dagger} P^{\pi} \Phi, \\ R_{\Phi}^{\pi} &= \Phi^{\dagger}(T(V^k) - V^k). \end{aligned}$$

These equations are in fact exactly the same as that given in Definition 5.1, except that the reward function at the abstract level is comprised of the "residual" Bellman error projected onto the space spanned by the basis Φ. It is shown in [11] that the error after one step of value iteration and aggregation consists of two terms:

$$\begin{aligned} E_1 &= (I - \Pi)(T(V^k) - V^k), \\ E_2 &= \gamma(I - \Pi)P^{\pi}\Phi y, \end{aligned}$$

where $\Pi = \Phi\Phi^{\dagger}$ is the orthogonal projection onto the range of Φ. The first error term E_1 is controlled by adaptively aggregating the states into m groups by dividing the Bellman residual error $T(V^k) - V^k$ into m intervals. It can be shown that the term E_1 measures the difference between the Bellman error at a state and the average Bellman error of all states in its partition. The second term E_2 is not controlled by this scheme. As we will see next, the basis construction methods described in Sections 6 and 7 allow controlling this second term by constructing invariant subspaces of the transition matrix P^{π} for which $E_2 = 0$. To achieve this capability, the basis matrices cannot be restricted to partitions anymore, but are generalized to sets of (usually) orthogonal vectors. We discuss the problem of finding invariant subspaces of transition matrices next.

5.4 Invariant Subspaces: Decomposing an Operator

The approach to representation discovery described in Sections 6 and 7 can abstractly be characterized as determining the *invariant subspaces* of a vector space under some operator T, and building basis functions that span these subspaces.

Definition 5.2. A subspace χ of a vector space V is invariant under a linear mapping T if for each vector $w \in \chi$, the result $Tw \in \chi$.

A key theorem regarding invariant subspaces is worth stating and proving formally [126].

Theorem 5.3. Let χ be an invariant subspace of T, and let the columns of matrix X form a basis for χ. Then, there is a unique matrix L such that

$$TX = XL.$$

In other words, the matrix L is the representation of T on the subspace χ with respect to the basis X. It is often useful to refer to the *restriction* of an operator T on a subspace χ as $T|_{\chi}$.

Proof. The proof is straightforward (see [126]). Since χ is an invariant subspace, for any vector $x_i \in \chi$, $Tx_i \in \chi$, and consequently can be expressed as a linear combination of the columns in X. That is, $Tx_i = Xl_i$, where l_i is the unique set of coefficients. The matrix $L = [l_1, \ldots, l_n]$. □

Often, we can determine a set of invariant subspaces such that every vector in a vector space can be written as the *direct sum* of vectors from each subspace. That is, given any $v \in V$, we can write it as

$$v = w_1 + w_2 + \cdots + w_k,$$

where each $w_i \in W_i$, an invariant subspace of an operator T. We will use the following notation for direct sum decomposition of V:

$$V = W_1 \oplus W_2 \oplus \cdots \oplus W_k$$

5.4.1 Finding Invariant Subspaces of a MDP

One key idea is to find the invariant subspaces of the transition matrix P^a for each action a (or P^π for each policy π).

Definition 5.3. $\hat{P}^a$ is a representation of the invariant subspace of P^a spanned by the basis Φ if and only if

$$P^a \Phi = \Phi \hat{P}^a. \tag{5.1}$$

For example, if P^a is diagonalizable (such as a reversible random walk), then its eigenspaces form invariant subspaces.

Definition 5.4. If the transition matrix P^a is diagonalizable, then each eigenspace of the form

$$V^\lambda = \langle \{ x \,|\, P^a x = \lambda x \} \rangle, \tag{5.2}$$

is an invariant subspace of P^a where $\langle \{x, y, \ldots\} \rangle$ represents the space spanned by the enclosed vectors.

Unfortunately, transition matrices P^a are often not reversible, in which case, diagonalization leads to complex eigenvectors. One solution is to replace P^a with a related stochastic matrix, i.e., reversible and whose eigenspaces provide a suitable space to compress a MDP. We will explore such diagonalization-based methods in Section 6.

Figure 5.1 contrasts two approaches to constructing low-dimensional representations. One approach tries to find a way to compress the transition dynamics such that the MDP can be "simulated" accurately in a lower-dimensional space. Here, $\phi(s)$ is a compression function that maps a potentially high-dimensional vector to a lower-dimensional one. One approach is to find a compression function ϕ and a compressed transition function such that

$$P^a = \hat{P}^a \circ \phi. \tag{5.3}$$

In other words, compress the state first to $\phi(s) = \hat{s}$ and then construct a compressed state prediction function $\hat{P}^a$ (for each action a).

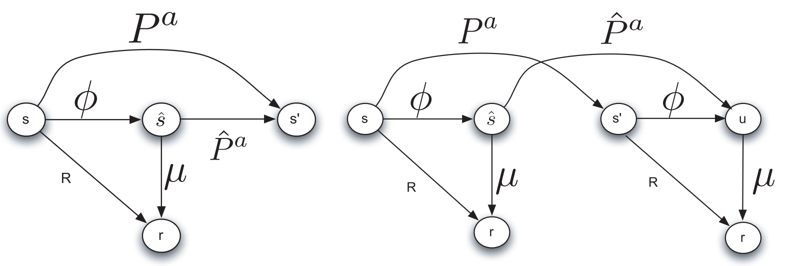


Fig. 5.1 One general principle for constructing low-dimensional representations of a MDP is to look for invariant subspaces. *Left*: One approach is to construct a compression function $\phi(s)$ of a state such that it can predict the next state accurately. *Right*: Another approach is to construct a *value-directed compression* function that suffices to predict rewards [107].

Separately, another compressed reward predictor μ is constructed to predict the true reward from the compressed state $\phi(s)$, so that

$$R = \mu \circ \phi. \tag{5.4}$$

Alternatively, the *reward-directed* (or *value-directed* [107]) approach does not attempt to construct a compression function that can predict the next state correctly, but that can only predict the rewards properly. In this latter case, it follows from the construction of the figure that

$$\phi \circ P^a = \hat{P}^a \circ \phi. \tag{5.5}$$

These two conditions are also related to the concept of reward-respecting state aggregations or *homomorphisms* used to construct reduced MDPs [48, 112]. In the special case of *linear compression*, ϕ is represented by a basis matrix Φ. In this case, the identity above shows that $\hat{P}^a$ is an invariant subspace of P^a on the space spanned by the basis Φ. We will explore methods in next two sections that do not track the one-step dynamics accurately, but that find invariant subspaces of P^π. In the context of policy iteration, it suffices to be able to find subspaces that enable evaluating each policy accurately.

6

Basis Construction: Diagonalization Methods

We begin our discussion of methods for constructing low-dimensional representations of MDPs in this section by focusing on *diagonalization* procedures. The main insight here is to look for *invariant spaces* spanned by the eigenvectors of some diagonalizable operator on the state (action) space. We begin by assuming that the Laplacian matrix $\mathbb{L}^\pi$ of a specific policy π is diagonalizable, and show how the resulting eigenvectors provide an efficient way to approximate its associated value function V^π. However, the resulting bases from diagonalizing the Laplacian matrix $\mathbb{L}^\pi$ will only be of use in approximating a specific policy π. This approach also assumes the Laplacian matrix $\mathbb{L}^\pi$ is known, or can be estimated. Consequently, to overcome these limitations, we develop an alternative approach where we substitute the random walk on a graph (or the graph Laplacian) as a diagonalizable operator, building on its attractive properties for regularizing functions on graphs.

6.1 Diagonalization of the Laplacian of a Policy

One subclass of diagonalizable transition matrices are those corresponding to *reversible* Markov chains (e.g., induced by the natural random walk on a graph). Transition matrices for general MDPs are *not* reversible, and their spectral analysis is more delicate, since it involves dealing with complex numbers. If the transition matrix P^π is diagonalizable, there is a complete set of eigenvectors $\Phi^\pi = (\phi_1^\pi, \ldots, \phi_n^\pi)$ that provides a change of basis in which the transition matrix P^π is representable as a diagonal matrix. For the sub-class of diagonalizable transition matrices represented by reversible Markov chains, the transition matrix is not only diagonalizable, but there is also an orthonormal basis. In other words, using a standard result from linear algebra, it follows that

$$\mathbb{L}^\pi = I - P^\pi = \Phi^\pi \Lambda^\pi (\Phi^\pi)^T,$$

where Λ^π is a diagonal matrix of *eigenvalues.*[1] Another way to express the above property is to write the Laplacian matrix as a sum of *projection matrices* associated with each eigenvalue:

$$\mathbb{L}^\pi = \sum_{i=1}^{n} \lambda_i^\pi \phi_i^\pi (\phi_i^\pi)^T,$$

where the eigenvectors ϕ_i^π form a complete orthogonal basis. Thus, $\| \phi_i^\pi \|_2 = 1$ and $\langle \phi_i^\pi, \phi_j^\pi \rangle = 0, i \neq j$. It readily follows that powers of $\mathbb{L}^\pi$ have the same eigenvectors, but the eigenvalues are raised to the corresponding power. Hence, $(I - P^\pi)^k \phi_i^\pi = (\lambda_i^\pi)^k \phi_i^\pi$. Since the basis matrix Φ^π spans all vectors on the state space S, the reward vector R^π can be expressed in terms of this basis as

$$R^\pi = \Phi^\pi \alpha^\pi, \tag{6.1}$$

where α^π is a vector of weights. For high powers of the transition matrix, the projection matrices corresponding to the largest eigenvalues will dominate the expansion. Combining Equation (6.1) with the

[1] $\mathbb{L}^\pi$ and P^π share the same eigenvectors, and their eigenvalues are also closely related: if λ_i is an eigenvalue of $\mathbb{L}^\pi$, then $1 - \lambda_i$ is an eigenvalue of P^π. We could obviously phrase the problem in terms of diagonalizing the transition matrix P^π directly, as has been formulated in [106]. However, in keeping with our unified theme, we describe it in terms of diagonalizing the Laplacian of the policy.

Neumann expansion in Equation (3.2), it follows that

$$\begin{aligned} V^\pi &= \sum_{i=0}^{\infty} (\gamma P^\pi)^i \Phi^\pi \alpha^\pi \\ &= \sum_{k=1}^{n} \sum_{i=0}^{\infty} \gamma^i (1 - \lambda_k^\pi)^i \phi_k^\pi \alpha_k^\pi \\ &= \sum_{k=1}^{n} \frac{1}{1 - \gamma(1 - \lambda_k^\pi)} \phi_k^\pi \alpha_k^\pi \\ &= \sum_{k=1}^{n} \beta_k \phi_k^\pi, \end{aligned}$$

using the property that $(P^\pi)^i \phi_j^\pi = (1 - \lambda_j^\pi)^i \phi_j^\pi$. Essentially, the value function V^π is represented as a linear combination of eigenvectors of the transition matrix. In order to provide the most efficient approximation, the summation can be truncated by choosing some small number $m < n$ of the eigenvectors, preferably those for whom β_k is large. Of course, since the reward function is not known, it might be difficult to pick *a priori* those eigenvectors that result in the largest coefficients. A simpler strategy instead is to focus on those eigenvectors for whom the coefficients $\frac{1}{1-\gamma(1-\lambda_k^\pi)}$ are the largest. In other words, the eigenvectors corresponding to the *smallest* eigenvalues of the Laplacian matrix $I - P^\pi$ should be selected (since the smallest eigenvalue of $\mathbb{L}^\pi$ is 0, the eigenvalues closest to 0 should be selected):

$$V^\pi \approx \sum_{k=1}^{m} \frac{1}{1 - \gamma(1 - \lambda_k^\pi)} \phi_k^\pi \alpha_k^\pi, \tag{6.2}$$

where the eigenvalues are ordered in nonincreasing order, so λ_1^π is the largest eigenvalue. If the transition matrix P^π and reward function R^π are both known, one can of course construct basis functions by diagonalizing P^π and choosing eigenvectors "out-of-order" (i.e., pick eigenvectors with the largest β_k coefficients above).

The spectral approach of diagonalizing the transition matrix is problematic for several reasons. One, the transition matrix P^π cannot be assumed to be reversible, in which case diagonalization may result in complex eigenvalues (and eigenvectors). Second, the transition matrix may be unknown. Of course, one can always use samples of the underlying MDP generated by exploration to estimate the transition matrix, but the number of samples needed may be large. Finally, in control learning, the policy keeps changing, causing one to have to re-estimate the transition matrix. For these reasons, a different approach to constructing bases is now described by diagonalizing the natural random walk on a graph induced from a MDP.

6.2 Regularization Using Graph Laplacian Operators

The graph Laplacian was introduced in Section 3. Here, we show how it can be used to regularize or smoothly interpolate functions on graphs from noisy samples. Let $G = (V,E,W)$ represent an undirected graph on $|V| = n$ nodes, where $(u,v) \in E$ is an edge from vertex u to v. Edges are all assumed to have a weight associated with them, specified by the W matrix. For now, we assume $W(u,v) = W(v,u)$, but this assumption will be relaxed later. The notation $u \sim v$ means an (undirected) edge between u and v, and the degree of u to be $d(u) = \sum_{u\sim v} w(u,v)$. D will denote the diagonal matrix defined by $D_{uu} = d(u)$, and W the matrix defined by $W_{uv} = w(u,v) = w(v,u)$.

The space of functions on a graph forms a Hilbert space, where each function $f : V \to \mathbb{R}$ under the inner product[2]:

$$\langle f,g\rangle = \sum_{v\in V} f(v)g(v).$$

The notion of a *smooth* function on a graph can now be formalized. The $\mathbb{L}^2$ norm of a function on G is defined as

$$\|f\|_2^2 = \sum_{v\in V} |f(v)|^2 d(v).$$

The gradient of a function is $\nabla f(i,j) = w(i,j)(f(i) - f(j))$ if there is an edge e connecting i to j, 0 otherwise. The smoothness of a function

[2] As discussed previously, one can also define a weighted inner product that takes into account the invariant (stationary) distribution of the Markov chain induced by a random walk on the graph.

on a graph can be measured by the *Sobolev norm* [82]:

$$\begin{aligned}\|f\|_{\mathcal{H}^2}^2 &= \|f\|_2^2 + \|\nabla f\|_2^2 \\ &= \sum_{v \in V} |f(v)|^2 d(v) + \sum_{u \sim v} |f(u) - f(v)|^2 w(u,v). \end{aligned} \tag{6.3}$$

The first term in this norm controls the size (in terms of $\mathbb{L}^2$-norm) for the function f, and the second term controls the size of the gradient. The smaller $\|f\|_{\mathcal{H}^2}$, the smoother is f. In the applications to be considered later, the functions of interest have small $\mathcal{H}^2$ norms, except at a few points, where the gradient may be large.

For simplicity, let us assume an orthonormal basis $(e_1, \ldots, e_{|V|})$ for the space $\mathbb{R}^{|V|}$. For a fixed precision ϵ, a function f can be approximated as

$$\left\| f - \sum_{i \in S(\epsilon)} \alpha_i e_i \right\| \leq \epsilon$$

with $\alpha_i = \langle f, e_i \rangle$ since the e_i's are orthonormal, and the approximation is measured in some norm, such as $\mathbb{L}^2$ or $\mathcal{H}^2$. The goal is to obtain representations in which the index set $S(\epsilon)$ in the summation is as small as possible, for a given approximation error ϵ. This hope is well founded at least when f is smooth or piecewise smooth, since in this case it should be compressible in some well chosen basis $\{e_i\}$.

The *combinatorial Laplacian* L [26] was earlier defined as $L = D - W$, where D is a diagonal matrix whose entries are the row sums of W. Alternatively, the *normalized* Laplacian $\mathcal{L} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ is often used, whose eigenvalues lie in $[0,2]$ [26].

One of the key attractive properties of the (combinatorial or normalized) Laplacian is that it is positive semi-definite. Since both the Laplacian operators, $\mathcal{L}$ and L, are also symmetric, the spectral theorem from linear algebra can be applied, yielding a discrete set of eigenvalues that are all nonnegative: $0 \leq \lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_i \leq \cdots$ and a corresponding orthonormal basis of real-valued eigenfunctions $\{\xi_i\}_{i \geq 0}$, solutions to the eigenvalue problem $\mathcal{L}\xi_i = \lambda_i \xi_i$.

The eigenfunctions of the Laplacian can be viewed as an orthonormal basis of global Fourier smooth functions that can be used for approximating any value function on a graph [26]. A striking property of these basis functions is that they capture large-scale features of a graph, and are particularly sensitive to "bottlenecks," a phenomenon widely studied in Riemannian geometry and spectral graph theory [23, 26, 44].

Observe that ξ_i satisfies $\|\nabla\xi_i\|_2^2 = \lambda_i$. In fact, the variational characterization of eigenvectors (described below) shows that ξ_i is the normalized function orthogonal to $\xi_0,\ldots,\xi_{i-1}$ with minimal $\|\nabla\xi_i\|_2$. Hence, the projection of a function f on S onto the top k eigenvectors of the Laplacian is the smoothest approximation to f, in the sense of the norm in $\mathcal{H}^2$. A potential drawback of Laplacian approximation is that it detects only global smoothness, and may poorly approximate a function which is not globally smooth but only piecewise smooth, or with different smoothness in different regions. These drawbacks are addressed in the context of analysis with diffusion wavelets described in Section 7.6 [30].

Building on the Dirichlet sum above, a standard *variational* characterization of eigenvalues and eigenvectors views them as the solution to a sequence of minimization problems. In particular, the set of eigenvalues can be defined as the solution to a series of minimization problems using the Rayleigh quotient [26]. This provides a variational characterization of eigenvalues using projections of an arbitrary function $g: V \to \mathcal{R}$ onto the subspace $\mathcal{L}g$. The quotient gives the eigenvalues and the functions satisfying orthonormality are the eigenfunctions:

$$\frac{\langle g, \mathcal{L}g\rangle}{\langle g, g\rangle} = \frac{\langle g, D^{-\frac{1}{2}} L D^{-\frac{1}{2}} g\rangle}{\langle g, g\rangle} = \frac{\sum_{u\sim v}(f(u) - f(v))^2 w_{uv}}{\sum_u f^2(u) d_u},$$

where $f \equiv D^{-\frac{1}{2}}g$. The first eigenvalue is $\lambda_0 = 0$, and is associated with the constant function $f(u) = \mathbf{1}$, which means the first eigenfunction $g_o(u) = \sqrt{D}\ \mathbf{1}$ (see, e.g., top left plot in Figure 6.1). The first eigenfunction (associated with eigenvalue 0) of the combinatorial Laplacian is the constant function $\mathbf{1}$. The second eigenfunction is the infimum over all functions $g: V \to \mathcal{R}$ that are perpendicular to $g_o(u)$, which gives us a formula to compute the first nonzero eigenvalue λ_1, namely

$$\lambda_1 = \inf_{f \perp \sqrt{D}\mathbf{1}} \frac{\sum_{u\sim v}(f(u) - f(v))^2 w_{uv}}{\sum_u f^2(u) d_u}.$$

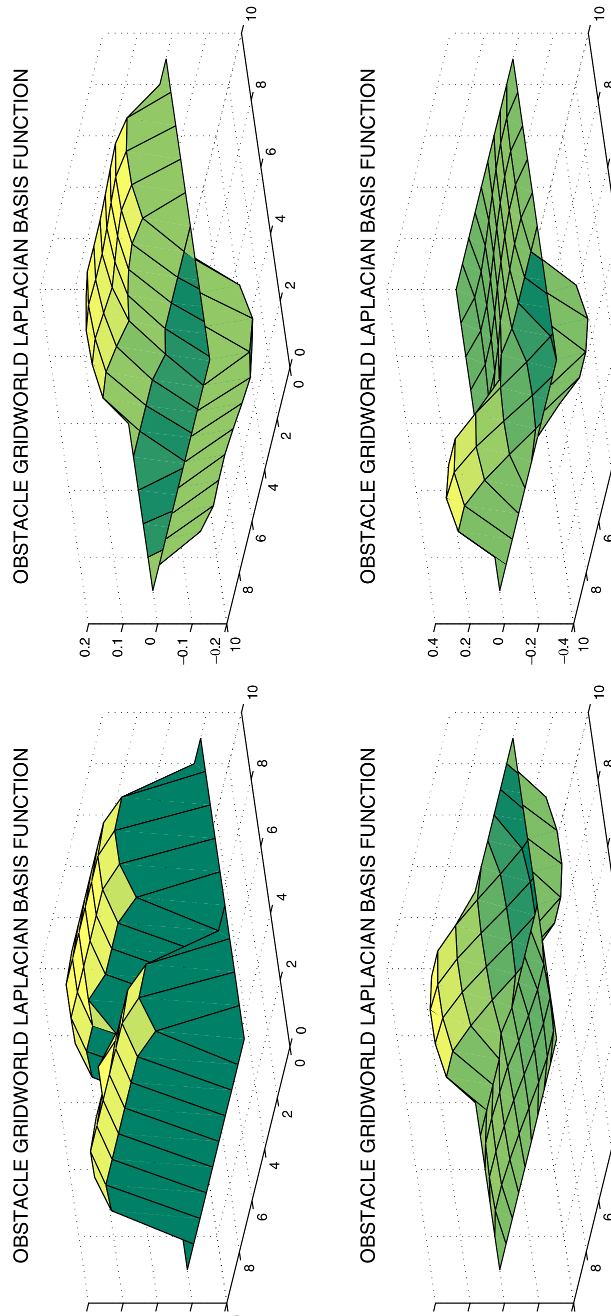


Fig. 6.1 First four eigenfunctions based on diagonalizing the normalized Laplacian operator for a "two-room" spatial environment shown earlier in Figure 1.1.

The Rayleigh quotient for higher-order basis functions is similar: each function is perpendicular to the subspace spanned by previous functions. In other words, the eigenvectors of the graph Laplacian provide a systematic organization of the space of functions on a graph that respects its topology.

6.2.1 Examples of Graph Laplacian Basis Functions

This section provides an intuitive pictorial overview of the problem of value function approximation and illustrates some types of bases produced by the methods described above. Figure 6.1 illustrates one type of basis function formed by diagonalizing a graph operator called the *normalized Laplacian* [26] over an undirected graph, where edges (s, s') are created with a unit weight if $P(s'|s,a) > 0$ or $P(s|s',a) > 0$, where s and s' are any two reachable states in the two-room MDP, and a is one of the four compass directions. Each *eigenfunction* (or eigenvector) can be viewed as a function mapping each discrete state to a real number. Notice how these eigenfunctions are highly sensitive to the geometry of the underlying state space, clearly reflecting the bottleneck connecting the two rooms.

6.2.2 Diagonalization of the Directed Graph Laplacian

In the previous section, basis functions were created by diagonalizing an undirected graph. In this section, we show that these ideas straightforwardly generalize to directed graphs using the directed graph Laplacian [27]. A weighted directed graph is defined as $G_d = (V, E_d, W)$. The major distinction between the directed and undirected graph is the nonreversibility of edges. A directed graph may have weights $w_{ij} = 0$ and $w_{ji} \neq 0$. In order to diagonalize a graph operator on a directed graph, the operator must be made self-adjoint. Since the standard random walk operator on a directed graph is not self-adjoint, it is symmetrized by using the leading eigenvector associated with the spectral radius (or largest eigenvalue) 1. This leading eigenvector is sometimes referred to as the *Perron* vector and has played a major role in web search engines.

A random walk on G_d defines a probability transition matrix $P = D^{-1}W$. The Perron-Frobenius Theorem states that if G_d is strongly connected then P has a unique left eigenvector ψ with all positive entries such that $\psi P = \rho\psi$, where ρ is the spectral radius. ρ can be set to 1 by normalizing ψ such that $\sum_i \psi_i = 1$. A more intuitive way of thinking of ψ is as the long-term steady state probability of being in any vertex over a long random walk on the graph. There is no closed-form solution for ψ; however, there are several algorithms to calculate it. The power method [49] iteratively calculates ψ starting with an initial guess for ψ using the definition $\psi P = \psi$ to determine a new estimate.

Definition 6.1. The combinatorial and normalized graph Laplacian for a directed graph $G = (V, E, W)$ is defined as

$$L_d = \Psi - \frac{\Psi P + P^T \Psi}{2}, \tag{6.4}$$

$$\mathcal{L}_d = I - \frac{\Psi^{1/2} P \Psi^{-1/2} + \Psi^{-1/2} P^T \Psi^{1/2}}{2}, \tag{6.5}$$

where Ψ is a diagonal matrix with entry $\Psi_{ii} = \psi_i$.

To find basis functions on directed state action graphs, we compute the k smoothest eigenvectors of L_d or $\mathcal{L}_d$. These eigenvectors form Φ and can be used with any of the approximation methods described in Section 4. A comparison of the directed and undirected Laplacian for solving MDPs can be found in [57]. The directed Laplacian requires a strongly connected graph, however, graphs created from an agent's experience may not have this property. In order to ensure that this property exists, a "teleporting" random walk can be used. With probability η the agent acts according to the transition matrix P and with probability $1 - \eta$ teleports to any other vertex in the graph uniformly at random.

6.3 Scaling to Large State Space Graphs

While diagonalization of Laplacian operators is a theoretically attractive framework for approximating MDPs, it can be computationally intractable to apply the framework to large discrete or continuous spaces. In this section, we describe ways of scaling this approach to large discrete spaces. We investigate several approaches, ranging from exploiting the structure of highly symmetric graphs [32, 63], to the use of sparsification and sampling methods [39] to streamline matrix computations. We first analyze structured graphs that are constructed from simpler graphs, based on the notion of a *Kronecker product* [32]. We describe a general framework for scaling basis construction to large *factored* discrete spaces using properties of product spaces, such as grids, cylinders, and tori. A crucial property of the graph Laplacian is that its embeddings are highly regular for structured graphs (see Figure 6.3). We will explain the reason for this property below, and how to exploit it to construct compact encodings of Laplacian bases. We also describe an *approximate* Kronecker decomposition that decomposes any matrix into a product of smaller matrices.

6.3.1 Product Spaces: Complex Graphs from Simple Ones

Building on the theory of graph spectra [32], we now describe a hierarchical framework for efficiently computing and compactly storing basis functions on product graphs. Many MDPs lead to *factored* representations where the space is generated as the Cartesian product of the values of variables (examples such MDPs were given in Figures 2.1 and 2.2). Consider a hypercube graph with d dimensions, where each dimension can take on k values. The size of the resulting graph is $O(k^d)$, and the size of each function on the graph is $O(k^d)$. Using the hierarchical framework presented below, the hypercube can be viewed as the *Kronecker sum* of d path or chain graphs, each of whose transition matrix is of size (in the worst case) $O(k^2)$. Now, each factored function can be stored in space $O(dk^2)$, and the cost of spectral analysis greatly reduces as well. Even greater savings can be accrued since usually only a small number of basis functions are needed relative to the size of the graph. Figure 6.2 illustrates the idea of scaling Fourier and wavelet basis functions to large product graphs.

Various compositional schemes can be defined for constructing complex graphs from simpler graphs [32]. We focus on compositions that

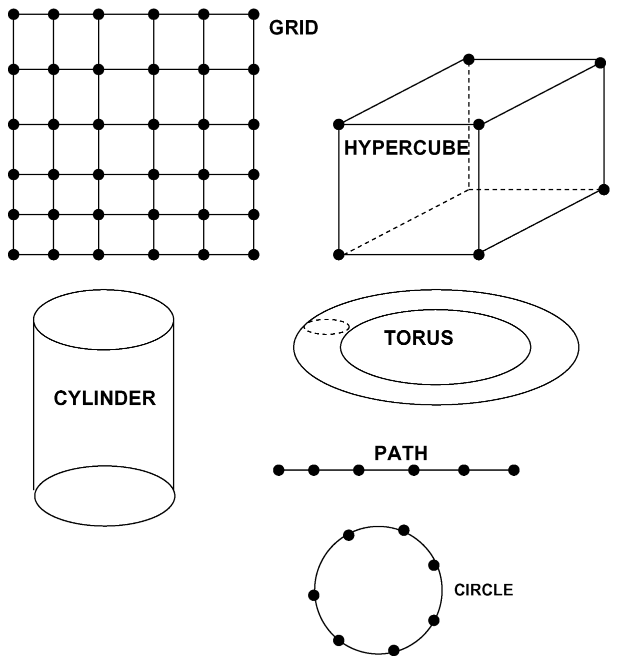


Fig. 6.2 The spectrum and eigenspace of structured state spaces, including grids, hypercubes, cylinders, and tori, can be efficiently computed from "building block" sub-graphs, such as paths and circles. This hierarchical framework greatly reduces the computational expense of computing and storing basis functions.

involve the Kronecker (or the tensor) sum of graphs. Let $G_1, \ldots, G_n$ be n undirected graphs whose corresponding vertex and edge sets are specified as $G_i = (V_i, E_i)$. The *Kronecker sum graph* $G = G_1 \oplus \cdots \oplus G_n$ has the vertex set $V = V_1 \cdots V_n$, and edge set $E(u,v) = 1$, where $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$, if and only if u_k is adjacent to v_k for some $u_k, v_k \in V_k$ and all $u_i = v_i, i \neq k$. For example, the grid graph illustrated in Figure 6.2 is the *Kronecker sum* of two path graphs; the hypercube is the Kronecker sum of three or more path graphs.

The Kronecker sum graph can also be defined using operations on the component adjacency matrices. If A_1 is a (p,q) matrix and A_2 is a (r,s) matrix, the Kronecker product matrix[3] $A = A_1 \otimes A_2$ is a (pr, qs) matrix, where $A(i,j) = A_1(i,j) * A_2$. In other words, each entry of A_1

[3] The Kronecker product of two matrices is often also referred to as the *tensor product.*

is replaced by the product of that entry with the entire A_2 matrix. The Kronecker sum of two graphs $G = G_1 \oplus G_2$ can be defined as the graph whose adjacency matrix is the Kronecker sum $A = A_1 \otimes I_2 + A_2 \otimes I_1$, where I_1 and I_2 are the identity matrices of size equal to number of rows (or columns) of A_1 and A_2, respectively. The main result that we will exploit is that the eigenvectors of the Kronecker product of two matrices can be expressed as the Kronecker products of the eigenvectors of the component matrices.

Theorem 6.1. Let A and B be full rank square matrices of size $r \times r$ and $s \times s$, respectively, whose eigenvectors and eigenvalues can be written as

$$Au_i = \lambda_i u_i, \quad 1 \leq i \leq r, \quad Bv_j = \mu_j v_j, \quad 1 \leq j \leq s.$$

Then, the eigenvalues and eigenvectors of the Kronecker product $A \otimes B$ and Kronecker sum $A \oplus B$ are given as

$$\begin{aligned}(A \otimes B)(u_i \otimes v_j) &= \lambda_i \mu_j (u_i \otimes v_j)\\ (A \oplus B)(u_i \otimes v_j) &= (A \otimes I_s + I_r \otimes B)(u_i \otimes v_j) = (\lambda_i + \mu_j)(u_i \otimes v_j).\end{aligned}$$

The proof of this theorem relies on the following identity regarding Kronecker products of matrices: $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$ for any set of matrices where the products AC and BD are well defined. We denote the set of eigenvectors of an operator $\mathcal{T}$ by the notation $X(\mathcal{T})$ and its spectrum by $\Sigma(\mathcal{T})$. A standard result that follows from the above theorem shows that the combinatorial graph Laplacian of a Kronecker sum of two graphs can be computed from the Laplacian of each subgraph. In contrast, the normalized Laplacian is not well-defined under sum, but has a well-defined semantics for the Kronecker or direct product of two graphs. The Kronecker product can also be used as a general method to approximate any matrix by factorizing it into the product of smaller matrices [137].

Theorem 6.2. If $L_1 = L(G_1)$ and $L_2 = L(G_2)$ are the combinatorial Laplacians of graphs $G_1 = (V_1, E_1, W_1)$ and $G_2 = (V_2, E_2, W_2)$, then the spectral structure of the combinatorial Laplacian $L(G)$ of the Kronecker sum of these graphs $G = G_1 \oplus G_2$ can be computed as

$$(\Sigma(L), X(L)) = \{\lambda_i + \kappa_j, l_i \otimes k_j\}, \quad 1 \leq i \leq |V_1|, \quad 1 \leq j \leq |V_2|,$$

where λ_i is the ith eigenvalue of L_1 with associated eigenvector l_i and κ_j is the jth eigenvalue of L_2 with associated eigenvector k_j.

The proof is omitted, but fairly straightforward by exploiting the property that the Laplace operator acts on a function by summing the difference of its value at a vertex with those at adjacent vertices. Figure 6.3 illustrates this theorem, showing that the eigenvectors of the combinatorial Laplacian produce a regular embedding of a grid in 2D as well as a cylinder in 3D. These figures were generated as follows. For the grid shown on the left, the eigenvectors were generated as the Kronecker product of the eigenvectors of the combinatorial Laplacian for two chains of size 10. The figure shows the embedding of the grid graph where each state was embedded in $\mathbb{R}^2$ using the second and third smallest eigenvector. For the cylinder on the right, the eigenvectors were generated as the Kronecker product of the eigenvectors of the combinatorial Laplacian for a 10 state closed chain and a 5 state open

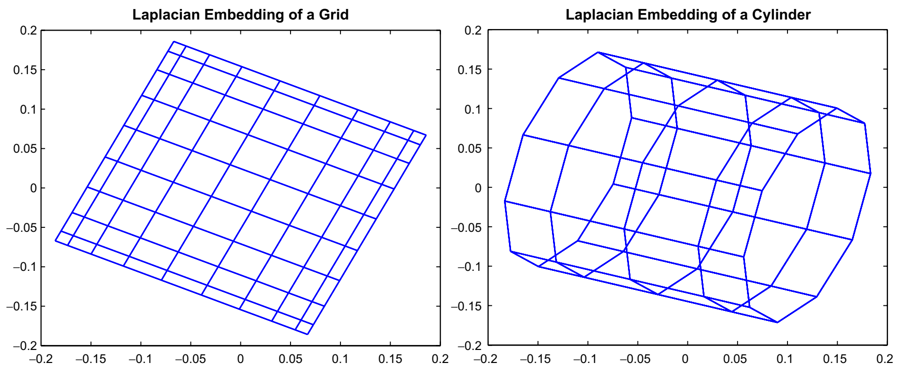


Fig. 6.3 *Left*: This figure shows an embedding in $\mathbb{R}^2$ of a 10×10 grid world environment using "low-frequency" (smoothest) eigenvectors of the combinatorial Laplacian, specifically those corresponding to the second and third smallest eigenvalues. *Right*: The embedding of a "cylinder" graph using two low-order eigenvectors ($3rd$ and $4th$) of the combinatorial Laplacian. The cylinder graph is the Kronecker sum of a closed and open chain graph.

chain. The embedding of the cylinder shown on the right was produced using the third and fourth eigenvector of the combinatorial Laplacian.

For the combinatorial Laplacian, the constant vector **1** is an eigenvector with associated eigenvalue $\lambda_0 = 0$. Since the eigenvalues of the Kronecker sum graph are the sums of the eigenvalues of the individual graphs, 0 will be an eigenvalue of the Laplacian of the sum graph as well. Furthermore, for each eigenvector v_i, the Kronecker product $v_i \otimes \mathbf{1}$ will also be an eigenvector of the sum graph. One consequence of these properties is that geometry is well preserved, so for example the combinatorial Laplacian produces well-defined embeddings of structured spaces.

6.3.2 Decomposing Large Graphs Using Approximation Methods

A variety of other approximation methods can be used to scale basis construction to large graphs, including matrix sparsification [1], low-rank approximation [46], graph partitioning [62], and Kronecker product approximation [137]. We review the latter two methods here. Kronecker product approximation [137] constructs two smaller stochastic matrices B and C whose Kronecker product $B \otimes C$ approximates a given matrix A.

Let $P_r = D^{-1}W$ denote the random walk matrix, as described in Section 3. P_r can be approximated by a Kronecker product of two smaller stochastic matrices P_a and P_b, which minimizes the Fröbenius norm of the error:

$$f(P_a, P_b) = \min_{P_a, P_b} \left(\|P_r - P_a \otimes P_b\|_F \right).$$

Pitsianis [137] describes a separable LSM to decompose stochastic matrices, but one problem with this approach is that the decomposed matrices, although stochastic, are not guaranteed to be diagonalizable. This problem was addressed by Johns et al. [58], who applied this approach for learning to solve MDPs. To ensure the diagonalizability of the decomposed matrices, Johns et al. [58] incorporated an additional step using the Metropolis Hastings algorithm [15] to approximate the smaller matrices P_a and P_b by *reversible matrices* P_a^r and P_b^r.

Then, the eigenvectors of the original random walk matrix P_r can be approximated as the Kronecker product of the eigenvectors of the factorized smaller reversible matrices P_a^r and P_b^r (since the smaller matrices are reversible, they can also be symmetrized using the normalized Laplacian, which makes the numerical task of computing their eigenvectors much simpler). Using this approach, Johns et al. [58] were able to reduce the size of the random walk weight matrices by a significant amount compared to the full matrix. For example, for the well-known Acrobot control problem illustrated in Figure 2.5, the original basis matrix was compressed by a factor of 36:1, without significant loss in solution quality. An important point to emphasize is that the full basis matrix never needs to be stored or computed in constructing the vertex embeddings from the smaller matrices. The factorization can be carried out recursively as well, leading to a further reduction in the size of the basis matrices.

6.3.3 Graph Partitioning

A general divide-and-conquer strategy is to decompose the original graph into subgraphs, and then compute local basis functions on each subgraph. This strategy can be used on any graph, however, unlike the methods described above, few theoretical guarantees can be provided except in special circumstances. A number of graph partitioning methods are available, including spectral methods that use the low-order eigenvectors of the Laplacian to decompose graphs [91], as well as hybrid methods that combine spectral analysis with other techniques.

Graph partitioning is a well-studied topic, and there are a large variety of nonspectral methods as well. METIS [62] is a fast graph partitioning algorithm that can decompose even very large graphs on the order of 10^6 vertices. METIS uses a *multiscale* approach to graph partitioning, where the original graph is "coarsened" by collapsing vertices (and their associated edges) to produce a series of smaller graphs, which are successively partitioned followed by uncoarsening steps mapping the partitions found back to the lower-level graphs.

7

Basis Construction: Dilation Methods

In this section, we introduce another general strategy for constructing low-dimensional representations of MDPs by using Laplacian operators to *dilate* a reward function or initial basis. We describe two dilation procedures. The simplest dilation algorithm is to use the *Krylov space* associated with the Laplacian $\mathbb{L}^{\pi}$ of a policy. We contrast this approach with another approach motivated by the Laurent series decomposition, where the dilation of rewards is accomplished using the Drazin inverse of the Laplacian $\mathbb{L}^{\pi D}$ introduced in Section 3.1. In both cases, the basis is generated from the powers $(\mathbb{T})^t R^{\pi}$ (where $\mathbb{T} = \mathbb{L}^{\pi}$ or $\mathbb{L}^{\pi D}$). The Krylov space has long been a staple of methods for efficiently solving systems of linear equations of the form $Ax = b$. These procedures are highly turned to a specific policy and reward function, and some attractive theoretical properties can be shown that follow readily from the geometry of Krylov spaces. Unfortunately, the resulting set of basis functions are limited in this regard as well. Hence, we describe a more general framework for constructing a series of multiscale basis functions, motivated by the framework of wavelet analysis in Euclidean spaces. A specific algorithm for multiscale basis construction on graphs using dilation of graph Laplacian operators is presented.

7.1 Krylov Spaces

The principle of dilation is easiest to explain using the well-known Krylov space of methods [118]. In Section 7.6, we will explain a more sophisticated approach using wavelets. Unlike the wavelet approach, which uses just the dyadic powers, Krylov spaces compute all powers of a matrix. Also, unlike the wavelet paradigm, which constructs a multiresolution analysis, keeping the bases at every scale separate, the Krylov bases do not admit a clear multiscale interpretation. Krylov bases have found much use in eigensolvers, and have been extensively studied in the context of solving systems of linear equations of the form $Ax = b$ [41, 49]. Note that reward-specific bases entail some compromises: if there are multiple tasks that are to be performed on the same state (action) space (e.g., navigating to different locations), bases learned from one goal cannot be easily transferred to another goal location. However, by constructing bases sensitive to a reward function, one gains the advantage of being able to prove (under idealized conditions) certain guarantees of reconstruction accuracy.

Definition 7.1. The jth *Krylov* subspace $\mathcal{K}_j$ generated by an operator T and a function f is the space spanned by the vectors:

$$\mathcal{K}_j = \{f, Tf, T^2 f, \ldots, T^{j-1} f\}. \tag{7.1}$$

Clearly, $\mathcal{K}_j \subset \mathbb{C}^{\mathbb{N}}$. Note that $\mathcal{K}_1 \subseteq \mathcal{K}_2 \subseteq \cdots$, such that for some $m, \mathcal{K}_m = \mathcal{K}_{m+1} = \mathcal{K}$. Thus, $\mathcal{K}$ is the T-invariant Krylov space generated by T and f. When T is completely diagonalizable, the projections of f onto the eigenspaces of T form a basis for the Krylov space $\mathcal{K}$ [88].

Theorem 7.1. If a matrix $T \in \mathbb{C}^n \times \mathbb{C}^n$ is diagonalizable, and has n distinct eigenvalues, the nontrivial projections of f onto the eigenspaces of T form a basis for the Krylov space generated by T and f.

Note that Krylov spaces can be used in situations when matrices are not diagonalizable. In the next two sections, we study two approaches to constructing Krylov bases: one uses the Drazin inverse of the Laplacian $\mathbb{L}^D$, and the other uses the Laplacian $\mathbb{L}$ directly.

7.2 Reward Dilation Using Laplacian Operators

The most direct method for constructing bases from Krylov spaces is to dilate the reward function R^π using the Laplacian $\mathbb{L}^\pi$ associated with a policy π.[1]

Definition 7.2. Given the Laplacian matrix $\mathbb{L}^\pi$ and reward function R^π, the jth *Krylov* subspace $\mathcal{K}_j$ is defined as the space spanned by the vectors:

$$\mathcal{K}^\pi{}_j = \{R^\pi, \mathbb{L}^\pi R^\pi, (\mathbb{L}^\pi)^2 R^\pi, \ldots, (\mathbb{L}^\pi)^{j-1} R^\pi\}. \tag{7.2}$$

Note that $\mathcal{K}^\pi{}_1 \subseteq \mathcal{K}^\pi{}_2 \subseteq \cdots$, such that for some $m, \mathcal{K}^\pi{}_m = \mathcal{K}^\pi{}_{m+1} = \mathcal{K}^\pi$. Thus, $\mathcal{K}^\pi$ is the $\mathbb{L}^\pi$-invariant Krylov space generated by $\mathbb{L}^\pi$ and R^π.

7.2.1 Bellman Error Bases

A closely related idea to reward-sensitive Krylov bases is the concept of the Bellman error basis function (BEBF), studied by several researchers, whose theoretical and empirical properties have been recently investigated by Parr et al. [103, 104]. The intuitive idea is to select the next basis function in the direction of the error in approximating a given value function V^π. More formally, let w^π_Φ be the set of weights associated with the weighted least-squares projection defined in Equation (4.1). Thus, the current approximation to the value function is then

$$V^\pi_\Phi = \Phi w^\pi_\Phi.$$

The Bellman error is then defined as

$$\mathrm{BE}(\Phi) = T(V^\pi_\Phi) - V^\pi_\Phi = R^\pi + \gamma P^\pi \Phi w^\pi_\Phi - \Phi w^\pi_\Phi.$$

The next basis that is added is proportional to the Bellman error, that is

$$\phi_{k+1} = \mathrm{BE}(\Phi).$$

[1] Once again, we couch this description in terms of dilation using the $\mathbb{L}^\pi = I - P^\pi$ operator, rather than P^π. The resulting bases are the same in either case.

Since the error is orthogonal to the subspace spanned by the current set of bases, one can easily show that the resulting set of bases forms (when suitably normalized) an orthonormal set, and hence constitutes a complete basis. If the initial basis $\phi_1 = R^\pi$, then it can be proved that the Bellman error basis is identical to the Krylov basis [104]. Furthermore, it can be shown that if the Bellman operator T is known exactly, then this procedure will drive down the error in approximating V^π at least as fast as value iteration (with a unit vector basis). Of course, these guarantees hold under idealized conditions when T is known exactly (i.e., P^π and R^π are given). If T has to be approximated from samples, then the performance of BEBFs is of course subject to sampling error and noise in the estimation process.

7.2.2 Examples of Krylov Bases of the Laplacian

Figure 7.1 illustrates the first four bases associated with a chain MDP of 50 states. This problem was previously studied by Lagoudakis and Parr [69]. The two actions (go left, or go right) succeed with probability 0.9. When the actions fail, they result in movement in the opposite direction with probability 0.1. The two ends of the chain are treated as "dead ends." The basis vectors shown are the result of applying a QR decomposition to orthogonalize the Krylov vectors. Notice that the first basis function is essentially the reward function (inverted and scaled to have length 1). One problem with these bases should be apparent: due to their limited support, approximation will be quite slow. We will describe below how Drazin bases overcomes this problem, resulting in their converging significantly faster in the chain domain in the experiments shown in Section 8.

Figure 7.2 illustrates some sample Krylov basis functions for the two room MDP, where P^π is the optimal policy transition matrix and R^π is the associated reward function. A sharp contrast can be made between the Drazin bases and the Krylov bases by comparing Figure 7.4 with Figure 7.2. Since Krylov bases remain largely dependent on the reward function initially, their support is largely localized if the reward function is so (in this case, the reward function was a delta function at one state). Consequently, in approximation of MDPs, this localization

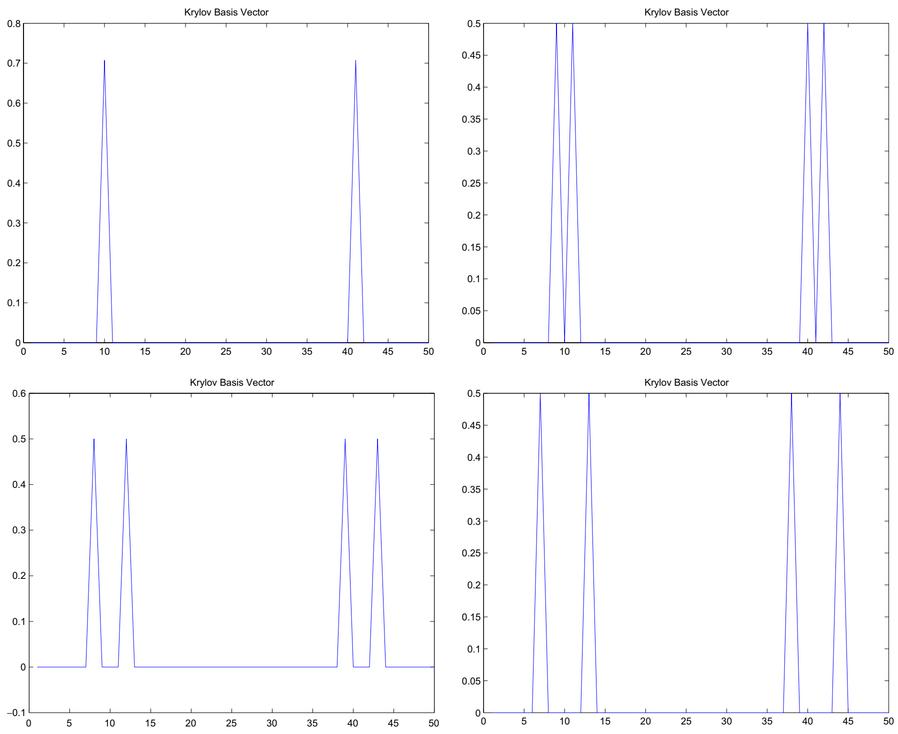


Fig. 7.1 The first four Krylov bases for a chain MDP with 50 states. Rewards of +1 are given in states 10 and 41. Actions are to go left or right, each succeeds with probability 0.9. The Laplacian associated with the optimal policy is used to dilate the reward function.

can make the progress rather slow, as the experiments in Section 8 will show. Drazin bases converge much more quickly as the initial terms are defined over the whole state space. Of course, there are many other issues that need to be considered as well, such as the computational complexity, in determining the "right" choice of a basis.

7.3 Reward Dilation Using Drazin Inverse of Laplacian

Recall from Section 3.1 that the discounted value function V^π associated with a policy π can be written in a Laurent series involving powers of the Drazin inverse of the Laplacian $\mathbb{L}^\pi$ associated with π. Thus, a natural set of basis vectors with which to approximate V^π are those generated by dilating R^π by the powers of $(\mathbb{L}^\pi)^D$.

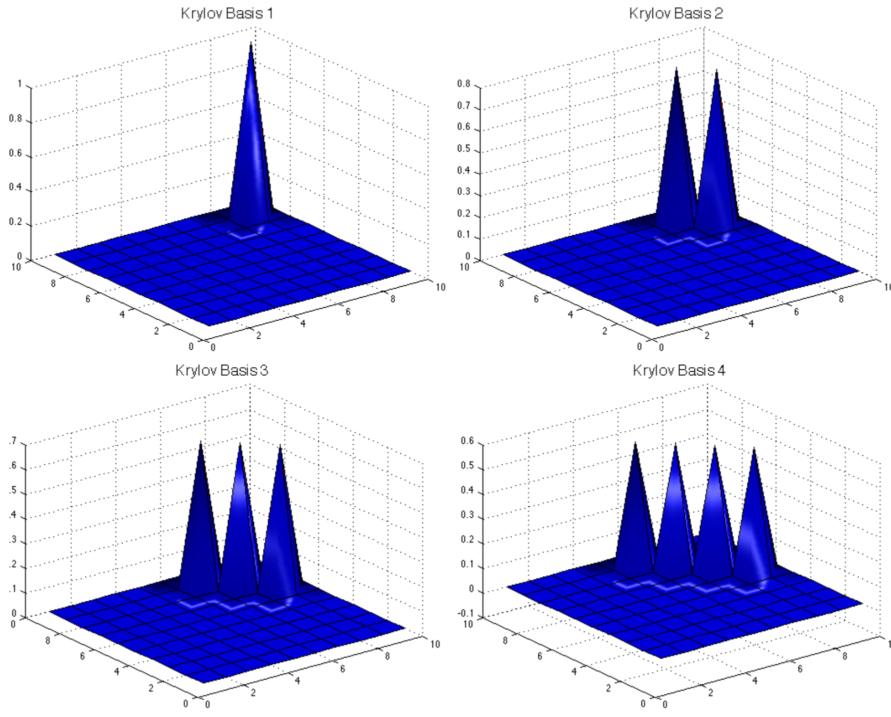


Fig. 7.2 The first four Krylov bases for the “two-room” environment based on dilation of the reward function using the Laplacian of a policy.

Definition 7.3. The *Drazin space* associated with a policy π and reward function R^π is defined as the space spanned by the set of Drazin vectors:

$$\mathbb{D}_m^\pi = \{P^* R^\pi, (\mathbb{L}^\pi)^D R^\pi, ((\mathbb{L}^\pi)^D)^2 R^\pi, \ldots, ((\mathbb{L}^\pi)^D)^{m-1} R^\pi\}. \qquad (7.3)$$

Note that $\mathcal{D}^\pi{}_1 \subseteq \mathcal{D}^\pi{}_2 \subseteq \cdots$, such that for some $m, \mathcal{D}^\pi{}_m = \mathcal{D}^\pi{}_{m+1} = \mathcal{D}^\pi$. Thus, $\mathcal{D}^\pi$ is the $(\mathbb{L}^\pi)^D$-invariant Krylov space generated by $\mathbb{L}^\pi$ and R^π.

The first basis vector is the average-reward or gain $g^\pi = P^* R^\pi$ of policy π. The second basis vector is the product of $\mathbb{L}^D R^\pi$. Subsequent basis vectors are defined by the dilation of the reward function R^π by higher powers of the Drazin inverse $\mathbb{L}^D$ used in the Laurent series expansion of V^π. The following property is worth noting.

Theorem 7.2. The product $(P^\pi)^*(\mathbb{L}^\pi)^D = 0$.

Proof. Expanding the product, we get

$$(P^\pi)^*(\mathbb{L}^\pi)^D = (P^\pi)^*\left((I - P^\pi + (P^\pi)^*)^{-1} - (P^\pi)^*\right) \tag{7.4}$$

$$= (P^\pi)^*(I - P^\pi + (P^\pi)^*)^{-1} - (P^\pi)^* \tag{7.5}$$

$$= (P^\pi)^* - (P^\pi)^* = 0 \tag{7.6}$$

□

A variety of algorithms for computing the Drazin inverse of a matrix are reviewed in [20], some of which were described in Section 3.3.4. As a prelude to describing the multiscale diffusion wavelet framework, we will show a computational trick called the Schultz expansion that can be used to compute both the Krylov bases as well as the Drazin inverse using dyadic powers (powers of two) of an operator. As we will see in Section 8, Drazin bases can give excellent results in simple problems. However, the computational complexity of computing Drazin bases needs to be taken into account. Parallel implementations for computing pseudo-inverses in general, and the Drazin inverse in particular, have been developed. These methods enabled generalized inverses can be computed in $O(\log^2 n)$, assuming there are sufficient processors to compute matrix multiplication in $O(\log n)$ [25, 141]. This parallel algorithm uses an iterative method to compute the Drazin inverse. In Section 7.4, we will show how this iterative expansion can be effectively computed using the Schultz expansion.

7.3.1 Examples of Drazin Bases of the Laplacian

Figure 7.3 illustrates the first four Drazin bases associated with a chain MDP of 50 states. The first basis is the average reward $\rho = P^* R^\pi$. Notice, how unlike the Krylov bases, the bases combine both global and local support. As a consequence, we will see in Section 8 that convergence is much more rapid. The second basis function is essentially the reward function (scaled to unit length), which will be important

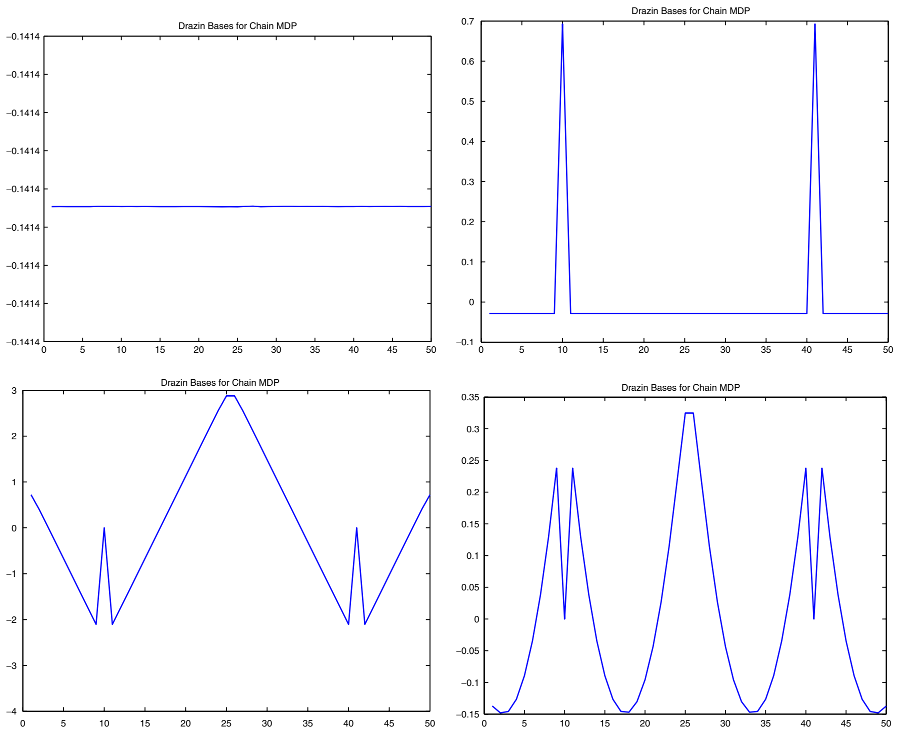


Fig. 7.3 The first four Drazin bases for a chain MDP with 50 states. Rewards of +1 are given in states 10 and 41. Actions are to go left or right, each succeeds with probability 0.9. The Laplacian associated with the optimal policy is used to dilate the reward function.

in reducing the overall error as will be shown in the control learning experiments.

Figure 7.4 illustrates some sample Drazin basis functions for the two-room MDP, where P^π is the optimal policy transition matrix and R^π is the associated reward function. The first basis is the gain $g^\pi = P^* R^\pi$. The second term is the bias h^π. The subsequent bases are orthogonal to the first two vectors.

7.4 Schultz Expansion for Drazin and Krylov Bases

As a prelude to introducing the general framework of diffusion wavelets, we motivate one of the key ideas by showing how both Drazin and Krylov bases can be significantly accelerated by a computational trick

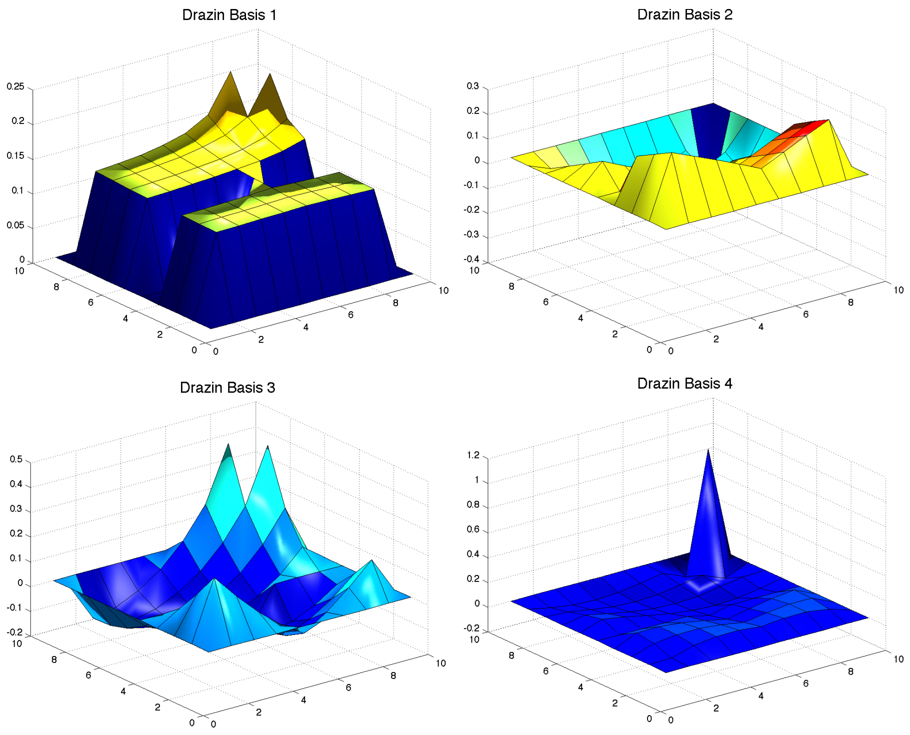


Fig. 7.4 Drazin bases for the "two-room" environment based on dilation of the reward function using the Drazin inverse of the Laplacian of a policy.

called the *Schultz expansion* [14]. Both these methods for basis construction fundamentally involve expanding the reward function R^π in terms of a sequence of powers of an operator T. Let us abstractly consider applying the inverse $(I - T)^{-1}$ (e.g., $T = P^\pi$) on a function f (e.g., $f = R^\pi$).

$$(I - T)^{-1} f = (I + T + T^2 + \cdots) f. \tag{7.7}$$

Let us consider finitely summable sequences derived from the above infinite series as follows

$$\begin{aligned}(I + T) f &= f + Tf \\ (I + T)(I + T^2) f &= f + Tf + T^2 f + T^3 f \\ (I + T)(I + T^2)(I + T^4) f &= f + Tf + T^2 f + T^3 + \cdots + T^7 f.\end{aligned}$$

Denoting the sum of the first 2^k terms in this series by S_k, we have the recurrence relation:

$$S_{k+1} = \sum_{i=1}^{2^k-1} T^i = (I + T^{2^k})S_k. \tag{7.8}$$

Thus, we can rewrite the Neumann series in Equation (7.7) using the Schultz expansion:

$$(I-T)^{-1}f = (I + T + T^2 + \cdots)f = \prod_{k=0}^{\infty}(I + T^{2^k})f. \tag{7.9}$$

In Section 7.6, we describe a general method for computing *compressed* representations of the dyadic powers T^{2^k} of an operator. The Schultz expansion provides an effective way of storing both the Drazin and Krylov bases. For example, we can rewrite the Neumann series for the discounted value function as follows

$$V^\pi = (I - \gamma P^\pi)^{-1}R^\pi = \prod_{k=0}^{\infty}(I + (\gamma P^\pi)^{2^k})R^\pi. \tag{7.10}$$

7.5 Multiscale Iterative Method to Compute Drazin Bases

To compute the Drazin bases using the Schultz expansion, we will use an iterative method called *Successive Matrix Squaring* (SMS) that was developed for efficient parallel implementation of general inverses of matrices [141, 25]. Note that from the definition of Drazin inverse in Section 3.3.3, we note that if X is a Drazin inverse of A, then it must satisfy the following properties:

$$XAX = X, \quad XA = AX, \quad A^{k+1}X = A^k, \tag{7.11}$$

where k is the index of A (for Laplacian matrices derived from P^π, the index $k = 1$). An iterative method can be developed starting with the last identity, as follows

$$\begin{aligned} X_{k+1} &= X - \beta(A^{k+1}X_k - A^k) \\ &= (I - \beta A^{k+1})X_k + \beta A^k = SX_{k+1} + Q, \end{aligned} \tag{7.12}$$

where $S = (I - \beta A^{k+1})$ and $Q = \beta A^k$. In our case, to compute the Drazin inverse of the Laplacian $\mathbb{L}^{\pi D} = (I - P^\pi)^D$, we have

$$S^\pi = (I - \beta A^2) = (I - \beta(I - P^\pi)^2), \quad Q^\pi = \beta(I - P^\pi). \tag{7.13}$$

Define the $2n \times 2n$ matrix T as follows

$$T = \begin{pmatrix} S & Q \\ 0 & I \end{pmatrix}. \tag{7.14}$$

The SMS algorithm consists of the following iteration:

$$T_m = (T_{m-1})^2, \tag{7.15}$$

whereby the mth term T_m can be written as follows

$$T_m = \begin{pmatrix} S^m & \sum_{i=0}^{2^m-1} S^i Q \\ 0 & I \end{pmatrix}. \tag{7.16}$$

The mth approximation to the Drazin inverse is thus given by the sub-matrix that appears on the top right, namely

$$A^D(m) = \sum_{i=0}^{2^m-1} S^i Q. \tag{7.17}$$

Now, we can once again apply the Schultz expansion here, and rewrite this sum as follows

$$A^D(m) = \prod_{k=0}^{m} (I + S^{2^k}) Q. \tag{7.18}$$

Thus, we see that both Drazin and Krylov bases can be effectively computed by using dyadic powers of an operator. We now turn to describe a general framework for compressing such powers using the principles of wavelets [86].

7.6 Dilation and Multiscale Analysis

We now introduce a broad framework for multiscale analysis using ideas inspired by *wavelet analysis* [86]. The concept of dilation is at the heart of wavelet analysis, and was first developed in Euclidean spaces. On the real line, a function is dilated by "stretching" the function in time, e.g., $f(x) \to f(2x)$. What does it mean to dilate a function on a general state space, such as a graph or manifold? Dilation in such cases means the application of an operator, such as the Laplacian matrix $\mathbb{L}^{\pi}$ of a specific policy, or the natural random walk Laplacian $L_r = I - P_r = I - D^{-1}W$ on a graph. Given a function f on a state space S, dilation corresponds to applying powers of the operator T to a function f, giving rise to Tf, $T^2 f, \ldots$. Any given function (e.g., the delta function δ_x mapping state x to 1 and taking the value 0 elsewhere) will be "diffused" by applying powers of an operator to it, where the rate of dilation depends on the operator.

Figure 7.5 illustrates the rationale for multiscale analysis of operators. If P represents one step of applying the operator, by the Markov property, P^t represents t steps. For an initial condition δ_x (i.e, where x is the starting state), $P^t \delta_x(y)$ represents the probability of being at y

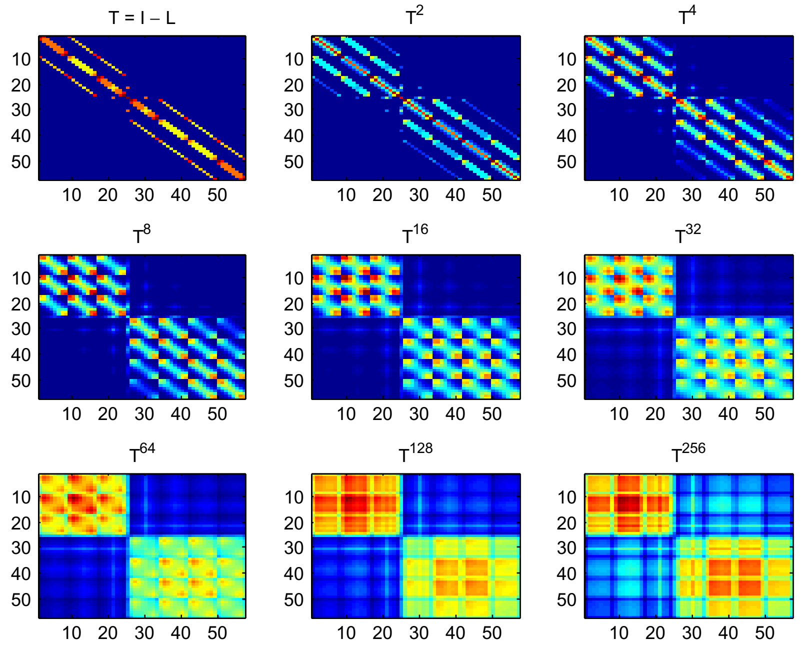


Fig. 7.5 Powers of the diffusion operator $T = I - \mathcal{L}$ for the 100 state "two-room" MDP. Note that for higher powers, the operator matrix is progressively less sparse.

at time t, conditioned on starting in state x. The matrix P encodes local similarities between points, and the matrix P^t is diffusing, or integrating, this local information for t steps to larger and larger neighborhoods of each point. The process $\{P^t\}_{t\geq 0}$ can be analyzed at different time scales. If P is diagonalizable, for very large times, the random walk can be analyzed through its top eigenvectors. However, eigenvectors are global, and not suited for analyzing small- and medium-scale behavior. On the other hand, many interesting features of the data and of functions on the data can be expected to exist at small and medium time scales. The task of analyzing P^t for all times and locations seems to require either large time in order to compute all powers of P (which is computationally expensive since, even if P is sparse, its powers are not), and/or large space to store those powers. However, there is redundancy in time and space in the family $\{P^t(x,y)\}_{t\geq 0;x,y\in X}$. There is a spatial redundancy: if x and y are close and t is large (depending on the distance between x and y), $P^t(x,\cdot)$ is very similar to $P^t(y,\cdot)$. Secondly, there is a redundancy across time scales: using the Markov property, $P^{2t}(x,y)$ can be computed from knowledge of $P^t(x,\cdot)$ and $P^t(\cdot,y)$. These two properties can be exploited resulting in an efficient multiscale algorithm called *diffusion wavelets* [30].

7.7 Diffusion Wavelets

Diffusion wavelets enable a fast multiscale analysis of functions on a manifold or graph, generalizing wavelet analysis and associated signal processing techniques (such as compression or denoising) to functions on manifolds and graphs. They allow the efficient and accurate computation of high powers of a Markov chain P on the manifold or graph, including direct computation of the Green's function, or the inverse of the Laplacian $L^{-1} = (I - P)^{-1}$ (on the complement of the kernel of the Laplacian).

A multi-resolution decomposition of the functions on a graph is a family of nested subspaces $V_0 \supseteq V_1 \supseteq \cdots \supseteq V_j \supseteq \cdots$ spanned by orthogonal bases of diffusion scaling functions Φ_j. If T^t can be interpreted as an operator on functions on the graph, then V_j is defined as the numerical range, up to precision ε, of $T^{2^{j+1}-1}$, and the scaling functions are smooth bump functions with some oscillations, at scale roughly 2^{j+1} (measured with respect to geodesic distance). The orthogonal complement of V_{j+1} into V_j is called W_j, and is spanned by a family of orthogonal diffusion wavelets Ψ_j, which are smooth localized oscillatory functions at the same scale.

Here and in the rest of this section, the notation $[L]_{B_1}^{B_2}$ indicates the matrix representing the linear operator L with respect to the basis B_1 in the domain and B_2 in the range. A set of vectors B_1 represented on a basis B_2 will be written in matrix form $[B_1]_{B_2}$, where the rows of $[B_1]_{B_2}$ are the coordinates of the vectors B_1 in the coordinate system defined by B_2.

The input to the algorithm is a stochastic matrix P, derived from the natural random walk $P_r = D^{-1}W$ on a weighted graph (G, E, W), or the transition matrix P^π of a policy π. The powers of P are used to "dilate," or "diffuse" functions on the state space, and then define an associated coarse-graining of the graph. Observe that in many cases of interest P is a sparse matrix. We usually *normalize* P and consider $T = \Pi P \Pi^{-1}$ where Π is the asymptotic distribution of P, which is assumed to exist, is unique and can be chosen to be a strictly positive distribution by the Perron-Fröbenius Theorem. If P is reversible, $\Pi = D^{\frac{1}{2}}$, and T is symmetric. In the other cases, if T is not symmetric, in what follows any statement regarding eigenvectors should be disregarded.

T is assumed to be a sparse matrix, and that the numerical rank of the powers of T decays rapidly with the power. A diffusion wavelet tree consists of orthogonal diffusion scaling functions Φ_j that are smooth bump functions, with some oscillations, at scale roughly 2^j (measured with respect to geodesic distance), and orthogonal wavelets Ψ_j that are smooth localized oscillatory functions at the same scale. The scaling functions Φ_j span a subspace V_j, with the property that $V_{j+1} \subseteq V_j$, and the span of Ψ_j, W_j, is the orthogonal complement of V_j into V_{j+1}. This is achieved by using the dyadic powers T^{2^j} as "dilations" (see, e.g., Figure 7.5), to create smoother and wider (always in a geodesic sense) "bump" functions (e.g., which represent densities for the symmetrized random walk after 2^j steps), and orthogonalizing and down-sampling appropriately to transform sets of "bumps" into orthonormal scaling functions.

```
$\{\Phi_j\}_{j=0}^{J}, \{\Psi_j\}_{j=0}^{J-1}, \{[T^{2^j}]_{\Phi_j}^{\Phi_j}\}_{j=1}^{J} \leftarrow$
DiffusionWaveletTree $([T]_{\Phi_0}^{\Phi_0}, \Phi_0, J, \mathrm{SpQR}, \varepsilon)$
// **Input**:
// $[T]_{\Phi_0}^{\Phi_0}$ : a diffusion operator, written on the orthonormal basis $\Phi_0$
// $\Phi_0$ : an orthonormal basis which $\varepsilon$-spans $V_0$
// $J$ : number of levels
// SpQR : function to compute a sparse $QR$ decomposition.
// $\varepsilon$: precision

// **Output**:
// The orthonormal bases of scaling functions $\Phi_j$, wavelets $\Psi_j$, and
// compressed representation of $T^{2^j}$ on $\Phi_j$ for $j$ in the requested
range.

**for** $j = 0$ **to** $J - 1$ **do**

        $[\Phi_{j+1}]_{\Phi_j}$ , $[T^{2^j}]_{\Phi_j}^{\Phi_{j+1}} \leftarrow$SpQR$([T^{2^j}]_{\Phi_j}^{\Phi_j}, \varepsilon)$
        $T_{j+1} := [T^{2^{j+1}}]_{\Phi_{j+1}}^{\Phi_{j+1}} \leftarrow ([T^{2^j}]_{\Phi_j}^{\Phi_{j+1}}[\Phi_{j+1}]_{\Phi_j})^2$
        $[\Psi_j]_{\Phi_j} \leftarrow$ SpQR$(I_{\langle\Phi_j\rangle} - [\Phi_{j+1}]_{\Phi_j}[\Phi_{j+1}]_{\Phi_j}^T, \varepsilon)$

**end**
```

Fig. 7.6 Pseudo-code for construction of a Diffusion Wavelet Tree.

The multiscale construction is now briefly described, and further details can be found in the original paper [30]. The algorithm is summarized in Figure 7.6. T is initially represented on the basis $\Phi_0 = \{\delta_k\}_{k\in G}$; the columns of T are now interpreted as the set of functions $\tilde{\Phi}_1 = \{T\delta_k\}_{k\in G}$ on G. A local multiscale orthogonalization procedure is used to carefully orthonormalize these columns to get a basis $\Phi_1 = \{\varphi_{1,k}\}_{k\in G_1}$ (G_1 is *defined* as this index set), written with respect to the basis Φ_0, for the range of T up to precision ε. This information is stored in the sparse matrix $[\Phi_1]_{\Phi_0}$. This yields a subspace denoted by V_1. Essentially Φ_1 is a basis for the subspace V_1 which is ε close to the range of T, and with basis elements that are well-localized. Moreover, the elements of Φ_1 are coarser than the elements of Φ_0, since they are the result of applying the "dilation" T once. Obviously $|G_1| \leq |G|$, but this inequality may already be strict since the numerical range of T may be approximated, within the specified precision ε, by a subspace of smaller dimension. The sparse matrix $[T]_{\Phi_0}^{\Phi_1}$ is a representation of an ε-approximation of T with respect to Φ_0 in the domain and Φ_1 in the range. T can also be represented in the basis Φ_1: with the notation above this is the matrix $[T]_{\Phi_1}^{\Phi_1}$. The next power $[T^2]_{\Phi_1}^{\Phi_1} = [\Phi_1]_{\Phi_0}[T^2]_{\Phi_0}^{\Phi_0}[\Phi_1]_{\Phi_0}^T$ is computed. If T is self-adjoint, this is equal to $[T]_{\Phi_0}^{\Phi_1}([T]_{\Phi_0}^{\Phi_1})^T$, which has the advantage that numerical symmetry is forced upon $[T^2]_{\Phi_1}^{\Phi_1}$. In the general (nonsymmetric) case, $[T^2]_{\Phi_1}^{\Phi_1} = ([T^2]_{\Phi_0}^{\Phi_1}[\Phi_1]_{\Phi_0})^2$.

The columns of $[T^2]_{\Phi_1}^{\Phi_1}$ are $\tilde{\Phi}_2 = \{[T^2]_{\Phi_1}^{\Phi_1}\delta_k\}_{k\in G_1}$. These are functions $\{T^2\varphi_{1,k}\}_{k\in G_1}$, up to the precision ε. Once again, a local orthonormalization procedure is applied to this set of functions, obtaining an orthonormal basis $\Phi_2 = \{\varphi_{2,k}\}_{k\in G_2}$ for the range of T_1^2 (up to precision ε), and also for the range of T_0^3 (up to precision 2ε). Observe that Φ_2 is naturally written with respect to the basis Φ_1, and hence encoded in the matrix $[\Phi_2]_{\Phi_1}$. Moreover, depending on the decay of the spectrum of T, $|G_2|$ is in general a fraction of $|G_1|$. The matrix $[T^2]_{\Phi_1}^{\Phi_2}$ is then of size $|G_2| \times |G_1|$, and the matrix $[T^4]_{\Phi_2}^{\Phi_2} = [T^2]_{\Phi_1}^{\Phi_2}([T^2]_{\Phi_1}^{\Phi_2})^T$, a representation of T^4 acting on Φ_2, is of size $|G_2| \times |G_2|$.

After j iterations in this fashion, a representation of T^{2^j} onto a basis $\Phi_j = \{\varphi_{j,k}\}_{k\in G_j}$ is obtained, encoded in a matrix $T_j := [T^{2^j}]_{\Phi_j}^{\Phi_j}$. The orthonormal basis Φ_j is represented with respect to Φ_{j-1}, and encoded in the matrix $[\Phi_j]_{\Phi_{j-1}}$. Let $\tilde{\Phi}_j = T_j\Phi_j$, the next dyadic power of T on Φ_{j+1} can be represented on the range of T^{2^j}. Depending on the decay of the spectrum of T, $|G_j| << |G|$, in fact in the ideal situation the spectrum of T decays fast enough so that there exists $\gamma < 1$ such that $|G_j| < \gamma|G_{j-1}| < \cdots < \gamma^j|G|$. This corresponds to downsampling the set of columns of dyadic powers of T, thought of as vectors in $\mathbb{L}^2(G)$. The hypothesis that the rank of powers of T decreases guarantees that down-sampling will result in coarser and coarser lattices in this space of columns.

While Φ_j is naturally identified with the set of Dirac δ-functions on G_j,these functions on the "compressed" (or "downsampled") graph G_j can be extended to the whole initial graph G by writing

$$\begin{aligned}[\Phi_j]_{\Phi_0} &= [\Phi_j]_{\Phi_{j-1}}[\Phi_{j-1}]_{\Phi_0} \\ &= \cdots = [\Phi_j]_{\Phi_{j-1}}[\Phi_{j-1}]_{\Phi_{j-2}}\cdots[\Phi_1]_{\Phi_0}[\Phi_0]_{\Phi_0}. \end{aligned} \qquad (7.19)$$

Since every function in Φ_0 is defined on G, so is every function in Φ_j. Hence, any function on the compressed space G_j can be extended naturally to the whole G. In particular, one can compute low-frequency eigenfunctions on G_j in compressed form, and then extend them to the whole G. The elements in Φ_j are at scale $T^{2^{j+1}-1}$, and are much coarser and "smoother," than the initial elements in Φ_0, which is how they can be represented in compressed form. The projection of a function onto the subspace spanned by Φ_j will be by definition an approximation to that function at that particular scale. An example of the use of diffusion wavelets to compress powers of a diffusion operator is shown in Figure 7.7.

There is an associated fast scaling function transform: suppose f is given on G and $\langle f, \varphi_{j,k}\rangle$ needs to be computed for all scales j and corresponding "translations" k. Being given f means $(\langle f, \varphi_{0,k}\rangle)_{k\in G}$ is given. Then, $(\langle f, \varphi_{1,k}\rangle)_{k\in G_1} = [\Phi_1]_{\Phi_0}(\langle f, \varphi_{0,k}\rangle)_{k\in G}$ can be computed, and so on for all scales. The sparser the matrices $[\Phi_j]_{\Phi_{j-1}}$ (and $[T]_{\Phi_j}^{\Phi_j}$),

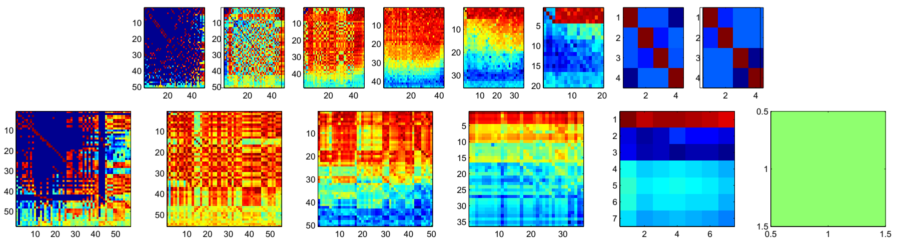


Fig. 7.7 Compressed representation of powers of the transition matrix $P^\pi = I - \mathbb{L}^\pi$ generated by the diffusion wavelet algorithm for the 50 state chain MDP (*Top*) and the two-room MDP (*Bottom*). From left to right, each row shows successive dyadic powers of the operator. The optimal policy is used in both plots. Notice that higher dyadic powers are represented by progressively smaller sized-matrices. In the two-room domain, the $2^6 = 64th$ dyadic power is effectively of size 1, a significant compression from its original size. Entries are displayed in $\log_{10}$ scale.

the faster this computation. This generalizes the classical scaling function transform. If the orthogonal projector onto V_j is denoted by Q_j, wavelet bases for the spaces W_j can be built analogously by factorizing $I_{V_j} - Q_{j+1}Q_{j+1}^T$, which is the orthogonal projection on the complement of V_{j+1} into V_j. The spaces can be further split to obtain wavelet packets [19]. The wavelets can be considered as high-pass filters, in the sense that they capture the detail lost from going from V_j to V_{j+1}, and also in the sense that their expansion in terms of eigenfunctions of the Laplacian essentially only involves eigenfunctions corresponding to eigenvalues in $[\varepsilon^{-2^j-1}, \varepsilon^{-2^{j+1}-1}]$. In particular, their Sobolev norm, or smoothness, is controlled.

In the same way, any power of T can be applied efficiently to a function f. Also, the Green's function $(I-T)^{-1}$ can be applied efficiently to any function, since it can be represented as product of dyadic powers of T, each of which can be applied efficiently. Simultaneously, the powers of the operator T and the space X itself are being compressed, at essentially the optimal "rate" at each scale, as dictated by the portion of the spectrum of the powers of T which is above the precision ε.

Observe that each point in G_j can be considered as a "local aggregation" of points in G_{j-1}, which is completely dictated by the action of the operator T on functions on G: the operator itself is dictating the geometry with respect to which it should be analyzed, compressed or applied to any vector.

Diffusion wavelets allow computing $T^{2^k} f$ for any fixed f. This is nontrivial because while the matrix representation of T is sparse, large powers of it are not, as Figure 7.5 illustrated, and the computation $T^{2^k} f = T \cdot T \cdots (T(Tf)) \cdots)$ involves 2^k matrix-vector products. As a notable consequence, this yields a fast algorithm for computing the Green's function, or fundamental matrix, associated with the Markov process T, via the Schultz expansion [76]:

$$(I-T)^{-1} f = \sum_{k \geq 0} T^k f = \prod_{k \geq 0} (I + T^{2^k}) f.$$

In a similar way one can compute $(I-P)^{-1}$. For large classes of Markov chains, this computation can be performed quickly in a direct (as opposed to iterative) fashion. This is remarkable since in general the matrix $(I - T)^{-1}$ is full and just writing down the entries would take time $\mathcal{O}(n^2)$. It is the multiscale compression scheme that allows efficiently representing $(I - T)^{-1}$ in compressed form, taking advantage of the smoothness of the entries of the matrix.

7.7.1 Examples of Scaling Function Bases

Figure 7.8 shows examples of *scaling function* bases constructed by the diffusion wavelet algorithm [30]. The figure shows scaling functions from a 6 level diffusion wavelet tree. At the lowest level (top left plot in the figure), the initial bases are just the unit vector bases. At subsequent

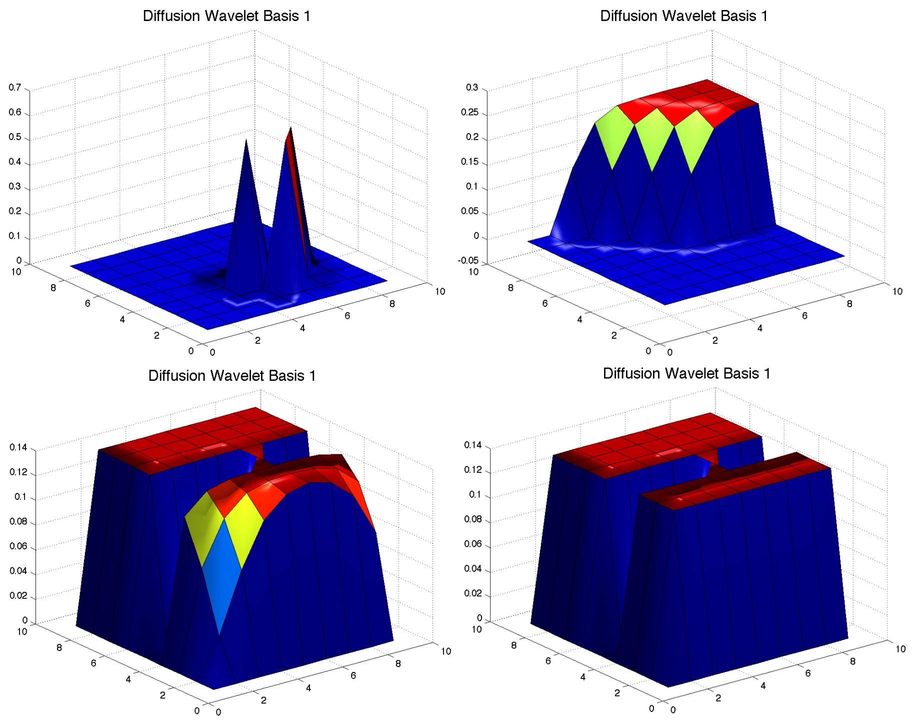


Fig. 7.8 Diffusion wavelet scaling functions based on dilating the optimal policy $P^\pi = I - \mathbb{L}^\pi$ in the two-room MDP. Reading clockwise, the bases displayed are at levels 1, 3, 4 and 5. At each level, one selected basis function is displayed. Notice how the higher level bases have larger support, it is important to note that these displays are generated by projecting the bases onto the original state space. In the diffusion wavelet tree, the bases at higher levels are stored in compressed form as described in the text.

levels, the unit vector bases are dilated using the optimal policy, and the resulting dilated vectors are then orthogonalized to construct the scaling functions. Notice how the scaling functions get coarser at each succeeding level, till at the top most level (bottom right plot), they look like the eigenfunctions shown in Figure 6.1. These bases are stored in a compressed form, where at each level, the set of basis functions is represented with respect to the bases at the lower level.

7.7.2 Laplacian Eigenfunction versus Diffusion Wavelet Bases

Figure 7.9 provides an illustrative example showing where diffusion wavelet bases excel, and where eigenfunctions of the Laplacian do poorly. The MDP is an 800 state two-room domain. The top left panel in Figure 7.9 shows a highly nonlinear "delta" function, which is significantly better approximated by the diffusion wavelet bases

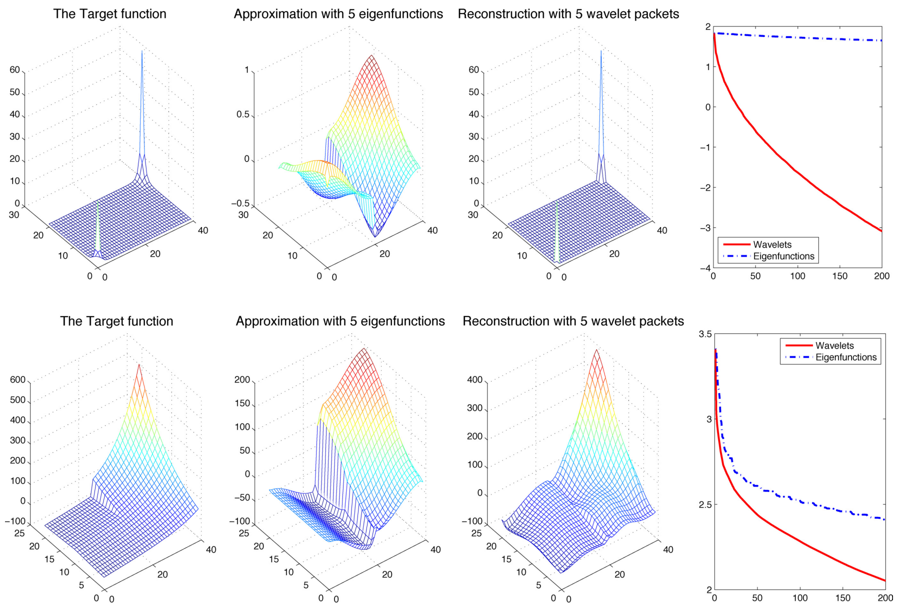


Fig. 7.9 *Left column*: Target functions. *Middle two columns*: Approximations produced by 5 diffusion wavelet bases and Laplacian eigenfunctions. *Right column*: Least-squares approximation error (log scale) using up to 200 basis functions (*Bottom curve*: Diffusion wavelets; *Top curve*: Laplacian eigenfunctions).

(second panel, top) as compared to the Laplacian (eigenfunction) basis (third panel, top). The bottom panel shows that the difference between eigenfunctions and wavelet bases is much less pronounced for smooth functions. The last column plots the error in reconstruction, measured on a log scale. The performance of Laplacian eigenfunctions is highly sensitive here to the smoothness of the target function, a finding which is highly consistent with known properties of Fourier bases in Euclidean spaces [86].

8

Model-Based Representation Policy Iteration

We now bring together the basis construction methods described in Sections 6 and 7 with the approximation methods described in Section 4 in an integrated framework for jointly learning representation and control in MDPs. This framework is referred to as *representation policy iteration* (RPI) since the underlying control learning framework can be thought of as iterating over policies, which is supplemented with algorithms that iterate over representations. A number of variants of RPI can be designed: we focus here primarily on model-based methods in discrete MDPs. Model-free methods that are better suited for continuous MDPs are discussed in the following sections. Our goal in this section is to compare bases that result from dilations of Laplacian operators from a known model and reward function. We primarily investigate two types of bases: *Krylov bases* generated from orthogonalizing the space of vectors resulting from dilating the reward function by powers of the Laplacian associated with a policy; and *Drazin bases* that are generated by expanding the discounted value function in the Laurent series expansion of powers of the Drazin inverse of the Laplacian of a policy.

8.1 Representation Policy Iteration: Drazin and Krylov Bases

Figure 8.1 specifies a more detailed algorithmic view of an overall framework called *Representation Policy Iteration* (RPI) [78] for jointly learning representation and control in MDPs. This particular variant of RPI uses a transition model and reward function to construct a low-dimensional representation of a MDP M by dilating the reward function using the Laplacian operator $\mathcal{L}^\pi = I - P^\pi$ or its Drazin inverse. Once the orthogonal basis matrix Φ is found, a low-dimensional Markov reward process is constructed and solved, whose dimension is $k \ll |S|$. The compressed solution is projected back to the original space, and a better policy is found. Note that the policy iteration step uses the uncompressed model $P^a_{ss'}$ for simplicity. In large MDPs, such model-based methods would exploit factored representations of transition matrices, such as a dynamic Bayes net [51]. It is also possible to reformulate this model-based algorithm using action-value functions $Q(x,a)$, similar to model-based LSPI [69]. The process continues until there is no change in the policy from one run to the next. Note that this constraint may be too severe: a more relaxed constraint is to terminate the algorithm when successive iterations produce compressed value functions V^π_Φ within some desired ϵ distance.

8.2 Representation Policy Iteration: Diffusion Wavelets

Figure 8.2 presents a variant of RPI using diffusion wavelets to evaluate the current policy. As described in Section 7, diffusion wavelets can be used to do a fast inversion of the policy evaluation equation

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi.$$

The Schultz expansion technique is used as described in Section 7.4 to compress the dyadic powers of P^π.

8.3 Experimental Results

We begin our experimental analysis of model-based RPI by focusing on the approximate policy evaluation step, whereby the original MDP M

```
Model-Based RPI $(\mathrm{M}, \pi, k)$:

// $M = (S, A_s, P, R)$: Input (discrete) MDP
// $\pi$: Initial policy
// **Flag** : Convergence condition for policy iteration
// $k$: Number of basis functions to use

    • **Repeat**

                    **Representation Construction Phase**

        – Define Laplacian operator $\mathbb{L}^\pi = I - P^\pi$.
        – Construct an $|S| \times k$ basis matrix $\Phi$ by orthogonalizing
          the vectors:
            ∗ **Krylov basis**:
                $\mathcal{K}_k^\pi = \{R^\pi, \mathbb{L}^\pi R^\pi, (\mathbb{L}^\pi)^2 R^\pi, \ldots, (\mathbb{L}^\pi)^{k-1} R^\pi\}$
            ∗ **Drazin basis**:
                $\mathbb{D}_k^\pi = \{(P^\pi)^* R^\pi, (\mathbb{L}^\pi)^D R^\pi, ((\mathbb{L}^\pi)^D)^2 R^\pi, \ldots, ((\mathbb{L}^\pi)^D)^{k-1} R^\pi\}$
        – Form the Markov reward process $M_\phi = (P_\phi^\pi, R_\Phi^\pi)$:
                $P_\Phi^\pi = \Phi^T P^\pi \Phi$
                $R_\Phi^\pi = \Phi^T R^\pi$

                    **Policy Evaluation Phase**

        – Find compressed solution: $(I - \gamma P_\Phi^\pi) w_\Phi = R_\Phi^\pi$.
        – Project solution back to original state space: $V_\Phi^\pi = \Phi w_\Phi$

                    **Policy Improvement Phase**

        – Find the "greedy" policy $\pi'$ associated with $V_\Phi^\pi$:
                $\pi'(s) \in \underset{a}{\operatorname{argmax}} \left( \sum_{s'} P_{ss'}^a \left( R_{ss'}^a + \gamma V_\Phi^\pi(s') \right) \right)$
        – If $\pi' \neq \pi$ set $\pi \leftarrow \pi'$, set **Flag** to false, return to Step 2
        – Else set **Flag** to true.

    • Until **Flag**
    • Return $\pi$.
```

Fig. 8.1 This figure shows a model-based algorithm for jointly learning representation and control in discrete MDPs.

```
DWT-Based RPI ($M, k, \pi, \epsilon$):

// $M = (S, A_s, P, R)$: Input (discrete) MDP
// $\pi$: Initial policy
// **Flag** : Convergence condition for policy iteration
// $k$: Number of levels for diffusion wavelet tree construction
// $\epsilon$: Desired resolution for fast inversion
// $\Phi_0$: Initial basis (e.g., unit vectors).
// **SpQR**: Sparse QR decomposition routine

    • **Repeat**

              **Representation Construction Phase**

    • Build a diffusion wavelet tree as described in Section 7.6.

              $\mathcal{T}_\pi = DWT(P^\pi, \Phi_0, k, \mathbf{SpQR}, \epsilon)$

              **Policy Evaluation Phase**

    • Compute a fast direct inverse using the Schultz expansion as
      described in Section 7.4

              $\hat{V}^\pi = \text{FastDirectInverse}(I - \gamma P^\pi, R^\pi)$

              **Policy Improvement Phase**

    • Find the "greedy" policy $\pi'$ associated with $\hat{V}^\pi_{\mathcal{T}}$:

              $\pi'(s) \in \underset{a}{\text{argmax}} \left( \sum_{s'} P^a_{ss'} \left( R^a_{ss'} + \gamma \hat{V}^\pi \right) \right)$

    • If $\pi' \neq \pi$ set $\pi \leftarrow \pi'$, set **Flag** to false
    • Else set **Flag** to true.
    • **Until Flag**

Return $\pi$.
```

Fig. 8.2 This figure shows a variant of RPI using fast inversion with diffusion wavelets.

is compressed using bases derived by dilating the reward function R^π of a policy π. This analysis is valuable since it tells us how each approach performs given exact knowledge of the transition matrix P^π and reward function R^π. We will find that Drazin bases perform surprisingly well, and outperform Laplacian eigenfunctions and Krylov bases.

8.3.1 Approximate Policy Evaluation

Figure 8.3 compares Drazin bases with Krylov and proto-value function (PVF) bases on a simple 20 state closed chain MDP. Here, the reward was set at 10 for reaching state 1. The discount factor was set at $\gamma = 0.9$. The policy π evaluated was a random walk, so the probability of moving to either neighbor was 0.5 uniformly. The plots in the figure represent the following error terms, following the style of evaluation proposed by Parr et al. [104].

(1) *Reward error*: The reward error measures the difference between the actual reward R^π and the approximated reward R^π_Φ (see Definition 5.1), plotted in the figure using the L_2-norm of the vector.

(2) *Weighted feature error*: The weighted feature error represents the product of P^π_Φ and w_Φ, where w_Φ was defined as

$$w_\Phi = (I - \gamma P^\pi_\Phi)^{-1} r_\Phi.$$

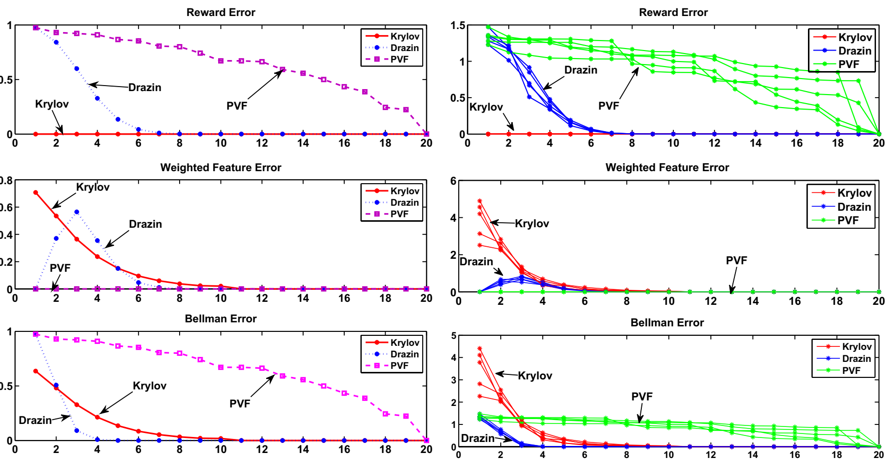


Fig. 8.3 Comparison of Drazin bases versus Krylov bases and PVFs (eigenfunctions of the combinatorial graph Laplacian) on a 20 state closed chain MDP. The reward for reaching state 1 was set at +10 for the plots on the left; the right plots show 5 separate runs with randomly generated rewards (uniform between 0 and 1). The policy evaluated in all cases was the random walk on the undirected chain graph.

(3) *Bellman error*: The Bellman error is the linear sum of the reward error and the feature error.

The plots reveal interesting differences between these three types of basis functions. Proto-value functions (PVFs) or the eigenfunctions of the combinatorial graph Laplacian (computed on the chain's adjacency matrix) have a high reward error, since the reward function is a delta function that is poorly approximated by the PVFs. In this case, PVFs are essentially the Fourier basis on the discrete circle (cosines), and this result is a well-known limitation of Fourier bases [86]. However, PVFs have zero feature error since the bases are made up of eigenvectors of the combinatorial Laplacian. These are the same as the eigenvectors of the transition matrix P^{π} in this case (the natural random walk on the graph), as was shown earlier in Section 3. The Bellman error remains large until the end when all 20 bases are used, since the reward error is large.[1]

Note the reward error for Krylov bases is by definition 0 as R^{π} is a basis vector itself. The feature error plot tells an interesting story: as Krylov vectors are added, the feature error goes down to 0 showing where an invariant subspace was produced. The Bellman error goes down to 0 much more quickly than with PVFs. Measured by the Bellman error, Drazin bases have the best performance overall at this task. The Bellman error goes down quickly to 0 using just 5 bases, twice as fast as the Krylov bases and three times more quickly than PVFs. The reward error initially is high since R^{π} is not in the basis (recall the first term is $P^{\pi}R^{\pi}$, or the gain of the policy g^{π}).

The plot on the left in Figure 8.4 shows the 50 state chain MDP studied previously in [69, 83]. The optimal policy is evaluated. Rewards are given at states 10 and 41 only. Drazin bases once again are clearly far superior to the others. The plots on the right in Figure 8.4 compare the Drazin bases with the Krylov bases on the two-room gridworld MDP. The value function for the optimal policy was shown earlier in

[1] It is important to view these comparative results using the Bellman error with some caution. Ultimately, in control learning, the goal is to learn an approximately optimal policy, which can be achieved even in the presence of large Bellman errors. In Section 10, we will show that PVFs perform excellently in both large discrete MDPs as well as continuous MDPs.

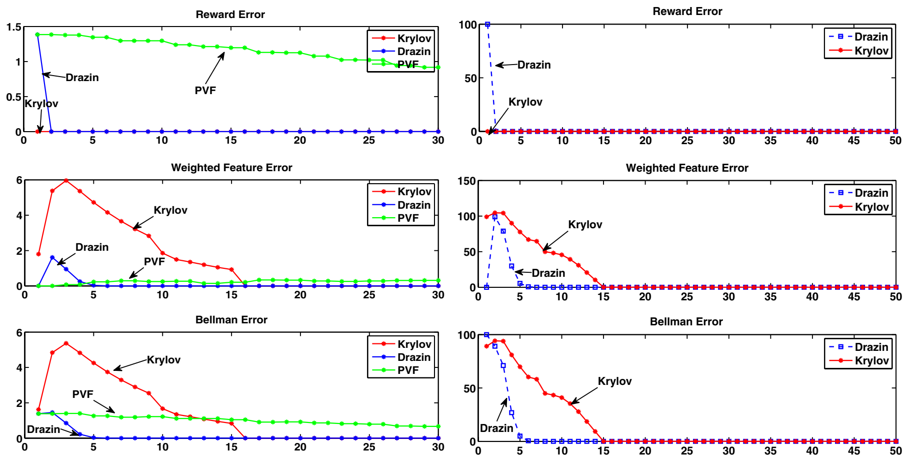
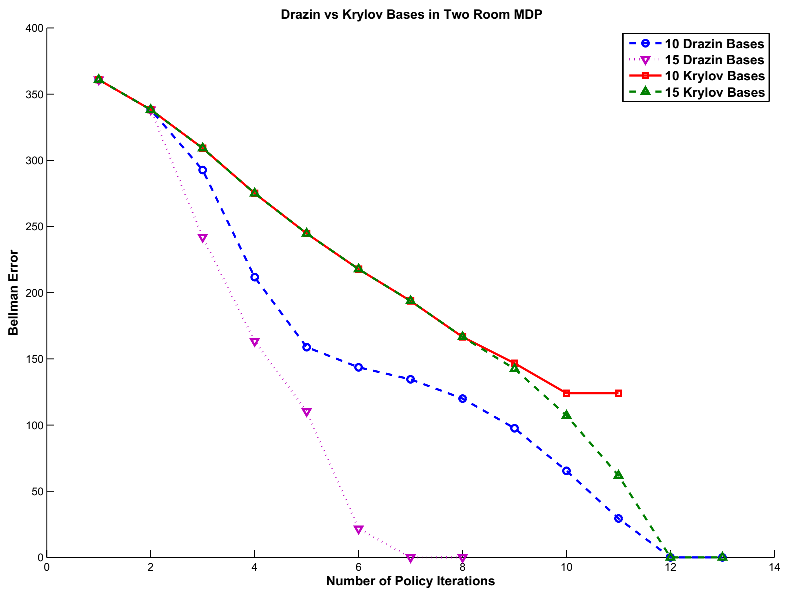


Fig. 8.4 *Left*: Comparison of Drazin and Krylov bases with PVFs on a 50 state chain MDP with rewards of +1 at states 10 and 41 (the settings are identical to [69]). The optimal policy was evaluated. *Right*: Comparison of Drazin bases versus Krylov bases in a 100 state two-room MDP, whose value function was shown earlier in Figure 3.2. Both bases were evaluated on the optimal policy. The reward was set at +100 for reaching a corner goal state.

Figure 3.2. This MDP has 100 states. The agent is rewarded by 100 for reaching a corner goal state. Compass navigation actions are stochastic, succeeding with probability 0.9. Drazin bases are again convincingly superior to Krylov bases in this problem.

8.3.2 Representation and Control Learning

Now we turn to evaluate the model-based RPI method, where both the representation and control were learned simultaneously. We use the two-room MDP once again. Figure 8.5 compares the performance of Drazin versus Krylov bases. Each curve shows the reduction in error in the approximated value function as a function of the number of iterations in policy iteration, given a specific number of bases. Drazin bases clearly perform better, achieving 0 error at a much earlier point, with just 10 bases. Krylov bases generate a large error given 10 bases, and need 15 bases to achieve 0 error. Using a weighted L_2-norm, Drazin bases converge, whereas Krylov bases generate a much larger error and do not converge (the convergence condition was set to 0.01 for the



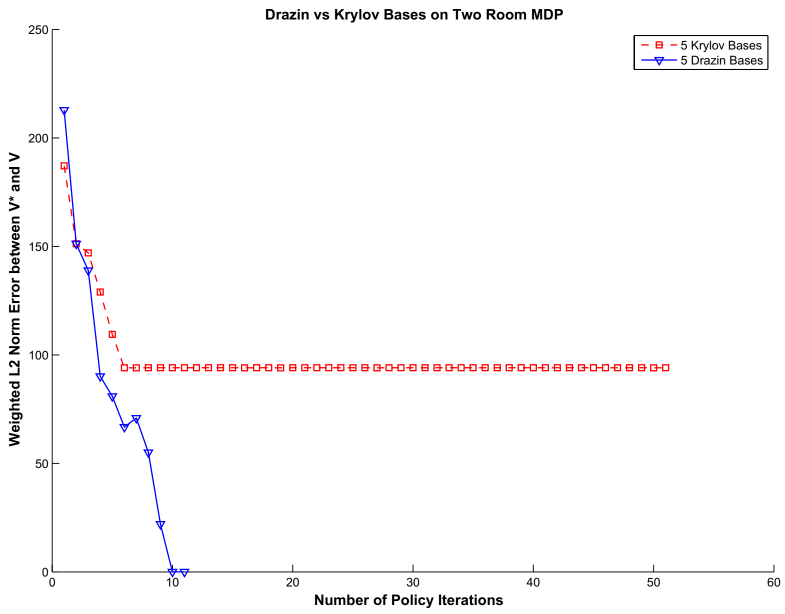


Fig. 8.5 Experimental comparison of model-based RPI using Drazin versus Krylov bases in the two-room domain using the model-based RPI algorithm described in Figure 8.1. *Top*: Error was measured by comparing the L_2-norm of the approximated value function with respect to the optimal value function. *Bottom*: A weighted L_2-norm was used with respect to the invariant distribution ρ^{π} of the policy being evaluated at each iteration.

weighted L_2-norm). The weighted least-squares projection was computed using D_{ρ^π} as defined in Equation (4.11). This result is not surprising given the local support of the initial Krylov bases, versus the more global support of the Drazin bases.

8.3.3 Comparison of Drazin versus Krylov Bases

Although Drazin bases performed better than Krylov bases, they are more expensive to generate as they require computing the Drazin inverse (or the long-term limiting matrix P^*). Computing the Drazin inverse or P^* for each policy involves significant computational resources. However, there are iterative parallelizable algorithms for computing Drazin inverses [141] described in Section 7, which could partially alleviate the computational burden. Also, in the context of policy iteration, if the policy and the associated transition matrix change locally (e.g., a few states), then it is possible to reuse the previous Drazin inverse and modify it based on the local changes. Specifically, if the transition matrix P^π changes locally, then the Drazin inverse of the new policy can be computed more quickly, i.e., $\approx O(n^2)$, than having to start from scratch (which is an $O(n^3)$ computation for a MDP with $|S| = n$ states). Additionally, a variety of aggregation methods have been developed that enable computing P^* in ergodic MDPs efficiently by decomposing P into a set of *censored* Markov chains [70].

8.3.4 RPI with Diffusion Wavelets

Finally, we turn to evaluating the RPI method using the diffusion wavelet algorithm as a means of doing policy evaluation. Figure 8.6 illustrates the approach on a larger 21×21 two-room MDP. In this procedure, the diffusion wavelet tree is constructed directly from the transition matrix P^π of the policy currently being evaluated. The compressed representation of the dyadic powers of the transition matrix of the final learned optimal policy is displayed. Additional comparisons on larger problems of DWT-based policy iteration with standard iterative methods like `cgs` in MATLAB are given in [76].

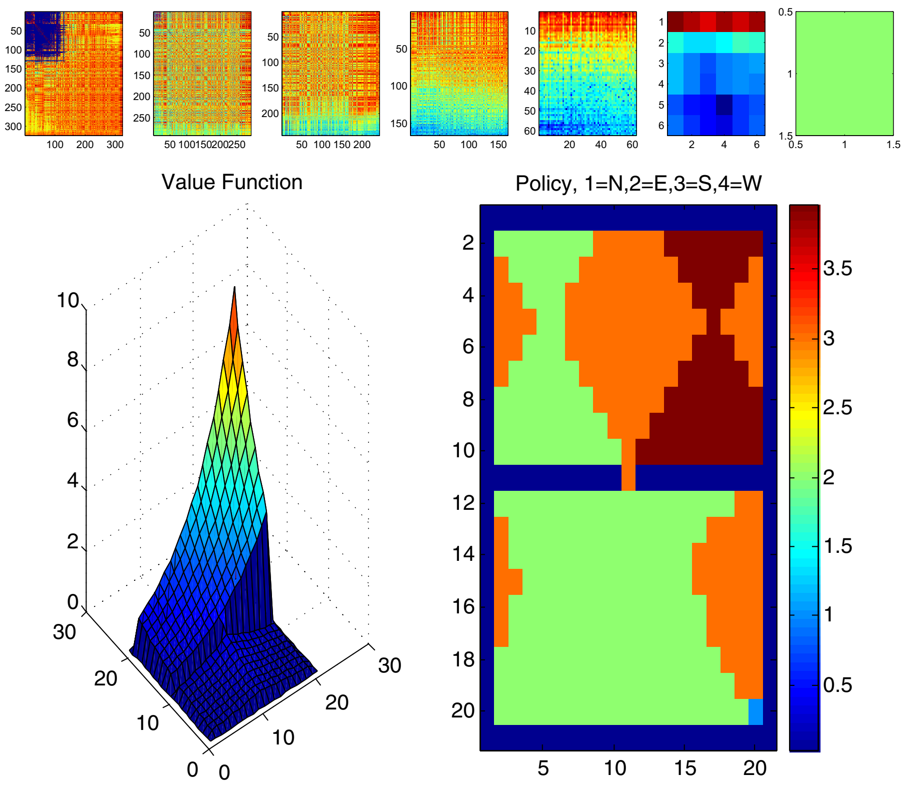


Fig. 8.6 Experimental evaluation of model-based RPI using diffusion wavelets in a 21×21 two-room domain. *Top*: Dyadic powers of the learned optimal policy transition matrix (entries are displayed in $\log_{10}$). Note that the $2^7 = 128th$ power is just a 1×1 matrix. *Bottom*: Learned optimal value function and optimal policy. The results are identical to direct inversion within the desired precision of $\epsilon = 10^{-6}$.

8.3.5 Factored Model-Based Approaches

Model-based basis construction methods depend on knowing the transition model and reward function. In experiments not shown here, it was found that sampling errors in estimating the transition model could degrade the performance. To scale such approaches to large MDPs requires exploiting significant structure in the transition matrix, such as using a factored dynamic Bayes net [51]. A number of studies have explored the problem of basis construction in factored MDPs [66, 108].

9

Basis Construction in Continuous MDPs

Thus far, the construction of bases was restricted to discrete MDPs. In this section, we turn to discuss basis construction in continuous MDPs, which present significant challenges not encountered in discrete state spaces. We will restrict our discussion primarily to Laplacian eigenfunction bases. We describe an extension of the graph Laplacian to continuous spaces called manifolds. A central result called *Hodge theorem* shows that the eigenfunctions of the manifold Laplacian provides a complete basis for all square-integrable (smooth) functions on a manifold. In practice, the basis functions can only be computed and stored on sampled real-valued states, and hence must be interpolated to novel states. We discuss sampling methods for interpolation of eigenfunction and wavelet bases on continuous spaces. We describe one standard method to extend Laplacian eigenfunctions from sample points to new points called the Nyström interpolation method. This approach has been studied previously in kernel methods [142] and spectral clustering [9].

9.1 Continuous Markov Decision Processes

A continuous state Markov decision process (MDP) $M = \langle S, A, P^a_{ss'}, R^a_{ss'} \rangle$ is defined by a set of states $S \subset \mathbb{R}^d$, a set of discrete actions A, a transition model $P^a_{ss'}$ specifying the distribution over future states s' when an action a is performed in state s, and a corresponding reward model $R^a_{ss'}$ specifying a scalar cost or reward. Usually, continuous control tasks are specified by some underlying *controller* or *plant* $s_{t+1} = f(s_t, a_t, \sigma_t)$ which specifies a functional mapping of a state s_t into a new state s_{t+1} in response to the control or action selected a_t and some (parametrically modeled) noise term σ_t. In this paper, we do not assume either the continuous control system or the noise model is known. Given a policy $\pi : S \to A$ mapping states to actions, its corresponding value function V^π specifies the expected long-term discounted sum of rewards received by the agent in a state s when actions are chosen using the policy. Any optimal policy π^* defines the same unique optimal action-value function $Q^*(s,a)$ which satisfies continuous Bellman equation:

$$Q^*(s,a) = \int_{s'} P^a_{ss'} \left(R^a_{ss'} + \max_{a'} \gamma Q^*(s', a') \right) \mathrm{d}s'.$$

A more detailed treatment can be found in standard treatises on MDPs [110]. We do not discuss the extension of MDPs to continuous actions, where the Bellman equation is often referred to as the *Hamilton–Bellman–Jacobi* (HJB) equation [93].

9.2 Riemannian Manifolds

There is a rich and well-developed theory of the Laplace operator on manifolds, which is briefly summarized in this section. The Laplace-Beltrami operator has been extensively studied in the general setting of Riemannian manifolds [115]. Riemannian manifolds have been actively studied recently in machine learning in several contexts, namely in semi-supervised learning [8], in designing new types of kernels for supervised machine learning [67] and faster policy gradient methods using the natural Riemannian gradient on a space of parametric policies [5, 61, 105]. The Laplacian on Riemannian manifolds and its eigenfunctions [115], which form an orthonormal basis for square-integrable functions on the manifold (Hodge's theorem), generalize Fourier analysis to manifolds. Historically, manifolds have been applied to many problems in AI, for example, configuration space planning in robotics, but these problems assume a model of the manifold is known [71, 72], unlike here where only samples of a manifold are given.

It has been known for over 50 years that the space of probability distributions forms a Riemannian manifold, with the Fisher information metric representing the Riemann metric on the tangent space. This observation has been applied to design new types of kernels for supervised machine learning [67] and faster policy gradient methods using the natural Riemannian gradient on a space of parametric policies [5, 61, 105]. Recently, there has been rapidly growing interest in manifold learning methods, including ISOMAP [131], LLE [116], and Laplacian eigenmaps [8]. These methods have been applied to nonlinear dimensionality reduction as well as semi-supervised learning on graphs [8, 28, 144].

9.2.1 Manifolds

This section introduces the Laplace-Beltrami operator in the general setting of Riemannian manifolds [115], as an extension of the graph Laplacian operator described earlier in the more familiar setting of graphs reviewed in Section 3. The material in this section is purely intended for review, and it is not necessary to understand the Nyström interpolation method described later.

Formally, a *manifold* $\mathcal{M}$ is a *locally Euclidean* set, with a *homeomorphism* (a bijective or one-to-one and onto mapping) from any open set containing an element $p \in \mathcal{M}$ to the n-dimensional Euclidean space $\mathbb{R}^n$. Manifolds with *boundaries* are defined using a homeomorphism that maps elements to the upper half plane $\mathcal{H}^n$ [74]. A manifold is a topological space, i.e., a collection of open sets closed under finite intersection and arbitrary union. In smooth manifolds, the homeomorphism becomes a *diffeomorphism*, or a continuous bijective mapping with a continuous inverse mapping, to the Euclidean space $\mathbb{R}^n$.

In a smooth manifold, a diffeomorphism mapping any point $p \in \mathcal{M}$ to its *coordinates* $(\rho_1(p), \ldots, \rho_n(p))$ should be a differentiable function with a differentiable inverse. Given two coordinate functions $\rho(p)$ and $\xi(p)$, or *charts*, the induced mapping $\psi : \rho \circ \xi^{-1} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ must have continuous partial derivatives of all orders. *Riemannian* manifolds are smooth manifolds where the Riemann metric defines the notion of length. Given any element $p \in \mathcal{M}$, the *tangent space* $T_p(\mathcal{M})$ is an n-dimensional vector space, that is, isomorphic to $\mathbb{R}^n$. A Riemannian manifold is a smooth manifold $\mathcal{M}$ with a family of smoothly varying positive semidefinite inner products $g_p, p \in \mathcal{M}$, where $g_p : T_p(\mathcal{M}) \times T_p(\mathcal{M}) \to \mathbb{R}$. For the Euclidean space $\mathbb{R}^n$, the tangent space $T_p(\mathcal{M})$ is clearly isomorphic to $\mathbb{R}^n$ itself. One example of a Riemannian inner product on $\mathbb{R}^n$ is simply $g(x,y) = \langle x,y\rangle_{\mathbb{R}^n} = \sum_i x_i y_i$, which remains the same over the entire space. If the space is defined by the set of probability distributions $P(X|\theta)$, then one example of a Riemann metric is given by the Fisher information metric $\mathcal{I}(\theta)$ [67].

9.2.2 Hodge Theorem

Hodge's theorem [115] states that any smooth function on a compact manifold has a discrete spectrum mirrored by the *eigenfunctions* of Δ, the Laplace-Beltrami self-adjoint operator. On the manifold $\mathbb{R}^n$, the Laplace-Beltrami operator is $\Delta = \sum_i \frac{\partial^2}{\partial x_i^2}$ (often written with a — sign for convention). Functions that solve the equation $\Delta f = 0$ are called *harmonic functions* [4]. For example, on the plane $\mathbb{R}^2$, the "saddle" function $x^2 - y^2$ is harmonic. *Eigenfunctions* of Δ are functions f such that $\Delta f = \lambda f$, where λ is an eigenvalue of Δ. If the domain is the unit circle S^1, the trigonometric functions $\sin(\theta)$ and $\cos(\theta)$ form eigenfunctions, which leads to *Fourier* analysis. Abstract harmonic analysis generalizes Fourier methods to smooth functions on arbitrary Riemannian manifolds. The *smoothness functional* for an arbitrary real-valued function on the manifold $f : \mathcal{M} \to \mathbb{R}$ is given by

$$S(f) \equiv \int_{\mathcal{M}} |\nabla f|^2 \, d\mu = \int_{\mathcal{M}} f \Delta f d\mu = \langle \Delta f, f\rangle_{\mathbb{L}^2(\mathcal{M})},$$

where $\mathbb{L}^2(\mathcal{M})$ is the space of smooth functions on $\mathcal{M}$, and ∇f is the gradient vector field of f. We refer the reader to [115] for an introduction to Riemannian geometry and properties of the Laplacian on Riemannian manifolds. Let $(\mathcal{M}, g)$ be a smooth compact connected

Riemannian manifold. The Laplacian is defined as

$$\Delta = \operatorname{div}\operatorname{grad} = \frac{1}{\sqrt{\det g}} \sum_{ij} \partial_i \left(\sqrt{\det g}\; g^{ij} \partial_j \right),$$

where g is the Riemannian metric, $\det g$ is the measure of volume on the manifold, ∂_i denotes differentiation with respect to the i^{th} coordinate function, and div and grad are the Riemannian divergence and gradient operators. We say that $\phi : \mathcal{M} \to \mathbb{R}$ is an eigenfunction of Δ if $\phi \neq 0$ and there exists $\lambda \in \mathbb{R}$ such that

$$\Delta\phi = \lambda\phi.$$

If $\mathcal{M}$ has a boundary, special conditions need to be imposed. Typical boundary conditions include Dirichlet conditions, enforcing $\phi = 0$ on $\partial\mathcal{M}$ and Neumann conditions, enforcing $\partial_\nu \phi = 0$, where ν is the normal to $\partial\mathcal{M}$. The set of λ's for which there exists an eigenfunction is called the spectrum of Δ, and is denoted by $\sigma(\Delta)$. We always consider eigenfunctions which have been $\mathbb{L}^2$-normalized, i.e., $\|\phi\|_{\mathbb{L}^2(\mathcal{M})} = 1$.

The quadratic form associated to the Laplacian is the Dirichlet integral

$$\begin{aligned} S(f) &:= \int_{\mathcal{M}} \|\text{grad } f\|^2 \mathrm{d}\,\text{vol} \\ &= \int_{\mathcal{M}} f \Delta f \mathrm{d}\,\text{vol} = \langle \Delta f, f \rangle_{\mathbb{L}^2(\mathcal{M})} = \|\text{grad } f\|_{\mathbb{L}^2(\mathcal{M})}, \end{aligned}$$

where $\mathbb{L}^2(\mathcal{M})$ is the space of square-integrable functions on $\mathcal{M}$, with respect to the natural Riemannian volume measure. It is natural to consider the space of functions $\mathcal{H}^1(\mathcal{M})$ defined as follows

$$\mathcal{H}^1(\mathcal{M}) = \left\{ f \in \mathbb{L}^2(\mathcal{M}) : \|f\|_{\mathcal{H}^1(\mathcal{M})} := \|f\|_{\mathbb{L}^2(\mathcal{M})} + S(f) \right\}. \qquad (9.1)$$

So clearly $\mathcal{H}^1(\mathcal{M}) \subsetneq \mathbb{L}^2(\mathcal{M})$ since functions in $\mathcal{H}^1(\mathcal{M})$ have a square-integrable gradient. The smaller the $\mathcal{H}^1$-norm of a function, the "smoother" the function is, since it needs to have small gradient. Observe that if ϕ_λ is an eigenfunction of Δ with eigenvalue λ, then $S(\phi_\lambda) = \lambda$: the larger is λ, the larger the square-norm of the gradient of the corresponding eigenfunction, i.e., the more oscillating the eigenfunction is.

Theorem 9.1 (Hodge [115]). Let $(\mathcal{M}, g)$ be a smooth compact connected oriented Riemannian manifold. The spectrum $0 \leq \lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_k \leq \cdots, \lambda_k \to +\infty$, of Δ is discrete, and the corresponding eigenfunctions $\{\phi_k\}_{k \geq 0}$ form an orthonormal basis for $\mathbb{L}^2(\mathcal{M})$.

In other words, Hodge's theorem shows that a smooth function $f \in \mathbb{L}^2(\mathcal{M})$ can be expressed as

$$f(x) = \sum_{i=0}^{\infty} a_i e_i(x),$$

where e_i are the eigenfunctions of Δ, i.e., $\Delta e_i = \lambda_i e_i$. Smoothness can be defined as

$$S(e_i) = \langle \Delta e_i, e_i \rangle_{\mathbb{L}^2(\mathcal{M})} = \lambda_i.$$

In particular, any function $f \in \mathbb{L}^2(\mathcal{M})$ can be expressed as

$$f(x) = \sum_{k=0}^{\infty} \langle f, \phi_k \rangle \phi_k(x),$$

with convergence in $\mathbb{L}^2(\mathcal{M})$ [98].

9.3 Sampling Techniques

The smoothness of functions on a manifold as defined by Equation (9.1) determines the number of samples necessary to approximate the function up to a given precision. This number of samples is independent of the number of points explored. Consider the following simple example. Suppose the state space is the interval $[0, 1]$, and that the function f is band-limited with bandwidth B. This means that the Fourier transform $\hat{f}$ is supported in $[-B, B]$. Then by the Whittaker-Shannon sampling theorem [86], only $B/(2\pi)$ equispaced samples are needed to recover V *exactly*. Suppose we have observed samples $\mathcal{S}'$ in a space $\mathcal{S}$, and that the function f is smooth so that a subset $\mathcal{S}''$ much smaller than $\mathcal{S}'$ would suffice to determine f. We propose two simple methods in order to select $\mathcal{S}''$.

Purely random sub-sampling: We fix $|\mathcal{S}''|$, and select $|\mathcal{S}''|$ points uniformly at random in $\mathcal{S}'$. For very large $|\mathcal{S}'|$ one would expect that the points in $\mathcal{S}''$ are going to be well-spread in $\mathcal{S}'$.

Well-spread random net: The previous algorithm has two main drawbacks: first, it is not clear how to select $|\mathcal{S}''|$, even if in theory this number can be determined by knowing the complexity of the value function to be approximated. Second, the points in $\mathcal{S}''$ are not going to be necessarily well-spread in $\mathcal{S}'$: while it is true that for large $|\mathcal{S}'|$, with very high probability, no two points in $\mathcal{S}''$ are going to be very close, it is not true that the points in $\mathcal{S}''$ are going to be roughly equidistant nor well equidistributed in balls contained in $\mathcal{S}'$.

In order to guarantee that the set of points is well-spread, we consider the following construction. We define an ϵ-net of points in $\mathcal{S}'$ to be a subset $\mathcal{S}''$ such that no two points are closer than ϵ, and that for every point y in $\mathcal{S}'$, there is a point in $\mathcal{S}''$ which is not farther than ϵ from y. One can construct a (random) ϵ-net in $\mathcal{S}'$ as follows. Pick $x_0 \in \mathcal{S}'$ at random. By induction, for $k \geq 1$ suppose $x_0, x_1, \ldots, x_k$ have been picked so that the distance between any pair is larger than ϵ. If

$$R_k := \mathcal{S}' \Bigg\backslash \left(\bigcup_{l=1}^{k} B_\epsilon(x_l) \right),$$

is empty, stop, otherwise pick a point x_{k+1} in R_k. By definition of R_k the distance between x_{k+1} and any of the points $x_0, \ldots, x_k$ is not smaller than ϵ. When this process stops, say after k^* points have been selected, for any $y \in \mathcal{S}'$ we can find a point in $\mathcal{S}''$ not farther than ϵ, for otherwise $y \in R_{k^*}$ and the process would not have stopped. One can prove upper bounds of the distance between the eigenfunctions of the Laplacian on $\mathcal{S}'$ and the eigenfunctions of the Laplacian on $\mathcal{S}''$, which depend on ϵ and the order of the eigenfunction. We will explore the effectiveness of these sampling methods for learning basis functions in continuous MDPs in the next section.

9.4 Learning Eigenfunctions Using Nyström Extension

To learn policies on continuous MDPs, it is necessary to be able to extend eigenfunctions computed on a set of points $\in \mathbb{R}^n$ to new unexplored points. We describe here the Nyström method, which can be combined with iterative updates and randomized algorithms for low-rank approximations. The Nyström method interpolates the value of eigenvectors computed on sample states to novel states, and is an application of a classical method used in the numerical solution of integral equations [6]. The eigenfunction problem can be stated as

$$\int_D K(x,y)\phi(y)\mathrm{d}y = \lambda\phi(x), \quad \forall\ x \in D, \tag{9.2}$$

where D can be any domain, e.g., $\mathbb{R}$. Using the standard quadrature approximation, the above integral can be written as

$$\int_D K(x,y)\phi(y)\mathrm{d}y \approx \sum_{i=1}^{n} w_i k(x,s_i)\hat{\phi}(s_i), \tag{9.3}$$

where w_i are the quadrature weights, s_i are n selected sample points, and $\hat{\phi}$ is an approximation to the true eigenfunction. Combining Equations (9.2) and (9.3) gives us

$$\sum_{i=1}^{n} w_i k(x,s_i)\hat{\phi}(s_i) = \hat{\lambda}\hat{\phi}(x). \tag{9.4}$$

By letting x denote any set of n points, for example, the set of quadrature points s_i itself, the kernel $k(s_i,s_j)$ becomes a symmetric matrix. This enables computing the approximate eigenfunction at any new point as

$$\hat{\phi}_m(x) = \frac{1}{\hat{\lambda}} \sum_{i=1}^{n} w_i k(x,s_i)\hat{\phi}_m(s_i). \tag{9.5}$$

Let us instantiate Equation (9.5) in the context of the normalized Laplacian $\mathcal{L} = I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$. First, note that if λ_i is an eigenvalue of $\mathcal{L}$, then $1 - \lambda_i$ is the corresponding eigenvalue of the diffusion matrix $D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$. Applying the the Nyström extension for computing the eigenfunctions of the normalized Laplacian $\mathcal{L}\phi_i = \lambda_i\phi_i$, we get the equation

$$\phi_i(x) = \frac{1}{1-\lambda_i} \sum_{y \sim x} \frac{w(x,y)}{\sqrt{\mathrm{d}(x)\mathrm{d}(y)}}\ \phi_i(y), \tag{9.6}$$

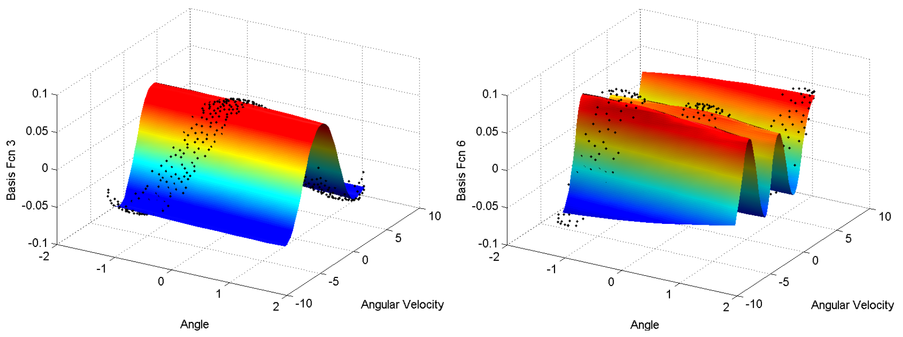


Fig. 9.1 This figure illustrates the Nyström interpolation method for extending eigenfunctions on samples to new states. *Left*: The 3*rd* eigenvector of the Laplacian plotted on a set of samples (shown as filled dots) drawn from a random walk in the inverted pendulum domain, as well as its Nyström interpolated values. *Right*: The Nyström interpolated 6*th* eigenvector illustrated on the entire state space as well as on the actual samples (again shown as filled dots).

where $\mathrm{d}(z) = \sum_{y \sim z} w(z, y)$, and x is a new vertex in the graph. Note that the weights $w(x, y)$ from the new state x to its nearest neighbors y in the previously stored samples is determined at "run time" using the same nearest neighbor (NN) weighting algorithm used to compute the original weight matrix W. An extensive discussion of the Nyström method is given in [39], and more details of its application to learning control in MDPs are given in [84].

Figure 9.1 illustrates the basic idea. Note that the Nyström method does *not* require recalculating eigenvectors — in essence, the embedding of a new state is computed by averaging over the already computed embeddings of "nearby" states. In practice, significant speedups can be exploited by using the following optimizations. We have empirically observed that roughly only 10% of the overall samples needed for learning a good policy are necessary to construct basis functions. Once the bases is defined over these sub-sampled states, the Nyström extended embeddings of the remaining 90% of training samples needs to be calculated only once, and henceforth can be cached during repeated runs of policy iteration. During testing, the Nyström embeddings of novel states encountered must be computed, but since the eigenvectors are defined over a relatively small core set of sample states, the extensions can be computed very efficiently using a fast NN algorithm.

10

Model-Free Representation Policy Iteration

We now discuss a model-free variant of the representation policy iteration (RPI) framework. We will primarily focus on Laplacian eigenfunction bases (PVFs) as well as diffusion wavelets, described in Sections 6 and 7. We will exploit the decomposition approach described in Section 6, and illustrate how Laplacian eigenfunction bases can be used successfully in a multiagent MDP with over 10^6 states. One of the challenges in applying these methods in continuous MDPs is that basis functions are generated from graphs constructed by sampling a continuous space. We will apply the sampling and interpolation methods described in Section 6 and present results comparing Laplacian eigenfunction bases to parametric bases, such as radial basis functions, on standard benchmark continuous control tasks. In some cases, large speedups are possible with automatically generated bases. We also show positive results on a 4-dimensional Acrobot control task using diffusion wavelets. Here, we show how Krylov bases can be used to tri-diagonalize a symmetric graph Laplacian operator, and speed-up the generation of multiscale wavelet bases.

10.1 Model-Free Representation Policy Iteration

Having described a variety of methods for constructing basis functions from operators, we now describe a model-free variant of the RPI framework previously introduced in Section 8. Figure 10.1 describes the overall algorithm. For the sake of concreteness, this procedure assumes that bases are constructed by diagonalizing or dilating an operator such as the normalized graph Laplacian. In the sample collection phase, an initial random walk (perhaps guided by an informed policy) is carried out to obtain samples of the underlying manifold on the state space. The number of samples needed is an empirical question. Given this set of samples, in the representation learning phase, an undirected (or directed) graph is constructed in one of several ways: two states can be connected by a unit cost edge if they represent temporally successive states; alternatively, a local distance measure such as k-NN can be used to connect states, which is particularly useful in the experiments on continuous domains reported below. From the graph, PVFs are computed using one of the graph operators, for example, the combinatorial or normalized Laplacian. The smoothest eigenvectors of the graph Laplacian (i.e., associated with the smallest eigenvalues) are used to form the suite of PVFs. The number of PVFs needed is a model selection question, which will be empirically investigated in the experiments described later. The encoding $\phi(s) : S \to \mathbb{R}^k$ of a state is computed as the value of the k PVFs on that state. To compute a state action encoding, a number of alternative strategies can be followed: the figure shows the most straightforward method of simply replicating the length of the state encoding by the number of actions and setting all the vector components to 0 except those associated with the current action.

10.2 Scaling to Large Discrete MDPs

In this section, we build on the general framework for scaling basis construction to large *factored* spaces described in Section 6.3, RPI (π_m,T,N,ϵ,k,$\mathcal{O}$,$\mathcal{D}$):

// π_m: Policy at the beginning of trial m
// T: Number of initial random walk trials
// N: Maximum length of each trial
// ϵ : Convergence condition for policy iteration
// k: Number of proto-value basis functions to use
// $\mathcal{O}$: Type of graph operator used
// $\mathcal{D}$: Initial set of samples

Sample Collection Phase

- **Off-policy or on-policy sampling:** Collect a data set of samples $\mathcal{D}_m = \{(s_i, a_i, s_{i+1}, r_i), \ldots\}$ by either randomly choosing actions (off-policy) or using the supplied initial policy (on-policy) for a set of T trials, each of maximum N steps.
- **(Optional) Subsampling step:** Form a subset of samples $\mathcal{D}_s \subseteq \mathcal{D}$ by some subsampling method such as random subsampling or trajectory subsampling.

Representation Learning Phase

- Build a diffusion model from the data in $\mathcal{D}_s$. In the simplest case of discrete MDPs, construct an undirected weighted graph G from $\mathcal{D}$ by connecting state i to state j if the pair (i,j) form temporally successive states $\in S$. Compute the operator $\mathcal{O}$ on graph G, for example the normalized Laplacian $\mathcal{L} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$.
- **Diagonalization:** Compute the k smoothest eigenvectors of $\mathcal{O}$ on the graph G. Collect them as columns of the basis function matrix Φ, a $|S| \times k$ matrix.
- **Dilation:** Build the diffusion wavelet tree from $\mathcal{O}$, and select k scaling functions and wavelets. Collect them as columns of the basis function matrix Φ, a $|S| \times k$ matrix.

Control Learning Phase

- Using a standard parameter estimation method (e.g. Q-learning or LSPI), find an ϵ-optimal policy π that maximizes the action value function $Q^\pi = \Phi w^\pi$ within the linear span of the bases Φ using the training data in $\mathcal{D}$.
- **Optional:** Set the initial policy π_{m+1} to π and call RPI(π_{m+1},T,N,ϵ,k,$\mathcal{O}$,$\mathcal{D}$).

Fig. 10.1 This figure shows a generic algorithm for jointly learning representation and control, where representations are specifically constructed by diagonalizing (or dilating an initial basis with) a graph operator, such as the normalized graph Laplacian.

where we exploit the spectral properties of the graph Laplacian in constructing embeddings that are highly regular for structured graphs (see Figure 6.3). In particular, as we saw previously, the eigenspace of the Kronecker sum of two graphs is the Kronecker product of the eigenvectors of each component graph.

10.2.1 Constructing Complex Bases from Simpler Bases

We can construct complex basis functions for large graphs out of the simpler bases for the "building block" component graphs. Recall that basis functions in this context are *column* eigenvectors of the diagonalized representation of a graph operator, whereas embeddings $\phi(s)$ are *row* vectors representing the first k basis functions evaluated on state s. By exploiting the property that $(A \otimes B)^T = A^T \otimes B^T$, it follows that embeddings for structured graphs can be computed as the Kronecker products of embeddings for the constituent state components. As a concrete example, a grid world domain of size $m \times n$ can be represented as a graph $G = G_m \oplus G_n$, where G_m and G_n are *path graphs* of size m and n, respectively. The basis functions for the entire grid world can be written as the Kronecker product $\phi(s) = \phi_m(s^r) \otimes \phi_n(s^c)$, where $\phi_m(s^r)$ is the basis (eigen)vector derived from a path graph of size m (in particular, the row s^r corresponding to state s in the grid world), and $\phi_n(s^c)$ is the basis (eigen)vector derived from a path graph of size n (in particular, the column s^c corresponding to state s in the grid world).

Extending this idea to state action pairs, the basis function $\phi(s,a)$ can written as $e_I(a) \otimes \phi(s)$, where $e_I(a)$ is the unit vector corresponding to the index of action a (e.g., action a_1 corresponds to $e_1 = [1, 0, \ldots]^T$). Actually, the full Kronecker product is not necessary if only a relatively small number of basis functions are needed. For example, if 50 basis functions are to be used in a $10 \times 10 \times 10$ hypercube, the full state embedding is a vector of size 1000, but only the first 50 terms need to be computed. Such savings imply PVFs can be efficiently computed even in very large structured domains. For a factored state space $s = (s^1, \ldots, s^m)$, we use the notation s^i to denote the value of the ith component. We can restate the update rules for factored RPI and

LSPI as follows

$$\begin{aligned}\tilde{A}^{t+1} &= \tilde{A}^t + \phi(s_t,a_t)\left(\phi(s_t,a_t) - \gamma\phi(s'_t,\pi(s'_t))\right)^T \\ &= \tilde{A}^t + e_{I(a_t)} \otimes \prod_{\otimes} \phi_i(s^i_t) \\ &\quad \times \left(e_{I(a_t)} \prod_{\otimes} \phi_i(s^i_t) - \gamma e_{I(\pi(s'_t))} \otimes \prod_{\otimes} \phi_i(s'^i_t)\right)^T.\end{aligned}$$

The corresponding update equation for the reward component is

$$\tilde{b}^{t+1} = \tilde{b}^t + \phi(s_t,a_t)r_t = \tilde{b}^t + r_t e_{I(a_t)} \otimes \prod_{\otimes} \phi_i(s^i_t).$$

10.2.2 Experimental Results: Blocker Domain

We now present a detailed study using a much larger factored multiagent domain called the "Blockers" task, which was studied in [119]. This task, which was illustrated in Figure 2.1, is a cooperative multiagent problem where a group of agents try to reach the top row of a grid, but are prevented in doing so by "blocker" agents who move horizontally on the top row. If any agent reaches the top row, the entire team is rewarded by +1; otherwise, each agent receives a negative reward of -1 on each step. The agents always start randomly placed on the bottom row of the grid, and the blockers are randomly placed on the top row. The blockers remain restricted to the top row, executing a fixed strategy. The overall state space is the Cartesian product of the location of each agent. These experiments on the blocker domain include more difficult versions of the task not studied in [119] specifically designed to test the scalability of the Kronecker product bases to "irregular" grids whose topology deviates from a pure hypercube or toroid. In the first variant, shown on the left in Figure 2.1, horizontal interior walls extend out from the left and right side walls between the second and third row. In the second variant, an additional interior wall is added in the middle as shown on the right.[1]

[1] In the Blocker domain, the interior walls are modeled as having "zero width," and hence all 100 states in each grid remain accessible, unlike the two-room environment.

The basis functions for the overall Blocker state space were computed as Kronecker products of the basis functions over each agent's state space. Each agent's state space was modeled as a grid or a cylinder (for the "wrap-around" case). Since the presence of interior walls obviously violates the pure product of cylinders or grids topology, each individual agent's state space was learned from a random walk. The overall basis functions were then constructed as Kronecker products of Laplacian basis functions for each learned (irregular) state grid.

Figure 10.2 compares the performance of the factored Laplacian bases with a set of radial basis functions (RBFs) for the first Blocker domain (shown on the left in Figure 2.1). The width of each RBF was set at $\frac{2|S_a|}{k}$, where $|S_a|$ is the size of each individual agent's grid, and k is the number of RBFs used. The RBF centers were uniformly spaced. The results shown are averages over 10 learning runs. On each run, the learned policy is measured every 25 training episodes. Each episode

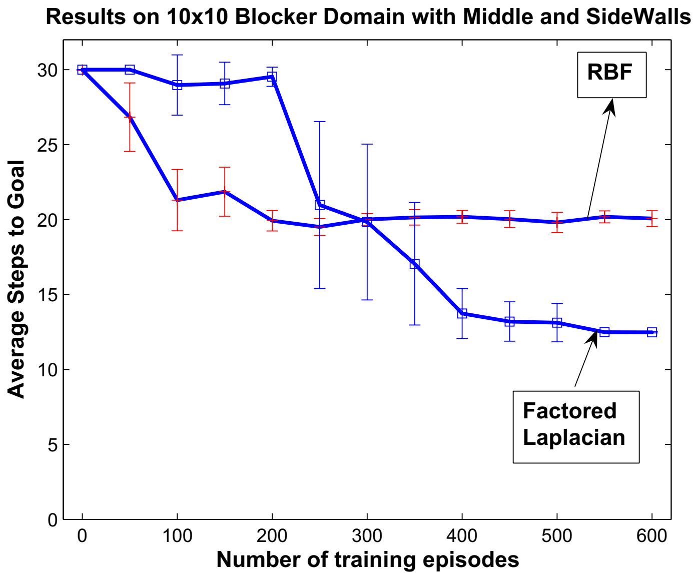


Fig. 10.2 This figure compares RPI using factored Laplacian eigenvector basis functions with LSPI using hand coded radial basis functions (RBF) on a 10 × 10 "wrap-around" Blocker domain. There were 3 agents and 2 blockers, resulting in a space of $>10^6$ states. Both approaches were tested using 100 basis functions.

begins with a random walk of a maximum of 70 steps (terminating earlier if the top row was reached). After every 25 such episodes, RPI is run on all the samples collected thus far. The learned policy is then tested over 500 test episodes. The graph plots the average number of steps to reach the goal. The experiments were conducted on both "normal" grids (not shown) and "wrap around" cylindrical grids. The results show that RBFs converge faster, but learn a worse policy. The factored Laplacian bases converge slower than RBFs, but learn a substantially better policy.

10.3 Experimental Results of RPI in Continuous MDPs

In this section, we present experimental results applying the model-free RPI algorithm to continuous MDPs. We compare the automatically generated basis functions with parametric hand-tuned bases in standard benchmark MDPs. A more detailed evaluation of RPI can be found in [83]. A crucial question here is how to build a graph by subsampling the set of states visited during the initial learning period. We discuss this sampling problem next.

10.3.1 Graph Construction from Point Sets in $\mathbb{R}^n$

Given a continuous MDP, samples are collected using some policy, either a random walk or a more informed policy from a previous round of RPI. Given a data set $\{x_i\}$ in $\mathbb{R}^n$, different weighted graphs can be constructed from this point set. There are different choices of edges and for any such choice there is a choice of weights on the edges. In the experiments below, the following construction was used. Edges were inserted between a pair of states x_i and x_j if:

- x_j is among the k-NNs of x_i, where $k > 0$ is a parameter.

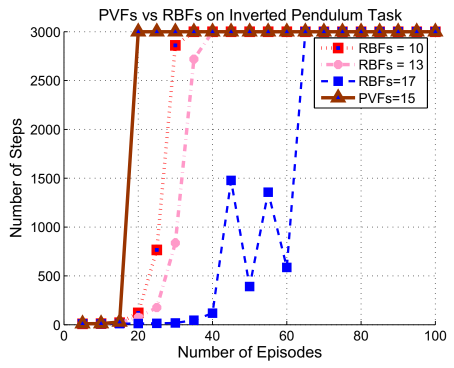
Weights were assigned to the edges in the following way:

- $W(i,j) = \alpha(i) e^{-\frac{\|x_i - x_j\|^2_{\mathbb{R}^n}}{\sigma}}$, where $\sigma > 0$ is a parameter, and α a weight function to be specified.

Observe that for undirected graphs, since x_j can be among the K-NNs of x_i but x_i may not be among the K-NNs of x_j, the above construction will still yield asymmetric weight matrices. An additional symmetrization step is needed where the weight matrix W is replaced by the symmetric $W + W^T$. If the states $\{x_i\}$ are drawn uniformly from a Riemannian manifold, then it is shown in [8] that the above construction, with $\alpha = 1$, approximates the continuous Laplace-Beltrami operator on the underlying manifold. If $\{x_i\}$ is not drawn uniformly from the manifold, as it typically happens in MDPs when the space is explored by an agent, it is shown in [68] that a pre-processing normalization step can (must) be performed that yields the weight function α, so that the above construction yields an approximation to the Laplace-Beltrami operator. Various ways of normalizing the weight matrix are explored elsewhere [83]. Here, it is assumed that the operator T used is the normalized Laplacian $\mathcal{L} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$.

A brief comparison of Laplacian eigenfunctions or PVFs with RBFs is now given. Radial basis functions (RBFs) are a popular choice of basis functions for both discrete and continuous MDPs. The comparison of PVFs and RBFs is restricted to the inverted pendulum and mountain car domains. To choose a suitable set of parameters for RBFs, the kernel widths were fine-tuned to find the best performing setting for RBFs. For PVFs, the number of bases and the scaling of the state space dimensions were varied to find the best performing setting. Generally speaking, the results below demonstrate that PVFs are significantly quicker to converge, by almost a factor of two in both the inverted pendulum and mountain car domains. Asymptotically, both approaches converge to the same result. These comparisons are meant to be *suggestive*, and not definitive. For example, fine-tuning the centers of the RBF bases, or incorporating the scaling factors used in the experiments with PVFs may produce better results with RBFs.

Inverted pendulum: Figure 10.3 compares the performance of PVFs with a linear RBF approximation architecture for the inverted pendulum domain, for a varying number of RBF architectures, with 15 PVFs. PVFs converge significantly faster to the final goal of balancing the pendulum for 3000 steps: PVFs take 20 trials to converge, but RBFs take roughly twice as long.



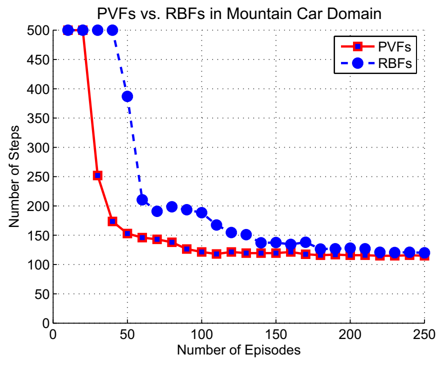


Fig. 10.3 This figure compares model-free RPI using automatically generated PVF bases with LSPI using hand-tuned RBF bases. *Left*: A comparison of 15 PVFs with several choices of RBFs on the inverted pendulum task, focusing on the initial 100 episodes averaged over 100 runs. *Right*: A comparison of 25 PVFs and 13 RBFs on the mountain car task. Higher number of RBFs produced worse results.

Mountain car: As with the inverted pendulum, the parameters for RBFs were determined by fine-tuning the kernel width, although the differences are less significant than in the inverted pendulum domain. Figure 10.3 plots the best performing RBF architecture (13 basis functions) compared with the PVF approach (25 basis functions). Given sufficient training experience, both converge to approximately the same result, although PVFs seem to converge to a slightly better result. However, as with the inverted pendulum results, PVFs converge significantly quicker, and clearly outperform RBFs for smaller numbers of samples.

10.4 Krylov-Accelerated Diffusion Wavelets

When an operator T is diagonalizable, the Krylov bases spanned by T and a function f are spanned by the projection of f onto its eigenspaces [88]. This insight can be used to develop faster algorithms for eigenspace projection, such as the well-known Lanczos method [49]. In other words, rather than directly projecting on the eigenspaces of a diagonalizable operator, Lanczos methods can be used to construct a highly compact tridiagonal representation of the operator T, constructed by restricting it to the *Krylov space* spanned by powers of T and a function f.

This idea can be applied to accelerate the construction of scaling function bases and wavelets, as recently shown in [79].

The Acrobot task [129] is a two-link under-actuated robot, that is, an idealized model of a gymnast swinging on a high bar. The only action available is a torque on the second joint, discretized to one of three values (positive, negative, and none). The reward is -1 for all transitions leading up to the goal state. The detailed equations of motion are given in [129]. The state space for the Acrobot is 4-dimensional. Each state is a 4-tuple represented by $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$. θ_1 and θ_2 represent the angle of the first and second links to the vertical, respectively, and are naturally in the range $(0, 2\pi)$. $\dot{\theta}_1$ and $\dot{\theta}_2$ represent the angular velocities of the two links. Notice that angles near 0 are actually very close to angles near 2π due to the rotational symmetry in the state space.

The time required to construct Krylov-accelerated diffusion wavelets with regular diffusion wavelets is shown in Figure 10.4 (left plot). There is a very significant decrease in running time using the Krylov-subspace restricted Lanczos tridiagonal matrix. In this experiment, data was generated doing random walks in the Acrobot domain, from an initial sample size of 100 to a final sample size of 1000 states. The performance of regular diffusion wavelets with Krylov-accelerated wavelets is shown on the right plot. This experiment was carried out

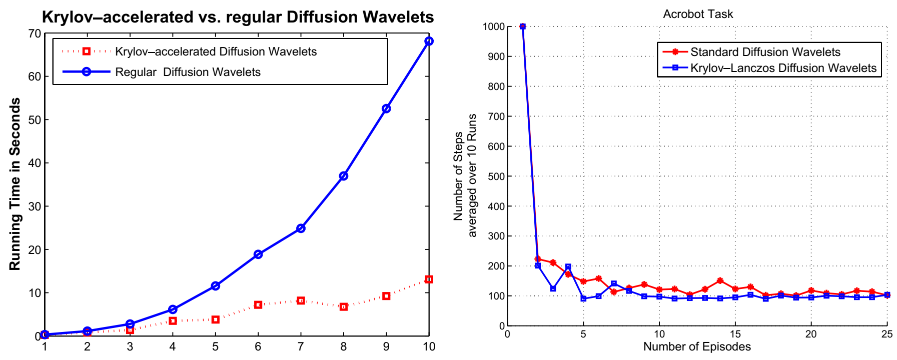


Fig. 10.4 This figure shows the performance of model-free RPI using multiscale diffusion wavelet bases. *Left*: Time required to construct regular versus Krylov-accelerated diffusion wavelets on the Acrobot control task; *Right*: The resulting performance with RPI on the Acrobot task.

using a modified form of RPI with on-policy re-sampling. Specifically, additional samples were collected during each new training episode from the current policy if it was the best-performing policy (in terms of the overall performance measure of the number of steps), otherwise a random policy was used. Note that the performance of Krylov-accelerated diffusion wavelets is slightly better than regular diffusion wavelets.

11

Related Work and Future Challenges

In this section, we briefly summarize related work in a number of different areas, including methods for approximating value functions to algorithms for constructing low-dimensional representations. We also outline a number of challenges that need to be addressed in scaling the proposed methods to larger MDPs: some of these directions have already been studied to some extent and where relevant, we discuss recent results.

11.1 Related Work

11.1.1 Value Function Approximation

Value function approximation with a fixed handcoded basis has been studied by many researchers. Largely, our presentation has been restricted to linear basis functions. However, there is a substantial literature on nonlinear methods for value function approximation using neural networks with fixed basis functions. Bertsekas and Tsitsiklis [12] provide a detailed review of nonlinear neural network methods. There has also been significant work on nonparametric methods for approximating value functions, including nearest neighbor (NN) methods

[50] and kernel density estimation [100]. Nonparametric kernel methods based on Hilbert spaces have also been applied to value function approximation, including support vector machines [38] and Gaussian processes [42, 111], where the kernels are pre-specified.

11.1.2 Representation Discovery

The basis construction methods described here can be applied to construct representations from data over a broad range of problems in AI, as described in [80]. The problem of representation discovery has a long history in AI. Amarel [3] was an early pioneer, advocating the study of representation learning through global state space analysis. Amarel's ideas motivated much subsequent research on representation discovery [128, 135], and many methods for discovering global state space properties like "bottlenecks" and "symmetries" have been studied [89, 90, 112]. The approaches described in this paper can be viewed as providing a formal framework showing how the geometrical analysis of a state space analysis can be transformed into useful representations for solving sequential decision problems.

There have been several attempts at overcoming the limitations of traditional function approximators, such as radial basis functions. In particular, it has been recognized that Euclidean smoothing methods do not incorporate geometric constraints intrinsic to the environment: states close in Euclidean distance may be far apart on the manifold. Dayan [35] proposed the idea of building *successor representations*. While this approach was restricted to policy evaluation in simple discrete MDPs, and did not formally build on manifold or graph–theoretic concepts, the idea of constructing representations that are faithful to the underlying dynamics of the MDP was a key motivation underlying this work. Drummond [40] also pointed out the nonlinearities that value functions typically exhibit, and used techniques from computer vision to detect nonlinearities. Neither of these studies formulated the problem of value function approximation as approximating functions on a graph or manifold, and both were restricted to discrete MDPs. There have been several attempts to dynamically allocate basis functions to regions of the state space based on the nonuniform occupancy probability of visiting a region (e.g., [64]), but these methods do not construct the basis functions adaptively. Finally, there has also been research on finding common structure among the set of value functions on a given state space, where only the goal location is changed [45], assuming a probabilistic generative (mixture) model of a value function, and using maximum likelihood estimation techniques.

11.1.3 Manifold and Spectral Learning

The approach of diagonalizing graph operators builds on recent work on manifold and spectral learning, including diffusion maps [28, 29, 31], ISOMAP [131], LLE [116], and Laplacian eigenmaps [8, 60]. One major difference is that these methods have largely (but not exclusively) been applied to nonlinear dimensionality reduction and semi-supervised learning on graphs, whereas this paper focuses on approximating (real-valued) value functions on graphs. Although related to regression on graphs [99], the problem of value function approximation is fundamentally different: the set of target values is not known *a priori*, but must be inferred through an iterative process of computing an approximate fixed point of the Bellman backup operator, and projecting these iterates onto subspaces spanned by the basis functions. Furthermore, value function approximation introduces new challenges not present in supervised learning or dimensionality reduction: the set of samples is not specified *a priori*, but must be collected through *active* exploration of the state space.

11.2 Future Work

This paper described recent work on solving Markov decision processes (MDPs) by simultaneously learning representation (basis functions) and control. Several approaches to constructing basis functions were described, principally by diagonalizing a Laplacian operator or by dilating a given basis or reward function using (the generalized inverse of) a Laplacian operator. Several Laplacian operators were described, ranging from those directly based on the transition matrix of a given policy to the natural random walk on a graph induced by the MDP on a set of sample states. Many extensions of these ideas are possible, and are being actively explored. Some of these ongoing investigations are briefly explored here.

11.2.1 Drazin Inverse of Laplacian Operators

A novel representation based on the Drazin inverse (or group inverse) of the Laplacian was shown to be useful, both in exactly solving MDPs as well as in approximating them. The approach was motivated by the theoretical result showing that the discounted value function can be written as a Laurent series involving powers of the Drazin inverse of the Laplacian [110]. While this approach was shown to be highly effective on small discrete MDPs given the transition model and the reward function, its approximation properties have yet to be studied in the case when the model must be learned from simulation data. Furthermore, computing the Drazin inverse can be computationally intractable in large MDPs. An interesting challenge is to develop algorithms that exploit the factored state structure or hierarchical task structure of large MDPs in scaling the computation of Drazin bases. There are also some interesting results showing that Drazin inverses can be updated incrementally when P changes slightly (e.g., a single row changes), without having to recompute the whole inverse again [70].

11.2.2 Transfer Learning

There are many examples of sequential decision problems in the literature, including robot navigation where multiple objects need to be retrieved in the same environment, and information retrieval where the Markov chain representing the web graph remains the same, but the retrieval goals change. A fundamental issue in basis construction that we have not addressed is how well a set of bases transfer from one problem to another. Work on this problem is beginning to be addressed by a number of researchers. For example, recent studies have investigated how well proto-value functions (PVFs) can be transferred across related tasks [43, 133].

11.2.3 Algorithms for Learning Representation and Control

This paper largely focused on combining basis construction methods with least-squares-based approximate policy iteration methods. As described in Section 4, there are a whole host of other ways of approximately computing policies in MDPs, including those that are based on convex optimization. An interesting challenge is to combine the basis construction methods described here with linear programming methods for solving MDPs [36] or reproducing kernel Hilbert space (RKHS) methods [42].

11.2.4 Basis Construction in Average-Reward MDPs

Largely, the proposed basis construction methods were evaluated in the discounted MDP framework. There are many interesting applications where the average-reward framework is more appropriate than the more commonly studied discounted framework, as discussed in [56, 110]. As described in Section 3, the Laplacian framework applies nicely to average-reward MDPs. In fact, the first two terms in the Drazin bases can be viewed as the average-reward approximation to a discounted MDP. A more comprehensive investigation of dimensionality reduction in average-reward MDPs is needed.

11.2.5 Exploiting Task Structure

It is well-known that task structure can be exploited in solving large MDPs [7]. It is possible to extend the approaches described here to semi-Markov decision processes (SMDPs), where actions are temporally extended, such as "exiting a room". For the graph-based approach, temporally extended actions result in longer "distal" edges connecting nonadjacent vertices (such as the vertices corresponding to interior states in a room with those representing the "door" state). Osentoski and Mahadevan [102] show that constructing PVFs over state-action graphs using these distal edges can significantly improve the performance over PVFs constructed over state graphs with only primitive actions. Recently, promising results have been obtained on large hierarchical MDPs such as the AGV domain shown earlier in Figure 2.4, where the basis functions are automatically constructed from the hierarchy [101]. The key idea is to construct a compressed graph at higher levels, and generate embeddings at more abstract levels by reusing embeddings from lower level subtasks.

11.2.6 Theoretical Analysis

Theoretical guarantees on the efficiency of constructing geometry-sensitive or reward-sensitive basis functions in approximating value functions and models need to be further investigated. The Drazin bases were motivated by the asymptotic convergence of the Laurent series to the discounted value function. However, the empirical results suggest that in practice, the Laurent series converges rather rapidly in many cases. A finite-dimensional analysis of this expansion would be valuable in understanding the properties of Markov chains for which fast convergence holds. Furthermore, both Drazin and Krylov bases were largely evaluated only for the model-based case. It is important to theoretically characterize their behavior when models have to be estimated from sampled transitions. For the graph-based basis construction methods, there is a rapidly growing literature showing how the various graph Laplacians converge to the Laplace-Beltrami operator on the underlying manifold. For example, Hein [53] shows that under non-uniform sampling conditions, the random walk Laplacian converges to a weighted Laplace-Beltrami operator. These results need to be combined with exploration techniques to investigate the conditions under which these sampling conditions can be met in the context of MDPs.

11.3 Summary

This paper described methods for automatically compressing Markov decision processes by learning a low-dimensional linear approximation defined by an orthogonal set of basis functions. A unique feature of this paper is the use of Laplacian operators, whose matrix representations have non-positive off-diagonal elements and zero row sums. The generalized inverses of Laplacian operators, in particular the Drazin inverse, was shown to be useful in the exact and approximate solution of MDPs. This paper also described a broad framework for solving MDPs, generically referred to as *representation policy iteration*, where both the basis function representation for approximation of value functions as well as the optimal policy within their linear span are simultaneously learned. Basis functions were constructed by diagonalizing a Laplacian operator, or by dilating the reward function or an initial set of bases by powers of the operator. The idea of decomposing an operator by finding its invariant subspaces was shown to be an important principle in constructing low-dimensional representations of MDPs. Theoretical properties of these approaches were discussed, and they were also compared experimentally on a variety of discrete and continuous MDPs. Challenges for further research were briefly outlined.

Acknowledgments

Portions of this research were carried out in collaboration with my PhD students. I would like to acknowledge Jeff Johns, Sarah Osentoski, and Chang Wang for their significant contributions to this research. In addition, I would also like to thank Mauro Maggioni at Duke University for much valuable help, including the use of his diffusion wavelet software. Ronald Parr of Duke University provided valuable comments on a draft of this paper. Finally, I would like to acknowledge the useful feedback of past and present members of the Autonomous Learning Laboratory at the University of Massachusetts, Amherst. Support for this research was provided in part by the National Science Foundation under grants IIS-0534999 and IIS-0803288.

References

[1] D. Achlioptas, F. McSherry, and B. Scholkopff, "Sampling techniques for Kernel methods," in *Proceedings of the 14th International Conference on Neural Information Processing Systems (NIPS)*, pp. 335–342, MIT Press, 2002.

[2] R. Agaev and P. Cheboratev, "On the spectra of nonsymmetric Laplacian matrices," *Linear Algebra and Its Applications*, vol. 399, pp. 157–168, 2005.

[3] S. Amarel, "On representations of problems of reasoning about actions," in *Machine Intelligence 3*, (D. Michie, ed.), pp. 131–171, Elsevier/North-Holland, 1968.

[4] S. Axler, P. Bourdon, and W. Ramey, *Harmonic Function Theory.* Springer, 2001.

[5] J. Bagnell and J. Schneider, "Covariant Policy Search," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1019–1024, 2003.

[6] C. T. H. Baker, *The Numerical Treatment of Integral Equations.* Oxford: Clarendon Press, 1977.

[7] A. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Systems Journal*, vol. 13, pp. 41–77, 2003.

[8] M. Belkin and P. Niyogi, "Semi-supervised learning on Riemannian manifolds," *Machine Learning*, vol. 56, pp. 209–239, 2004.

[9] S. Belongie, C. Fowlkes, F. Chung, and J. Malik, "Spectral partitioning with indefinite Kernels using the Nyström extension," in *Proceedings of the 7th European Conference on Computer Vision*, pp. 531–542, 2002.

[10] A. Berman and R. Plemmons, *Nonnegative Matrices in the Mathematical Sciences.* SIAM Press, 1994.

[11] D. Bertsekas and D. Castanon, “Adaptive Aggregation Methods for infinite horizon dynamic programming,” *IEEE Transactions on Automatic Control*, vol. 34, pp. 589–598, 1989.

[12] D. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming.* Belmont, MA: Athena Scientific, 1996.

[13] B. Bethke, J. How, and A. Ozdaglar, “Approximate dynamic programming using support vector regression,” in *Proceedings of the IEEE Conference on Decision and Control*, 2008.

[14] G. Beylkin, R. R. Coifman, and V. Rokhlin, “Fast wavelet transforms and numerical algorithms,” *Common Pure and Applied Mathematic*, vol. 44, pp. 141–183, 1991.

[15] L. Billera and P. Diaconis, “A geometric interpretation of the Metropolis-Hasting algorithm,” *Statistical Science*, vol. 16, pp. 335–339, 2001.

[16] J. A. Boyan, “Least-squares temporal difference learning,” in *Proceedings of the 16th International Conference on Machine Learning*, pp. 49–56, San Francisco, CA: Morgan Kaufmann, 1999.

[17] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, 2004.

[18] S. Bradtke and A. Barto, “Linear least-squares algorithms for temporal difference learning,” *Machine Learning*, vol. 22, pp. 33–57, 1996.

[19] J. Bremer, R. Coifman, M. Maggioni, and A. Szlam, “Diffusion wavelet packets,” *Applied and Computational Harmonic Analysis*, vol. 21, no. 1, pp. 95–112, July 2006.

[20] S. Campbell and C. Meyer, *Generalized Inverses of Linear Transformations.* Pitman, 1979.

[21] X. Cao, “The relations among potentials, perturbation analysis, and Markov decision processes,” *Discrete-Event Dynamic Systems*, vol. 8, no. 1, pp. 71–87, 1998.

[22] J. Caughman and J. Veerman, “Kernels of directed graph Laplacians,” *Electronic Journal of Combinatorics*, vol. 13, no. 1, pp. 253–274, 2006.

[23] I. Chavel, *Eigenvalues in Riemannian Geometry: Pure and Applied Mathematics.* Academic Press, 1984.

[24] P. Chebotarev and R. Agaev, “Forest matrices around the Laplacian matrix,” *Linear Algebra and Its Applications*, vol. 15, no. 1, pp. 253–274, 2002.

[25] L. Chen, E. Krishnamurthy, and I. Macleod, “Generalized matrix inversion and rank computation by repeated squaring,” *Parallel Computing*, vol. 20, pp. 297–311, 1994.

[26] F. Chung, *Spectral Graph Theory,* Number 92 in *CBMS Regional Conference Series in Mathematics.* American Mathematical Society, 1997.

[27] F. Chung, “Laplacians and the Cheeger inequality for directed graphs,” *Annals of Combinatorics*, vol. 9, no. 1, pp. 1–19, April 2005.

[28] R. Coifman, S. Lafon, A. Lee, M. Maggioni, B. Nadler, F. Warner, and S. Zucker, “Geometric diffusions as a tool for harmonic analysis and structure definition of data. Part i: Diffusion maps,” *Proceedings of National Academy of Science*, vol. 102, no. 21, pp. 7426–7431, May 2005.

[29] R. Coifman, S. Lafon, A. Lee, M. Maggioni, B. Nadler, F. Warner, and S. Zucker, “Geometric diffusions as a tool for harmonic analysis and structure definition of data. Part ii: Multiscale methods,” *Proceedings of the National Academy of Science*, vol. 102, no. 21, pp. 7432–7437, May 2005.

[30] R. Coifman and M. Maggioni, “Diffusion wavelets,” *Applied and Computational Harmonic Analysis*, vol. 21, no. 1, pp. 53–94, July 2006.

[31] R. Coifman, M. Maggioni, S. Zucker, and I. Kevrekidis, “Geometric diffusions for the analysis of data from sensor networks,” *Curr Opin Neurobiol*, vol. 15, no. 5, pp. 576–584, October 2005.

[32] D. Cvetkovic, M. Doob, and H. Sachs, *Spectra of Graphs: Theory and Application.* Academic Press, 1980.

[33] T. Das, A. Gosavi, S. Mahadevan, and N. Marchalleck, “Solving semi-Markov decision problems using average-reward reinforcement learning,” *Management Science*, vol. 45, no. 4, pp. 560–574, 1999.

[34] I. Daubechies, *Ten Lectures on Wavelets*, Society for Industrial and Applied Mathematics. 1992.

[35] P. Dayan, “Improving generalisation for temporal difference learning: The successor representation,” *Neural Computation*, vol. 5, pp. 613–624, 1993.

[36] D. de Farias, “The linear programming approach to approximate dynamic programming,” in *Learning and Approximate Dynamic Programming: Scaling Up to the Real World*, John Wiley and Sons, 2003.

[37] F. Deutsch, *Best Approximation in Inner Product Spaces.* Canadian Mathematical Society, 2001.

[38] T. Dietterich and X. Wang, “Batch value function approximation using support vectors,” in *Proceedings of Neural Information Processing Systems*, MIT Press, 2002.

[39] P. Drineas and M. W. Mahoney, “On the Nyström method for approximating a Gram matrix for improved Kernel-based learning,” *Journal of Machine Learning Research*, vol. 6, pp. 2153–2175, 2005.

[40] C. Drummond, “Accelerating reinforcement learning by composing solutions of automatically identified subtasks,” *Journal of AI Research*, vol. 16, pp. 59–104, 2002.

[41] M. Eiermann and O. Ernst, “Geometric aspects of the theory of Krylov subspace methods,” *Acta Numerica*, pp. 251–312, 2001.

[42] Y. Engel, S. Mannor, and R. Meir, “Bayes meets Bellman: The Gaussian process approach to temporal difference learning,” in *Proceedings of the 20th International Conference on Machine Learning*, pp. 154–161, AAAI Press, 2003.

[43] K. Ferguson and S. Mahadevan, “Proto-transfer learning in Markov decision processes using spectral methods,” in *International Conference on Machine Learning (ICML) Workshop on Transfer Learning*, 2006.

[44] M. Fiedler, “Algebraic connectivity of graphs,” *Czechoslovak Mathematical Journal*, vol. 23, no. 98, pp. 298–305, 1973.

[45] D. Foster and P. Dayan, “Structure in the space of value functions,” *Machine Learning*, vol. 49, pp. 325–346, 2002.

[46] A. Frieze, R. Kannan, and S. Vempala, “Fast Monte Carlo algorithms for finding low-rank approximations,” in *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pp. 370–378, 1998.

[47] M. Ghavamzadeh and S. Mahadevan, “Hierarchical average-reward reinforcement learning,” *Journal of Machine Learning Research*, vol. 8, pp. 2629–2669, 2007.

[48] R. Givan and T. Dean, “Model minimization in Markov decision processes,” *AAAI*, 1997.

[49] G. Golub and C. V. Loan, *Matrix Computations*. Johns Hopkins University Press, 1989.

[50] G. Gordon, “Stable function approximation in dynamic programming,” Technical Report, CMU-CS-95-103, Department of Computer Science, Carnegie Mellon University, 1995.

[51] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman, “Efficient solution algorithms for factored MDPs,” *Journal of AI Research*, vol. 19, pp. 399–468, 2003.

[52] D. Gurarie, *Symmetries and Laplacians: Introduction to Harmonic Analysis, Group Representations and Laplacians*. North-Holland, 1992.

[53] M. Hein, J. Audibert, and U. von Luxburg, “Graph Laplacians and their convergence on random neighborhood graphs,” *Journal of Machine Learning Research*, vol. 8, pp. 1325–1368, 2007.

[54] M. Herbster, M. Pontil, and L. Wainer, “Online learning over graphs,” in *Proceedings of the Twenty-Second International Conference on Machine Learning*, 2005.

[55] J. Hoey, R. St-aubin, A. Hu, and C. Boutilier, “SPUDD: Stochastic planning using decision diagrams,” in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 279–288, Morgan Kaufmann, 1999.

[56] R. Howard, *Dynamic Programming and Markov Decision Processes*. MIT Press, 1960.

[57] J. Johns and S. Mahadevan, “Constructing basis functions from directed graphs for value function approximation,” in *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 385–392, ACM Press, 2007.

[58] J. Johns, S. Mahadevan, and C. Wang, “Compact spectral bases for value function approximation using Kronecker factorization,” in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2007.

[59] T. Jolliffe, *Principal Components Analysis*. Springer-Verlag, 1986.

[60] P. Jones, M. Maggioni, and R. Schul, “Universal parametrizations via Eigenfunctions of the Laplacian and heat kernels,” Forthcoming 2007.

[61] S. Kakade, “A natural policy gradient,” in *Proceedings of Neural Information Processing Systems*, MIT Press, 2002.

[62] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM Journal of Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1999.

[63] A. Kaveh and A. Nikbakht, “Block diagonalization of Laplacian matrices of symmetric graphs using group theory,” *International Journal for Numerical Methods in Engineering*, vol. 69, pp. 908–947, 2007.

[64] R. Kretchmar and C. Anderson, "Using temporal neighborhoods to adapt function approximators in reinforcement learning," in *International Work Conference on Artificial and Natural Neural Networks*, pp. 488–496, 1999.

[65] H. Kushner and G. Yin, *Stochastic Approximation and Recursive Algorithms and Applications.* Springer, 2003.

[66] B. Kveton, "Learning basis functions in hybrid domains," in *Proceedings of the 21st National Conference on Artificial Intelligence*, pp. 1161–1166, 2006.

[67] J. Lafferty and G. Lebanon, "Diffusion Kernels on statistical manifolds," *Journal of Machine Learning Research*, vol. 6, pp. 129–163, 2005.

[68] S. Lafon, "Diffusion maps and geometric harmonics," PhD thesis, Yale University, Department of Mathematics and Applied Mathematics, 2004.

[69] M. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[70] A. Langville and C. Meyer, "Updating the stationary vector of an irreducible Markov chain with an eye on google's pagerank," *SIAM Journal on Matrix Analysis*, vol. 27, pp. 968–987, 2005.

[71] J. C. Latombe, *Robot Motion Planning.* Kluwer Academic Press, 1991.

[72] S. Lavalle, *Planning Algorithms.* Cambridge University Press, 2006.

[73] J. Lee and M. Verleysen, *Nonlinear Dimensionality Reduction.* Springer, 2007.

[74] J. M. Lee, *Introduction to Smooth Manifolds.* Springer, 2003.

[75] L. Li, T. Walsh, and M. Littman, "Towards a unified theory of state abstraction for MDPs," in *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pp. 531–539, 2006.

[76] M. Maggioni and S. Mahadevan, "Fast direct policy evaluation using multiscale analysis of Markov diffusion processes," in *Proceedings of the 23rd International Conference on Machine Learning*, pp. 601–608, New York, NY, USA: ACM Press, 2006.

[77] S. Mahadevan, "Proto-value functions: Developmental reinforcement learning," in *Proceedings of the International Conference on Machine Learning*, pp. 553–560, 2005.

[78] S. Mahadevan, "Representation policy iteration," in *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pp. 372–37, AUAI Press, 2005.

[79] S. Mahadevan, "Fast spectral learning using lanczos eigenspace projections," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2008.

[80] S. Mahadevan, *Representation Discovery Using Harmonic Analysis.* Morgan and Claypool Publishers, 2008.

[81] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial Intelligence*, vol. 55, pp. 311–365, 1992. Appeared originally as *IBM TR RC16359*, December 1990.

[82] S. Mahadevan and M. Maggioni, "Value function approximation with diffusion wavelets and Laplacian eigenfunctions," in *Proceedings of the Neural Information Processing Systems (NIPS)*, MIT Press, 2006.

[83] S. Mahadevan and M. Maggioni, "Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes," *Journal of Machine Learning Research*, vol. 8, pp. 2169–2231, 2007.

[84] S. Mahadevan, M. Maggioni, K. Ferguson, and S. Osentoski, "Learning representation and control in continuous Markov decision processes," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2006.

[85] S. Mahadevan, N. Marchalleck, T. Das, and A. Gosavi, "Self-improving factory simulation using continuous-time average-reward reinforcement learning," in *Proceedings of 14th International Conference on Machine Learning*, pp. 202–210, Morgan Kaufmann, 1997.

[86] S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Transactions on Pattern Analysis of Machanical Intelligence*, vol. 11, no. 7, pp. 674–693, 1989.

[87] S. Mallat, *A Wavelet Tour in Signal Processing.* Academic Press, 1998.

[88] D. Malsen, M. Orrison, and D. Rockmore, "Computing isotypic projections with the lanczos iteration," *SIAM*, vol. 2, nos. 60/61, pp. 601–628, 2003.

[89] S. Mannor, I. Menache, A. Hoze, and U. Klein, "Dynamic abstraction in reinforcement learning via clustering," *International Conference on Machine Learning*, 2004.

[90] A. McGovern, "Autonomous discovery of temporal abstractions from interactions with an environment," PhD thesis, University of Massachusetts, Amherst, 2002.

[91] M. Meila and J. Shi, "Learning segmentation by random walks," *NIPS*, 2001.

[92] C. Meyer, "Sensitivity of the stationary distribution of a Markov chain," *SIAM Journal of Matrix Analysis and Applications*, vol. 15, no. 3, pp. 715–728, 1994.

[93] A. Moore, "Barycentric interpolators for continuous space and time reinforcement learning," in *Advances in Neural Information Processing Systems*, MIT Press, 1998.

[94] R. Munos, "Error bounds for approximate policy iteration," in *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 560–567, 2003.

[95] E. Nelson, *Tensor Analysis.* Princeton University Press, 1968.

[96] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *NIPS*, 2002.

[97] A. Ng, H. Kim, M. Jordan, and S. Sastry, "Autonomous helicopter flight via Reinforcement Learning," in *Proceedings of Neural Information Processing Systems*, 2004.

[98] P. Niyogi and M. Belkin, "Semi-supervised learning on Riemannian manifolds," Technical Report TR-2001-30, University of Chicago, Computer Science Deparment, November 2001.

[99] P. Niyogi, I. Matveeva, and M. Belkin, "Regression and regularization on large graphs," Techncial Report, University of Chicago, November 2003.

[100] D. Ormoneit and S. Sen, "Kernel-based reinforcement learning," *Machine Learning*, vol. 49, nos. 2–3, pp. 161–178, 2002.

[101] S. Osentoski, “Action-based representation discovery in Markov decision processes,” PhD thesis, University of Massachusetts, Amherst, 2009.

[102] S. Osentoski and S. Mahadevan, “Learning state action basis functions for Hierarchical Markov decison processes,” in *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 705–712, 2007.

[103] R. Parr, C. Painter-Wakefiled, L. Li, and M. Littman, “Analyzing feature generation for value function approximation,” in *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 737–744, 2007.

[104] R. Parr, C. Painter-Wakefiled, L. Li, and M. Littman, “An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.

[105] J. Peters, S. Vijaykumar, and S. Schaal, “Reinforcement learning for humanoid robots,” in *Proceedings of the Third IEEE-RAS International Conference on Humanoid Robots*, 2003.

[106] M. Petrik, “An analysis of Laplacian methods for value function approximation in MDPs,” in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2574–2579, 2007.

[107] P. Poupart and C. Boutilier, “Value Directed Compression of POMDPs,” in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 2003.

[108] P. Poupart, C. Boutilier, R. Patrascu, and D. Schuurmans, “Piecewise linear value function approximation for factored Markov decision processes,” in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 285–291, 2002.

[109] W. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* Wiley, 2007.

[110] M. L. Puterman, *Markov Decision Processes.* New York, USA: Wiley Interscience, 1994.

[111] C. Rasmussen and M. Kuss, “Gaussian processes in reinforcement learning,” in *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 751–759, MIT Press, 2004.

[112] B. Ravindran and A. Barto, “SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes,” in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 2003.

[113] C. Robert and G. Casella, *Monte-Carlo Methods in Statistics.* Springer, 2005.

[114] K. Rohanimanesh and S. Mahadevan, “Coarticulation: An approach for generating concurrent plans in Markov decision processes,” in *Proceedings of the International Conference on Machine Learning*, ACM Press, 2005.

[115] S. Rosenberg, *The Laplacian on a Riemannian Manifold.* Cambridge University Press, 1997.

[116] S. Roweis and L. Saul, “Nonlinear dimensionality reduction by local linear embedding,” *Science*, vol. 290, pp. 2323–2326, 2000.

[117] S. Rusell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 2002.

[118] Y. Saad, *Iterative Methods for Sparse Linear Systems.* SIAM Press, 2003.

[119] B. Sallans and G. Hinton, "Reinforcement learning with factored states and actions," *Journal of Machine Learning Research*, vol. 5, pp. 1063–1088, 2004.

[120] B. Scholkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2001.

[121] P. Schweitzer, "Perturbation theory and finite Markov chains," *Journal of Applied Probability*, vol. 5, no. 2, pp. 410–413, 1968.

[122] P. Schweitzer and A. Seidmann, "Generalized polynomial approximations in Markov decision processes," *Journal of Mathematical Analysis and Applications*, vol. 110, pp. 568–582, 1985.

[123] J. Serre, *Linear Representations of Finite Groups.* Springer, 1977.

[124] R. St-Aubin, J. Hoey, and C. Boutilier, "Approximate policy construction using decision diagrams," *NIPS*, 2000.

[125] E. M. Stein and R. Shakarchi, *Fourier Analysis: An Introduction.* Princeton University Press, 2003.

[126] G. Stewart and J. Sun, *Matrix Perturbation Theory.* Academic Press, 1990.

[127] G. Strang, *Introduction to Linear Algebra.* Wellesley-Cambridge Press, 2003.

[128] D. Subramanian, "A theory of justified reformulations," PhD thesis, Stanford University, 1989.

[129] R. Sutton and A. G. Barto, *An Introduction to Reinforcement Learning.* MIT Press, 1998.

[130] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[131] J. Tenenbaum, V. de Silva, and J. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, 2000.

[132] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, pp. 215–219, 1994.

[133] Y. Tsao, K. Xiao, and V. Soo, "Graph Laplacian based transfer learning in reinforcement learning," in *AAMAS '08: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1349–1352, 2008.

[134] B. Turker, J. Leydold, and P. Stadler, *Laplacian Eigenvectors of Graphs.* Springer, 2007.

[135] P. Utgoff and D. Stracuzzi, "Many-layered learning," *Neural Computation*, vol. 14, pp. 2497–2529, 2002.

[136] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform.* SIAM Press, 1987.

[137] C. Van Loan and N. Pitsianis, "Approximation with Kronecker products," in *Linear Algebra for Large Scale and Real Time Applications*, pp. 293–314, Kluwer Publications, 1993.

[138] B. Van Roy, "Learning and value function approximation in complex decision processes," PhD thesis, MIT, 1998.

[139] G. Wahba, "Spline models for observational data," *Society for Industrial and Applied Mathematics*, 1990.

[140] C. Watkins, “Learning from delayed rewards,” PhD thesis, King’s College, Cambridge, England, 1989.

[141] Y. Wei, “Successive matrix squaring algorithm for computing the Drazin inverse,” *Applied Mathematics and Computation*, vol. 108, pp. 67–75, 2000.

[142] C. Williams and M. Seeger, “Using the Nyström method to speed up Kernel machines,” in *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 682–688, 2000.

[143] W. Zhang and T. Dietterich, “A reinforcement learning approach to job-shop scheduling,” in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1114–1120, 1995.

[144] X. Zhou, “Semi-supervised learning with graphs,” PhD thesis, Carnegie Mellon University, 2005.